MASTER IN COMPUTER SCIENCE

Database migration from relational databases towards NoSQL - Method and implementation

Corbaye, Bastien; Crespeigne, Romain; Minet, Martin

Award date:
2015

Awarding institution:
University of Namur

Link to publication

Université de Namur
Faculty of Computer Science
Academic Year 2014–2015

Database migration from relational databases towards NoSQL - Method and implementation

Bastien Corbaye
Romain Crespeigne
Martin Minet

Internship mentor: Dr. Jens Weber

Supervisor: ______________________ (Signed for Release Approval - Study Rules art. 40)
Dr. Anthony Clève

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the University of Namur

Abstract

This thesis draws a panorama of main technologies related to databases, from relational to non-relational, including NoSQL technologies but also NewSQL. They are compared and a critical eye is brought on advantages and disadvantages of each of them. Moreover, this thesis describes two approaches that complete each other to apprehend database migration: the transformational and co-transformational approaches. After providing that global view about databases and database migration, this thesis covers two distinct aspects.

In a first time, it aims at bringing a method to perform a complete migration from a relational database towards a non-relational database with a particular focus on Datomic as target technology. To do so, the method addresses three crucial elements.

The first element addressed is the structure. By combining two different processes that are the database reverse engineering and the database forward engineering, the migration method has for a goal to match the relational model and Datomic's data model in order to ensure that there is no loss in the information capacity. The second element addressed is the data. The method describes the complete data conversion process that consists of inserting the existing data within the target database while transforming it according to the structure changes and preventing information loss. The last element addressed is the application program(s). The method aims at providing a systematic approach that allows the existing application program(s) to communicate with the target database while balancing aspects such as the possibilities offered by the new technology, the security and the necessary effort to convert the program.

In a second time, this thesis focuses on a sample of NoSQL technologies which support Datomic. An evaluation of the performance of these databases has been made in order to choose the most appropriate database for the OSCAR system, in a future work. The benchmark “Yahoo! Cloud Serving Benchmark” has been used to evaluation the sample of NoSQL technologies.

Keywords: *Cassandra, Couchbase, Database migration, Datomic, DynamoDB, ETL, MongoDB, NoSQL, YCSB, Riak*

Preamble

We first wish to address our gratitude to all people who, directly or indirectly, contributed to both our experience in Canada - without which this work would make little sense - and to the achievement of this thesis itself.

We express our thanks to our supervisors from Canada and Belgium: Dr. Jens Weber, Department of Computer Sciences, University of Victoria and Dr. Anthony Cleve, Department of Computer Sciences, University of Namur. They have overseen the progress of our work during a whole year and allowed us not only to be able to present a thesis we are proud of today, but also to live an amazing life experience together at the other side of the globe.

We also wish to thank the Simbioses Research lab where we have been working for four months in a wonderful atmosphere and with great people, and our landlord to have welcomed us into her house during this period and for having taken care of us daily. Together, they helped make this internship as pleasant as fruitful.

At last, we would like to thank our families, friends and loved ones for having provided their support all those months and without whom none of this would have been possible.

Table of Contents

Glossary **13**

1 Introduction **17**

1.1 Motivations . . . 17

1.2 Problem Statement . . . 18

1.3 Scope of Work . . . 18

1.4 Research Questions . . . 19

2 Technological background **21**

2.1 Relational databases . . . 21

2.1.1 Relational model . . . 21

2.1.2 Relational database management system . . . 21

2.2 Non-relational databases . . . 22

2.2.1 NoSQL . . . 22

2.2.1.1 Overview . . . 22

2.2.1.2 Features . . . 23

2.2.1.3 Consistency Availability Partition Theorem . . . 24

2.2.1.4 Families . . . 25

2.2.1.4.a Key-value store . . . 26

2.2.1.4.b Document store . . . 27

2.2.1.4.c Column store . . . 28

2.2.1.4.d Graph database . . . 30

2.2.2 NewSQL . . . 34

2.2.2.1 Datomic . . . 35

2.2.2.1.a Data Model . . . 35

2.2.2.1.b Architecture . . . 37

2.3 Sample of NoSQL technologies . . . 40
2.3.1 AmazonDynamoDB . . . 40
2.3.2 Riak . . . 42
2.3.3 Couchbase . . . 43
2.3.4 MongoDB . . . 45
2.3.5 Cassandra . . . 46
2.4 Non-Relational versus Relational Databases . . . 48
2.5 CASE Tools . . . 49
2.5.1 CASE Tools classification . . . 49
2.5.2 Kinds of CASE Tools . . . 52
2.5.3 Advantages and Disadvantages . . . 54
2.5.3.1 Advantages . . . 54
2.5.3.2 Disadvantages . . . 54
2.6 DB-Main . . . 55
2.7 ETL tools . . . 56
2.7.1 Pentaho Data Integration . . . 57
2.7.2 Talend Open Studio . . . 57
2.7.3 CloverETL . . . 57
2.7.4 Jaspersoft . . . 58
2.7.5 Informatica PowerCenter . . . 58

3 State of the art 59
3.1 Database migration . . . 59
3.1.1 Transformational approach . . . 59
3.1.1.1 Schema conversion . . . 60
3.1.1.1.a Database reverse engineering . . . 60
3.1.1.1.b Database forward engineering . . . 65
3.1.1.2 Pivot model . . . 67
3.1.1.2.a Conceptual level . . . 68
3.1.1.2.b Logical level . . . 69
3.1.1.2.c Physical level . . . 70

3.1.1.3 Schema transformation 70

3.1.1.3.a Definition 70

3.1.1.3.b Semantical issues about schema transformations . . 72

3.1.2 Co-Transformational approach 74

3.1.2.1 Overview 74

3.1.2.2 Data conversion 75

3.1.2.3 ETL tools 76

3.1.2.3.a Extract step 76

3.1.2.3.b Transform step 77

3.1.2.3.c Load step 79

3.1.2.3.d Reconciliation 80

3.1.2.4 Program conversion 82

3.1.2.4.a Wrapper strategy 83

3.1.2.4.b Statements rewriting strategy 84

3.1.2.4.c Logic rewriting strategy 84

4 General overview **87**

4.1 Schema conversion 88

4.2 Data synchronization 90

5 Tools support **93**

5.1 Schema conversion 93

5.2 Data synchronization 93

6 Schema Conversion **95**

6.1 Database reverse engineering 95

6.1.1 Physical extraction 95

6.1.2 Logical extraction 96

6.1.3 Conceptualization 96

6.2 Database forward engineering 97

6.2.1 Conceptual analysis 97

6.2.2 Logical design 97

6.2.2.1 Adapt the attribute names (and quit types of entities) . . 98
6.2.2.2 Change attributes data type 98
6.2.2.3 Transform IDs 99
6.2.2.4 Transform relationships 100
6.2.2.4.a One-to-Many 101
6.2.2.4.b One-to-One 104
6.2.2.4.c Many-to-Many 106
6.2.2.5 Transformation plan 109
6.2.3 Physical design 109
6.2.3.1 Internal structure 110
6.2.3.2 External structure 110
6.3 Coding 111
6.4 Case study 112

7 Data conversion 121
7.1 Overview 121
7.2 Data conversion approaches 123
7.2.1 Data migrated manually beforehand 123
7.2.2 Data generated by new system afterwards 123
7.2.3 Data migrated by tools beforehand 123
7.2.4 Comparison 124
7.3 ETL tools 126
7.3.1 ETL tool selection 128
7.3.2 ETL tools and Datomic 133
7.4 Case study 134
7.4.1 CSV file input 137
7.4.2 Add field 138
7.4.3 Transform fields 138
7.4.4 Cassandra Output 139

8 Program conversion 143

8.1 Overview 143

8.2 Maintain both databases 145

8.2.1 Wrapper strategy 145

8.2.2 Statements rewriting strategy 146

8.2.3 Logic rewriting strategy 146

8.2.4 Global analysis 147

8.3 Maintain target database only 148

8.3.1 Wrapper strategy 148

8.3.2 Statements rewriting strategy 148

8.3.3 Logic rewriting strategy 149

8.3.4 Global analysis 150

8.4 Advocated method 151

8.4.1 General method 151

8.4.2 Building wrappers 152

8.4.3 Rewriting the program's logic 154

8.5 Case study 156

9 Performance **157**

9.1 Overview 157

9.2 Performance evaluation : Benchmarking 160

9.3 Yahoo! Cloud Serving Benchmark 163

10 Performance of a sample NoSQL technologies **165**

10.1 Overview 165

10.2 Relational Database 166

10.2.1 YCSB Benchmarking of MySQL 166

10.3 Document store 171

10.3.1 YCSB Benchmarking of MongoDB 171

10.3.2 YCSB Benchmarking of Couchbase 175

10.4 Column-Oriented Databases 180

10.4.1 YCSB Benchmarking of Cassandra 180

10.5 Short comparison 184

11 Conclusion 185

12 Future Work 189

12.1 Schema conversion 189

12.2 Data synchronization 189

12.3 NoSQL performance 190

13 Appendix A 193

13.1 MySQL 193

13.11MongoDB 195

13.12Couchbase 197

13.13Cassandra 199

Bibliography 208

Glossary

Aggregate in NoSQL
An aggregate is a set of related objects that you can treat as an unit [Lamllari, 2013].

Agile Software Development
Agile Software Development gathers a lot of development methods which fit with change promoting adaptive planning, evolutionary development, early delivery and continuous improvement.

American National Standards Institute (ANSI)
The American National Standards Institute is a private non-profit organization which has as goal to promote standards about products, processes, systems, and personnel in the United States.

Availability in CAP
The availability of a system measures its ability to keep working even if there is any problem. [Strauch et al., 2011].

Big data
Big data represents the phenomenon where the volume of data and its variety are overgrowing and data is produced more and more in real-time [Laney, 2001].

Business Intelligence
Business Intelligence (BI) designates tools, methods and techniques used to analyze data from multiple sources and process it to obtain meaningful information that will be used for strategic decisions.

Cluster
A computer cluster can be considered as a set of connected computers that work together and can be see as a single one system. Each one is to perform the same kind of task.

Consistency in CAP
The consistency of a system measures its ability to be "in a consistent state after the execution of an operation" [Strauch et al., 2011].

Data Integration

Data Integration (DI) represents the concrete methods, tools and techniques allowing data mapping from multiple sources towards a data warehouse.

Data Warehousing

Data Warehousing (DW) is the process of gathering multiple source data models and map them into a target data model. This process is performed at a high level of abstraction and is concretely supported by Data Integration techniques.

ETL tool

An ETL (Extract-Transform-Load) tool is a computer software that performs the tasks of extracting data, transforming it and loading it in a context of data warehousing or database migration.

First-order logic

"First-order logic is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. In first-order logic, a predicate can only refer to a single subject. First-order logic is also known as first-order predicate calculus or first-order functional calculus." [WhatIs, 2015a]

International Standards Organization (ISO)

The International Standards Organisation is an international standard-setting body which is composed of many other organizations in charge of standards.

Load balancing

Load balancing is to reduce the throughput on any computing resource by distributing workload across multiple ones.

Network Partition Tolerance in CAP

This is the ability to recover from the failure of a network device involving a network to be split.

OLAP

On-line Analytical Processing (OLAP) involves relatively low volume of transactions with regards to OLTP. But queries are often very complex and involve aggregations and the response time is very important and this kind of system is also strongly related to data mining.

OLTP

On-line Transaction Processing (OLTP) involves a large number of short on-line transactions (INSERT, UPDATE, DELETE). Its main goal is to be fast query processing maintaining data integrity in multi-access environments.

OSCAR System

The OSCAR System stores Electronic Medical Records (EMRs) designed to help improve health care from individual to population health levels while reducing costs [Oscar, 2015].

Relation

Relation R of degree n is a subset of the cartesian product S1 x S2 x ... x Sn where Sx are sets. Its elements are tuples of the same type and unordered in mathematics.

Transaction

A list of operations involving changes to the database if they all are correctly executed. If not, the transaction is aborted and nothing is updated into the database.

WebCrawler

WebCrawler is a metasearch engine which provides users to find images, audio, video, news, etc... from Google and Yahoo for example.

Chapter 1

Introduction

1.1 Motivations

Nowadays, data tend to become so big it is increasingly difficult to treat them with common tools. Furthermore, they are always faster generated, shared and updated, along with becoming less and less homogeneous. In other words, today's data is voluminous, varied and swift. That trend is called the Big Data phenomenon.

That phenomenon is a real concern for the industry, at the point that many organizations such as Amazon, Google and others decided to address it and to employ different means to store and share data. Those other ways are called NoSQL. Indeed, *traditional* relational databases were not able anymore to cope with their needs, greatly slowing the operation of those companies. However, enterprises specialized in online services are not the only ones impacted by the limits of relational databases. Many work domains see their needs and expectations in terms of data treatment grow with years, such as banking and health sectors. An increasing number of organizations of all sizes are concerned by the phenomenon and have to find solutions in order to maintain the quality of the services they propose.

A fundamental issue met by those actors is the gap between the two types of technologies, often keeping them stuck to their old system. Indeed, those new technologies are the most often used to begin new projects rather than maintaining and making the existing ones evolve. Thus, though companies decide to change the way they store data, the migration from a relational database towards a non-relational database is very uncommon and mostly performed by those that can afford a non-standardized process. This process requires a deep analysis to capture the match between both data models that are by definition different. Between the existing restrictive model and the new one with no or few constraints, a compromise has to be found.

Furthermore, there is currently no standardization regarding NoSQL databases. Even though multiple new technologies of that kind emerge, each brings its own way to apprehend that explosion of data. There exist technology families that share common characteristics, but no real consensus has been found and that classification makes it even more difficult to address the database migration. Indeed, MongoDB, DynamoDB, BigTable, Cassandra and others all possess their own data model and their own data manipulation language. That disparity does not exist for relational databases that respect the same data model and implement the same language (SQL). To build a consensus, it is necessary to go beyond technologies and reflect at a higher level of abstraction.

Finally, even if the Big Data phenomenon is growing and necessitates another mean to store data, NoSQL is not necessarily the absolute answer. Indeed, that kind of storage insures no or few consistency. Domains like healthcare do need high accuracy in the information they manipulate. Lives are at stake and it is not conceivable that a patient's data integrity can be put in jeopardy. Thus, a compromise between the old too restrictive and the new model must be found in order to be efficient while maintaining accuracy.

1.2 Problem Statement

The need for high volume data treatment shows SQL's weaknesses and highlights NoSQL technologies. However, a major issue emerges: the deep disparity between the relational and non-relational worlds. Fundamentally, those two types of technologies have different structures and architectures, relying on different principles. Where relational databases seek data consistency, NoSQL technologies relaxes it to aim at a greater availability.

Besides, standardization of the relational world has been made possible by the pioneers of the domain and its great diffusion within enterprises. Thence, relational databases align on a common data model and implement a common language, SQL. On the other hand, the non-relational world tends to break away from the different principles relative to this standard in order to address technological needs. Those being different depending on the context, linked NoSQL technologies do not always share the same objectives, whether at the technical level or at the data model level, thus impeding a standardization of the non-relational world.

1.3 Scope of Work

A common issue is shared by situations such as those described above. That issue involves both relational databases and non-relational databases. Our thesis will aim at covering as much as possible those different technologies to characterize and compare them on the logical side as well as the technical side. Indeed, this thesis will attempt to establish a genuine topography of existing NoSQL technologies.

Moreover, our fundamental goal is to build a bridge between those very different conceptions and technologies. We will detail what is a database migration, the steps and different methods that exist. Thus, we will define a method by exploring those that fit the best to the context of migrating from a relational database to a NoSQL database. This method is not specific to a particular technology but rather aims at being general in order to put aside the lack of standardization of that kind of data storage.

Finally, since our case study is a healthcare system, the target of our migration method must ensure accuracy in the data stored and manipulated. For that reason, a data model called *New SQL* will be presented.

1.4 Research Questions

1. Is it possible to migrate an existing database from the relational world to the non-relational world (NoSQL)?
    (a) If it is, how can that be achieved? What are the global steps to perform that migration?
        i. How to convert the database schema without suffering any loss of information?
            A. What are the necessary processes to perform that conversion?
            B. What is a schema's information capacity? What are the different ways to apprehend it while performing those transformations?
            C. Are there particular mechanisms that facilitate those transformations? If there are, how to use them concretely?
        ii. How to achieve the synchronization between existing source data and the target database?
            A. What are the elements involved in that synchronization?
            B. What are the steps to follow to run a target system up to date with no loss of information?
    (b) If it is not possible as it is, are there hypothesis or assumptions necessary to make in order for it to become possible?

2. How to define the performance of databases? How is the performance influenced? Why is it interesting to measure the performance of databases? How to evaluate the performance of databases? What are the techniques used for that?
   (a) Which type of the chosen technique is used for the evaluation of selected databases?
   (b) What are the results of this evaluation?

Chapter 2

Technological background

2.1 Relational databases

2.1.1 Relational model

Relational databases are based on the relational model elaborated by Edgar F. Codd in 1969. This model relies on the theory of relations from the mathematics. But the relations are quite perceived as tables with column(s) named. Each row represents an atomic tuple of a relation, their ordering does not matter and they are all different in content. However each of them can change over time unlike relations in mathematics [Codd, 1990].

2.1.2 Relational database management system

The management of relational databases is intended to share numerous data among numerous users. It is possible thanks to the manipulation of tables by a language based on the first-order predicate logic as the Structure Query Language (SQL). This is the standard from 1986 by American National Standards Institute (ANSI) and from 1987 by the International Standards Organization (ISO). This language is now used by all Relational Database Management Systems (RDBMS).

Also, Relational Database Management Systems must ensure data integrity with some features [Codd, 1990] to share data with users. Here are some of them:

- **Entity integrity**: Entity integrity is guaranteed by the primary key mechanism. This is a combination of column(s) where the value in each row identifies that row uniquely. This must have an unique non-null value for each row.
- **Referential integrity**: Referential integrity is supported by the foreign key feature. This is a combination of column(s) where the value in each row refers a primary key with the same structure. This must refer to an existing row in the parent table.
- **Uniqueness**: This constraint can be declared on set of columns and makes it unique. In other words, the value in each row identifies this row uniquely.

- **Not-null**: This constraint for a column requires that this one has no "null" value.
- **Check**: This mechanism defines a range of values which is possible for one or more columns.

But also, RDBMSs must support the transactions [Codd, 1990] while ensuring ACID properties to process them reliably [Gray et al., 1981]:

- **Atomicity**: A transaction either concludes entirely or not at all.
- **Consistency**: All operations will preserve a consistent state of the database. It means that every constraint is still correct after the completion of any transaction.
- **Isolation**: No matter the number of concurrent transactions, they are all executed as if they were one after the other.
- **Durability**: Once a transaction is committed, it cannot be repealed.

Though, RDBMSs have emerged three decades ago and have been developed by a lot of companies as Oracle, IBM and Microsoft for example.They have become the standard in the market of data stores. But, today, industry's needs have evolved and the Big Data has emerged [Laney, 2001]. RDBMSs need to scale up to meet the new needs. But relations and database normalization, which has deleting all redundancies into tables, impede this scaling [Needham, 2013].

2.2 Non-relational databases

The main principle of non-relational databases aims to reject what Edgar F. Codd promulgated, the non duplication of rows [Codd, 1990]. Although many critics are mentioned about this new kind of databases as being just a "Hype" for example [Strauch et al., 2011], non-relational databases are to address this Big Data phenomenon.

This section will be subdivided into two parts according to the two big classes of non-relational databases: NoSQL and NewSQL.

2.2.1 NoSQL

2.2.1.1 Overview

NoSQL is a pretty young technology (less than 10 years) and means *"Not Only SQL"* because NoSQL databases are not based on the relational model and they are not intended to replace the relational databases [Lamllari, 2013].

The goal is not to erase traditional databases but to do what they can not anymore, such as to support the Web 2.0 and all OLTP/OLAP-style applications involved. Indeed, Web 2.0 involves much more interactivity through an internal complexity of the technology [Rouse, 2015b] and by definition is used by thousands or even millions users. There are more and more reads and writes, more and more data [Cattell, 2010]. Facebook, Twitter and Amazon among others are mainly concerned.

Therefore, it is necessary to be horizontally scalable. This means adding new clusters with minimal resources [Jatana et al., 2012] to allow load balancing, performance and efficiency. That is what NoSQL databases aim to provide. Though, this kind of databases is more flexible than traditional ones [Veronika Abramova, 2014]. Moreover, the NoSQL technologies are designed to effectively operate replication and fragmentation of data with a simple data model. The ACID transaction model is inadequate to make way for a more flexible model [Hainaut, 2015].

The first databases referred as being NoSQL are the *BigTable distributed storage system* at Google aiming to store results from the WebCrawlers but also the *Dynamo technology* at Amazon [DeCandia et al., 2007a]. Since, many technologies compose NoSQL class of databases like MongoDB, riak, Infinispan and others.

2.2.1.2 Features

All NoSQL databases have three main characteristics that allow them to be the answer to Big Data phenomenon [Lamllari, 2013].

- **Schema flexibility**: They can contain a mix of structured, semi-structured and unstructured data.
- **High performance**: NoSQL technologies are developed to run on clusters thanks to use of aggregate that speeds up the operations.
- **Application development agility**: NoSQL technologies also make the AGILE development easier because they take less time to stand, run and go from concept to implementation because the focus moves to domain design.

Furthermore, [Gajendran, 2011] argues that NoSQL databases have some other common concepts. Here is a small list of these:

- **Sharding**: Sharding is used to partition records on different clusters under some common key.
- **Consistent hashing**: This mechanism uses the same hash function in order to hash the object and the node.
- **Map-reduce**: This concept is designed by Google to process large data sets. There are a map function, generates intermediate key/value pairs from processing of initial key/value pair, and a reduce function, merges all intermediate values associated with the same intermediate key.
- **MVCC**: This mechanism is used to provide concurrent access to the database. It guarantees a read-consistent view of the database, but resulting in multiple conflicting versions.
- **Vector clocks**: When data are distributed among different clusters in a asynchronous distributed system, a modification can affect the consistency of the database. This mechanism is used to maintain version of each node.

2.2.1.3 Consistency Availability Partition Theorem

In order to know more about non-relational databases, it is important to know its place in the data stores landscape. Though, Consistency Availability Partition tolerance (CAP) theorem has been mentioned for the first time by Eric Allen Brewer in a keynote at ACM's Principles of Distributed Computing [Brewer et Eric, 2000]. According to him, it is impossible to optimize consistency, availability and network partition tolerance at the same time but just two of them. CAP is now widely used by big companies as Amazon or Google because it settles the trade-off that they have to do when they build a DBMS [Strauch et al., 2011]. In other terms, this theorem allows them to combine as well as possible consistency and availability [Lamllari, 2013].

However, this theorem is often confused with ACID and BASE (Basically Available, Soft-state, Eventually consistency) principles although they do not treat the same thing. ACID and BASE are not about a trade-off but they are just a way of implementing database operations [DB, 2015][Lamllari, 2013]. Unlike ACID properties that promote consistency and isolation, BASE approach encourages the fact that the system basically functions everytime in a known-state eventually but not consistent all the time [Strauch et al., 2011]. In other terms, BASE approach promotes "availability, graceful degradation, and performance" [Brewer et Eric, 2000].

Though, E. Brewer divides the different systems into three different categories through his theorem as presented in the figure 2.1:

1. **AP systems**: Systems focus on high availability and network partition tolerance to keep working as much time as possible and being more scalable than other categories.
2. **CA systems**: Systems first focus on consistency to ensure data reliability and then availability.
3. **CP systems**: Systems focus on consistency with more scalability than CA to have reliability and flexibility at the same time.

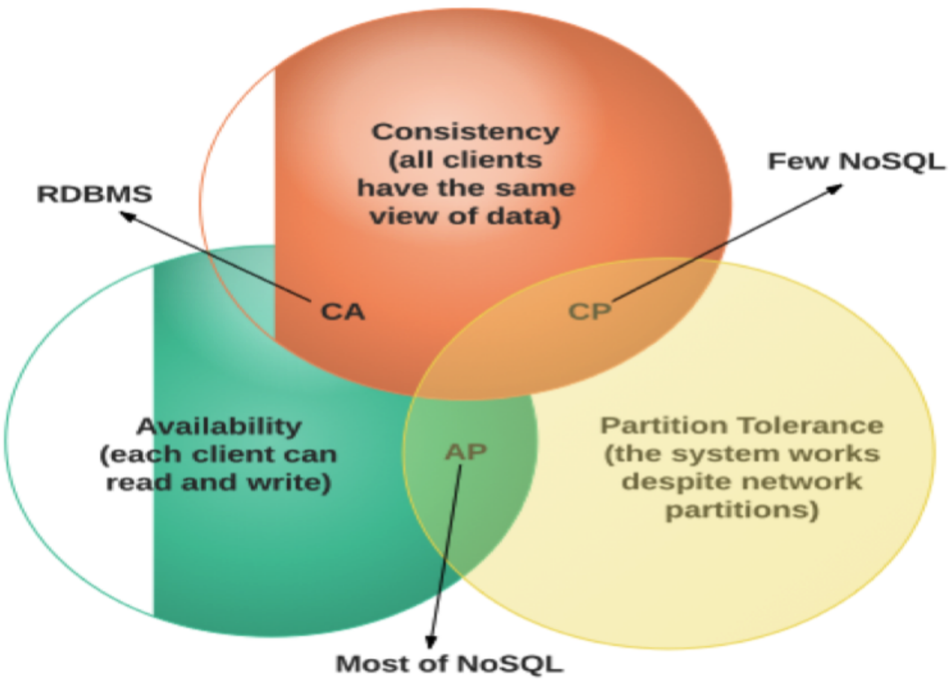

Consistency (all clients have the same view of data)
RDBMS
Few NoSQL
CA
CP
Availability (each client can read and write)
AP
Partition Tolerance (the system works despite network partitions)
Most of NoSQL

Figure 2.1: CAP systems [Lamllari, 2013]

2.2.1.4 Families

Among all AP and CP systems, there are different kinds of NoSQL storages divided into several families according to their specificities. Nowadays, there is no standard that properly lists all of them but here are the most popular types of NoSQL technologies:

1. **Key-value stores**: Data is represented as a collection of key-value pairs.
2. **Document stores**: Key-value where value is the document (semi-structured data).

3. **Column stores**: A form of key-value stores with an organization in a semi-schematized and hierarchical pattern through columns.

4. **Graph stores**: Databases organized in graphs where entities are nodes and relationships are edges.

2.2.1.4.a Key-value store

The first NoSQL technology covered in this thesis is the data storage named “Key-Value Store”. This is one of the simplest technology but the key-value store is not a less efficient and powerful model than others. This technology is based on a very simple system using a key and a value, which represents the data. The figure 2.2 illustrates this principle, where the arrow means that the key is associated with its value (the key “1a34” is associated with the value “Value 1”). This principle is called “the key-value” pair. This kind of storage refers to “hash tables” principle where the key is used as an index, which makes the key-value store faster than traditional relational database management systems. So, the key is the only way to retrieve data in the database. This data may be of various types, either a data type (such as Integer, String, Array, etc.) defined by the programming language, or an object [Nayak et al., 2013] [Cattell, 2010].

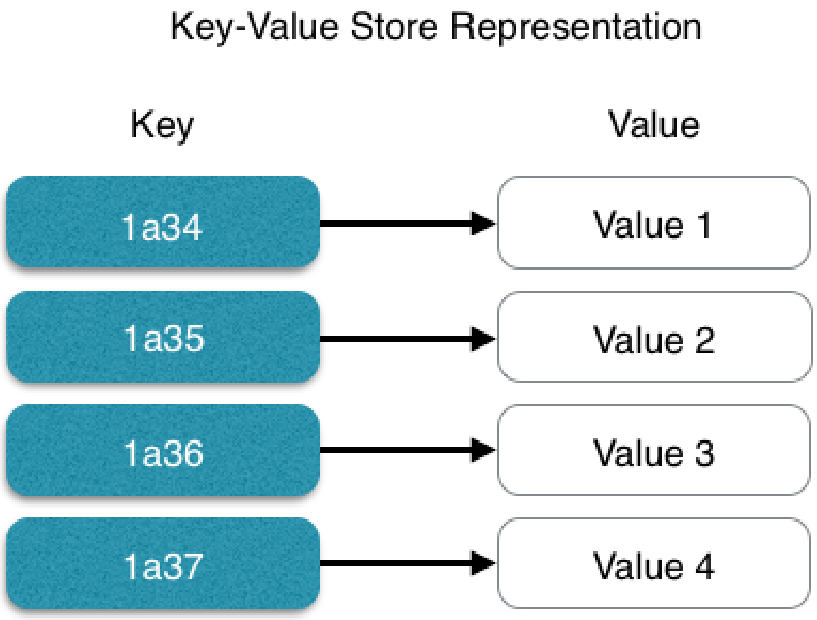


Figure 2.2: Key-Value Store Representation

Thanks to this very simple structure, key-value stores are completely schema-less. Thus, new values can be easily added, without affecting other stored data or the availability of the database. However, a drawback for being schema-less is that it is difficult to create customized views of data. The particularity of kind of store is that it uses the cache memory for complex queries or queries that are frequently requested. This allows to faster execute queries. While the other queries are simply stored in memory. The performances are better than SQL queries.

Moreover, the table 2.1 is a table thought by [Lamllari, 2013], which gives a summary of favorable situations for the use of key-value stores and situations where this technology is not suitable.

Best used for	**Not suitable for**
Big Data	Complex query and aggregation needs
Cache or Blob data	Relationships between sets of data
Session and Shopping cart data	Multi-operations transaction
Online Social Gaming	Key ranges processing
User profiles	

Table 2.1: Application Use Cases for Key-Value Stores [Lamllari, 2013]

2.2.1.4.b Document store

Another NoSQL technology is “Document store databases”. This type of database can be seen as a database that stores data in the form of documents. The term “document” refers to the concept of record in relational database but a document is more flexible due to the fact that it is schema-less. A document store supports more complex data than key-value store but it uses the same “key-value” mechanism [Cattell, 2010]. It contains key-value pairs within documents where keys have to be unique. A set of documents is called a collection. To ensure that all documents are unique, each of them encapsulates an “ID” key, which is a simple string or a string representing an URI or a path [Nayak et al., 2013]. This is represented in the figure 2.3 which is a comparison of a relational- and a document store representation.

In contrast to key-value stores, the value is a well-known document format such as XML, BSON, JSON, etc. Moreover, a document store database can contain multiple types of documents and nested documents [Cattell, 2010]. This type of database is perfect for systems which need to store document having specific characteristics than can not be stored in a table. Moreover, the structure of the fields in documents is dynamic, meaning that users can modify, add or delete fields from existing document [Bryden,].

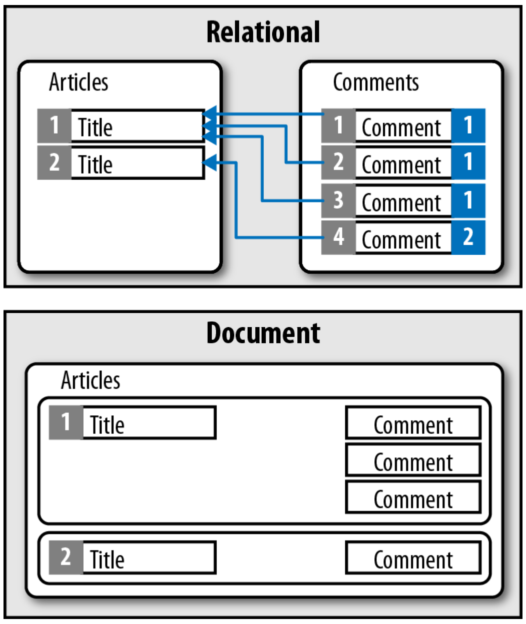


Figure 2.3: Document Store Representation

Moreover, the table 2.2 is a table thought by [Lamllari, 2013], which gives a summary of favorable situations for the use of document stores and situations where this technology is not suitable.

Best used for	Not suitable for
Big Data	Multiple document transaction
Event Logging	Ad-hoc queries
Content Management Systems	
Blogging Platforms	
Web and Real-Time analytics	
E-commerce Application	
Online Social Gaming	

Table 2.2: Application Use Cases for Document Stores [Lamllari, 2013]

2.2.1.4.c Column store

Column-oriented databases are part of a database type called "Extensible record store". This type had been motivated by Google with its system named "Big Table". "Big Table" is "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers" [Chang et al., 2006].

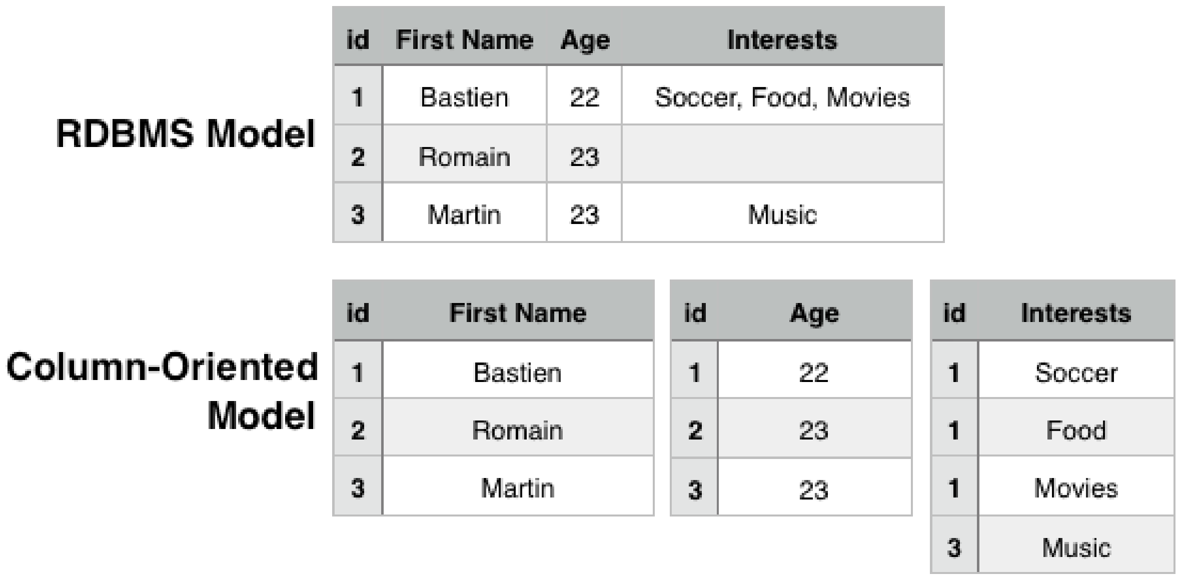
Thus, adding new machines to this system is facilitated by its architecture. It also offers several mechanisms such as persistence, consistency and fault tolerance. "Big Table" is used in a plurality of applications developed by Google itself such as Gmail, YouTube and Google Earth. Due to “Big Table”, Google ignited a craze for the “Extensible Record Store” database systems [Cattell, 2010].

Moreover, the “Extensible record store” database type groups two different types of database that have the capacity to solve same queries:

- **Row-Oriented databases:** handle rows (horizontal partitioning)
- **Column-Oriented databases:** handle columns (vertical partitioning)

The general principle of column-oriented databases is opposed to relational databases. While RDBMS store their records in rows one after the other, a column-oriented database stores a dataset of one column and all columns are individually treated [Abadi et al., 2009]. The figure 2.4 illustrates the difference between these two database types. Naturally, it is possible to rebuild one row from several columns but it requires some complex mechanisms.

The vertical partitioning of the database has been thought in the 70’ and studies had been made in order to find and to show advantages of this kind of model in comparison with the traditional row-based storages [Abadi et al., 2009]. However, several factors (like marketing needs and the small amount of data to be processed) enabled traditional row-base storages to keep an important place in society.



RDBMS Model

id	First Name	Age	Interests
1	Bastien	22	Soccer, Food, Movies
2	Romain	23	
3	Martin	23	Music

Column-Oriented Model

id	First Name
1	Bastien
2	Romain
3	Martin

id	Age
1	22
2	23
3	23

id	Interests
1	Soccer
1	Food
1	Movies
3	Music

Figure 2.4: Column-Oriented Database representation

Structuring the database depending on columns allows speeding up operations when these operations only affect a small group of columns. A DBMS only needs to read the values of columns required for processing a given query [Stonebraker et al., 2005] which allows not having to collect irrelevant data in memory. For example, take a table with four columns, each containing 1GB data. If the query only uses data from the third column, 1GB data have potentially been processed. Accordingly, read- and write operations on all columns are expensive costs. However, some databases use advantages of reading on small amount of columns and the benefits of storage optimized for write operations, as discussed in the section dedicated to "Cassandra" system [Steemann, 2012].

Because columns contain values of same type, it can compress data in order to retrieve more data when a query is executed. The compression enables to reduce disks I/O when read- and write operations are processed on columns. It can have an expensive cost for the CPU. Moreover, operations on this type of database are made by blocks, reducing the number of function call [Steemann, 2012].

2.2.1.4.d Graph database

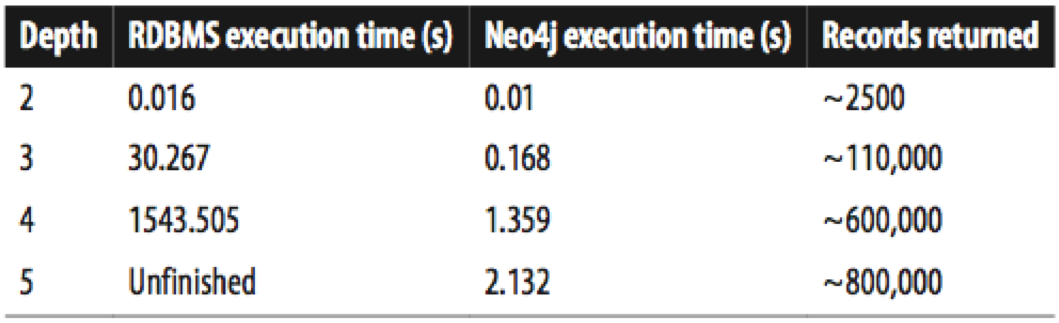
Before anything else, this section had been written according to [Robinson et al., 2013]. So, a graph is a set of nodes (to model things) and relationships (to model structure) between nodes that connect them.

In that way, a graph database is an online database management system with CRUD (Create, Read, Update, Delete) methods such as those found in relational databases. Graph databases are built for being used with transactional (OLTP — Online Transaction Processing) systems. Graph databases keep a set of proven features from the relational databases (such as transaction, ACID — Atomicity, Consistency, Isolation and Durability —, etc.). A lot of big companies like Google, Facebook, Twitter, etc. are using graph databases to manipulate and store information, especially to maintain account histories because it is easy to manipulate with graphs.

To explain the concept of graph databases, we focus on one of them, named Neo4J which is the most popular graph database and because there is no other section dedicated to graph database in this thesis. According to Neo4J developers, Neo4J is "embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables ". Furthermore, Neo4J is open source which is an advantage.

Relational and NoSQL databases issues from the point of view of graph databases

Relational and some NoSQL databases lack of relationships. For example, relational database need to join tables to find friends of mine at different depth ("depth 2" in the figure 2.5 means that the queries look for friends of friends of me, etc).



Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Figure 2.5: Finding friends at different depth

Figure 2.5 shows us the time needed in seconds to execute simple queries to find friends of mine. We can see that Neo4J execution time is faster than RDBMS execution time due to the complexity of joints in relational databases.

The other NoSQL databases store sets of disconnected values, documents, etc. So, it is difficult to use them for connected data or graphs. Without going into details, there are strategies for adding relationships to such stories but not for free.

Cypher Query Language

The chosen language for Neo4J is Cypher which is a language for describing and querying property graphs. Cypher allows expressing complicated databases queries. The clauses of Cypher look familiar, especially for SQL developers, that makes Cypher easy to learn. It is important to bear in mind that Cypher is optimized for reading and not for writing.

Labels, constraints and indexes are used to speed up queries. Due to label, it is possible to group nodes together and by doing so, create a constraint on this label to help enforce data integrity because. For example, it is possible to use unique constraints to ensure that property values are unique for all nodes with a specific label. Concerning indexes, we can create indexes that will improve the performance when looking for nodes within the set of nodes.

Architecture

Neo4J is peculiar because it can be run in server as well as embedded mode. These two modes have their advantages and their drawbacks. The first is the most common way of deploying the database and its advantages are:

- REST API
- Platform independence
- Scaling independence
- Isolation from application garbage collector (GC) behaviors

But Neo4J in server mode has a few drawbacks:

- Network overhead
- Per-request transactions

Whereas the second mode means that Neo4J runs in the same process as our application, its advantages are:

- Low latency
- Choice of APIs
- Explicit transactions
- Named indexes

But Neo4J in embedded mode has a few drawbacks :

- JVM Only
- Garbage collector behaviors
- Database life cycle

According to [Robinson et al., 2013], “Embedded Neo4j can be clustered for high availability and horizontal read scaling just as the server version. In fact, it is possible to run a mixed cluster of embedded and server instances (clustering is performed at the database level, rather than the server level).”, it shows that it is possible to mix embedded and server mode to improve our architecture, if necessary.

Another important thing of the architecture is clustering. In simple words, clustering consists of connecting two or more computers together in such a way that they behave like a single computer. There are several strategies for clustering in Neo4J which works with a number of slaves and a master. The first is replication which involves directing of write queries to the master and then replicated to slaves because it provides a bigger reliability, availability and consistency of data. Otherwise, it is possible to directly write queries to slaves but it is slower than writing directly to the master. A second strategy is to use queue to buffer writes and regulate load.

The load balancing concept is important to maximize throughput and reduce latency in clusters networks. However, Neo4J does not include native load balancing but it relies on the load balancing capabilities of the network infrastructure. It is important to separate flows of read and write queries. The majority of write queries should be directed to the master by a write load balancer while read queries traffic should be directed to the entire cluster by a read load balancer.

Furthermore, to maintain a high performance with a dataset that exceeds main memory space, Neo4J implements the solution of “cache sharding”. Cache sharing consists of directing each request to a database instance in a special cluster where a part of the graph required to satisfy the request is likely already in main memory.

Performance

Performance increases when dealing with connected data (unlike relational databases and some NoSQL stores). For example, in relational databases, performance decreases when the dataset becomes bigger because of join-intensive query while a graph database tends to remain constant even as the dataset gets bigger. Furthermore, execution time for each query is not concerned with the size of the entire graph, but rather the size of relevant part of the graph to satisfy the query.

There are three areas to optimize performance. The first is to increase the object cache, the second is to increase the percentage of the store mapped into the filesystem cache and the third is to invest in faster disks (SSDs or enterprise flash hardware).

The two first need to increase RAM memory. Without going into details, allocating RAM to object cache is more expensive per graph than allocating RAM to filesystem cache because graphs on object cache can be up to 10 times bigger than their on-disk representations. However, allocation of RAM (for object- and filesystem cache depends on the estimated size of the graph. According to figure 2.6, these different strategies have costs, there are trade-offs between performance and cost.

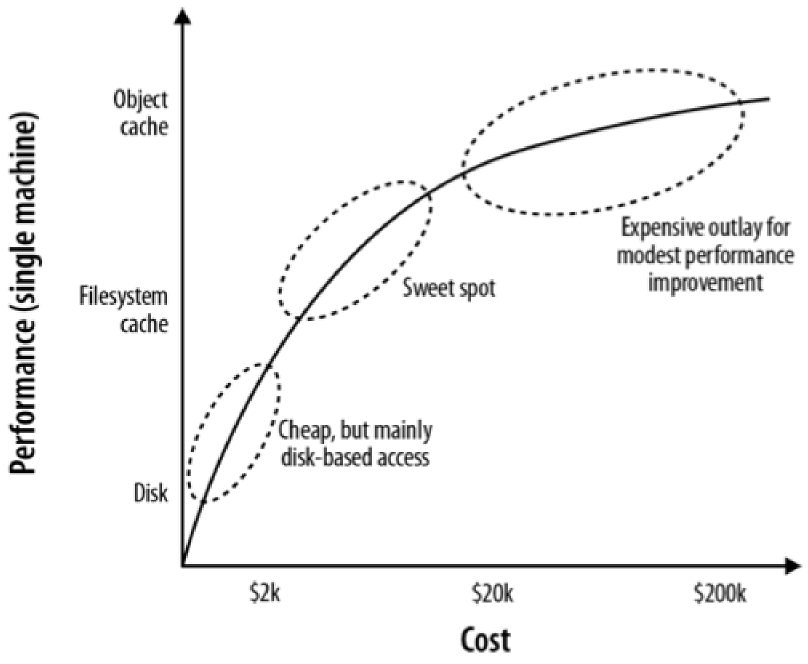


Figure 2.6: Trade-offs between performance and cost

That being said, each family can be categorized according to their performance, scalability, flexibility, complexity and functionality. Table 2.3 pictures how each one differs from the other. But also, the table highlights better flexibility and performance for NoSQL than relational databases.

	Performance	**Scalability**	**Flexibility**	**Complexity**	**Functionality**
Key-Value Stores	high	high	high	none	variable (none)
Column stores	high	high	moderate	low	minimal
Document stores	high	variable (high)	high	low	variable (low)
Graph databases	variable	variable	high	high	graph theory
Relational databases	variable	variable	low	moderate	relational algebra

Table 2.3: Classifications - Categorization and Comparison by Scofield and Popescu [Strauch et al., 2011]

2.2.2 NewSQL

Another emerging way to store data than NoSQL is called NewSQL. Even though ACID transaction model is not flexible enough [Hainaut, 2015], it is still relevant in some cases. That is why this new kind of databases is born to meet new needs but this time also has the goal to preserve ACID properties while ensuring scalability. In other words, NewSQL aims at merging the integrity of traditional databases with the scalability of NoSQL. This is made possible by three different approaches: a new database, a new MySQL storage engine or a transparent clustering [Lamllari, 2013].

This concept is relatively new and there is not a lot of documentation yet. But Datomic's data model used in this thesis is close with this family of data stores as described below.

2.2.2.1 Datomic

2.2.2.1.a Data Model

All databases are built on fundamental units. In the case of Datomic, there is only one unit, the atomic fact called Datom [Hickey, 2013]. A Datom is immutable. It means that it is impossible to update it. Instead, a new Datom should be added if the reality changed while the old stays in the database. Datoms are indexed and index in Datomic does not point the data but contains it. Moreover, storage and cache services are the distribution network of the index's data segment [Datomic, 2015c].
A Datom is identified by four properties:

- **Entity**
- **Attribute**
- **Value**
- **Transaction** (database time)

For example, let us consider that we want to store the fact that our new client is named « Henry Dupont », we would represent this as below:

```
[<e-id> <attribute> <value> <tx-id>]: [ 167 :client/name "Henry Dupont" 102 ]
```

In the light of this example, Datomic's data model is close with the key-value model even though this belongs to NewSQL stores.

On the contrary of relational databases composed by a plurality of constructions (tables, columns, rows, etc.), Datomic is only composed by facts as described above. It could totally seem "schema-less" but Datomic requires the definition of its attributes [Datomic, 2015c]. Attributes are generally defined in files of EDN format by data structures composed of lists of maps and not by DDL code like in SQL [Hickey, 2013]. The following is an example of “client’s name” attribute:

```
[ ;; client {:db/id #db/id[:db.part/db]
:db/ident :client/name
:db/valueType :db.type/string
:db/cardinality :db.cardinality/one
:db/fulltext true
:db/doc "A client’s name"
:db.install/_attribute :db.part/db} ]
```

Despite these attribute definitions, Datomic schema is so minimal that the applications written on the basis of its data model are not subject to the rigidity of applications written on the basis of relational databases [Datomic, 2015c]. For example, the rigidity due to “NULL” notion disappears because Datomic attributes do not have any constraint about their existence. Therefore, entities are open and sparse [Datomic, 2015d].

Before elaborating each property composing an attribute in more details [Datomic, 2015d]. A notion specific to Datomic did not define yet.
Datomic databases are divided into several partitions. They are intended to group semantically equivalent things improving the performance of queries. There are 3 partitions by default as pictured in the table 2.4.

Name	**Description**
:db.part/db	System’s partition
:db.part/tx	Transactions’ partition
:db.part/user	User’s partition

Table 2.4: Typology of default partitions in Datomic

System’s partition contains all attributes and subpartitions of a Datomic schema. Otherwise, user’s partition is used to store Datoms. A good practice consists of create partitions according to an area. Each partition gathers related Datoms and speeds requests up [Datomic, 2015c].

Though, it is now possible to understand all the properties that define attributes of Datomic as explained in the table 2.5.

Property	**Explanation**
:db/id [key]	[key] determines the partition where the attribute is stored
:db/ident: <namespace>/<name> (Required)	Specifying the attribute's name ":<namespace>/<name>"
:db/valueType <type> (Required)	Specifying data type of the attribute's values
:db/cardinality (Required)	Specifying value's cardinality of the attribute (one or many)
:db/doc	The attribute's documentation
:db/unique	Uniqueness constraint (upsertable or not)
:db/index <true|false>	False by default but if true, an index is created
:db/fulltext <true|false>	False by default but if true, a full text index is created
:db/isComponent	Specifying the attribute's reference to a sub-component
:db/noHistory	False by default but if true, it means that the previous values should not be stored anymore
:db.install/_attribute	Attach the attribute with a partition

Table 2.5: Attribute definition in Datomic

2.2.2.1.b Architecture

Datomic's architecture is quite different from classic databases due to its data model. Its general architecture is pictured in the figure 2.7 and here will be presented in more details each component of it.

As we can see in the figure 2.8, the application is seen as a Peer which calls Datomic Peer Library to query a database. Anything can be a peer from the moment he has datomic-specific application code [Datomic, 2015a].

Datomic's peer library is embedded in your peer (application) and is the intermediary between your application and the rest of the database. It is in charge of providing data access, query capability, submitting transactions and accepting changes from the transactor. But also, this library provides caching which is filled along the reading of facts from the Storage Service. In the end, this cache implementing a least-recently used policy represents a copy of many facts in the database. On the one hand, this feature allows reducing the network traffic for reading. On the other hand, this ability to query locally provides query results as simple data structures. There is no need to have abstractions nor object-relational mapping layers [Datomic, 2015a][Datomic, 2015b].

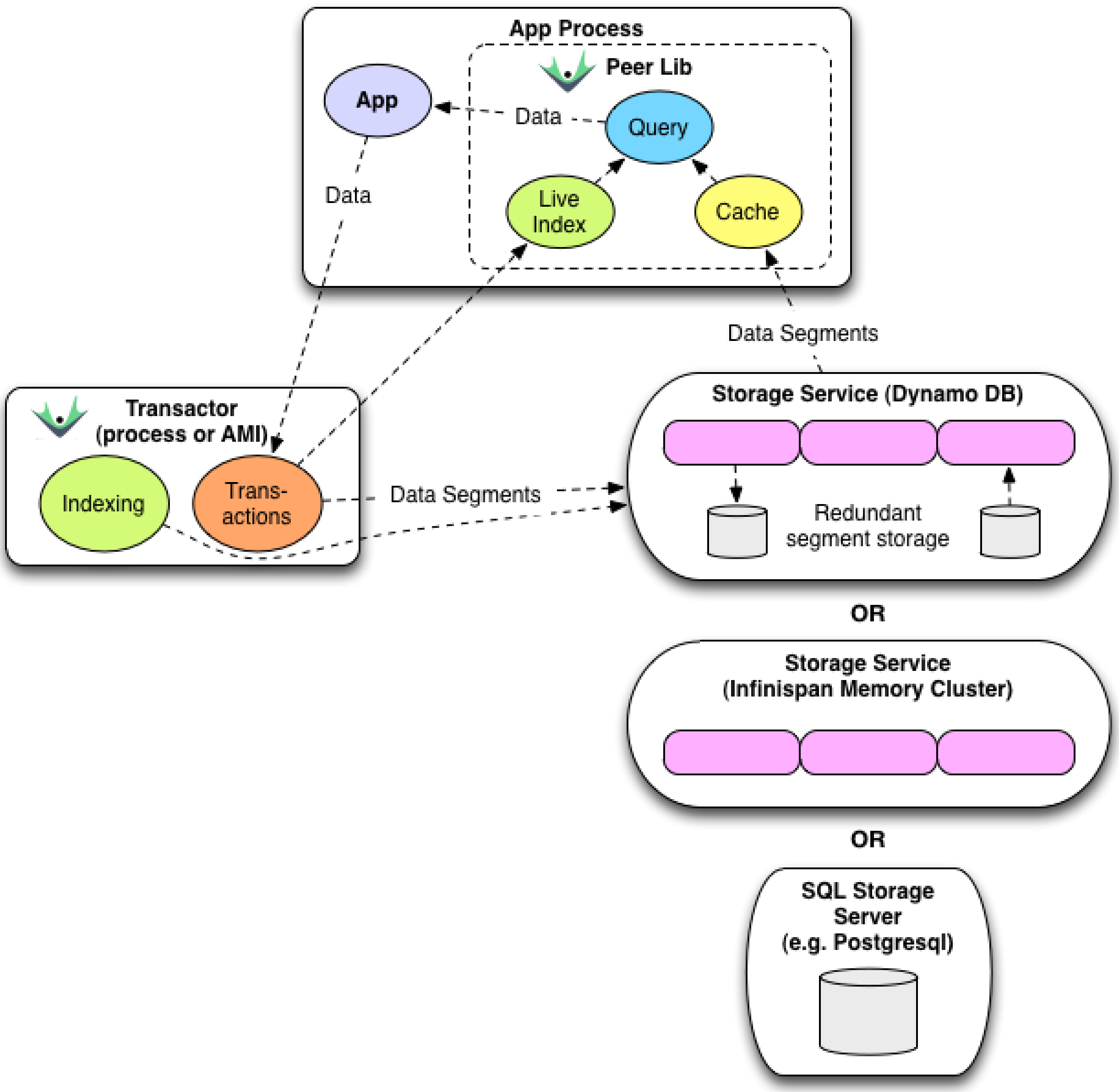

App Process
Peer Lib
App
Data
Query
Live Index
Cache
Data
Data Segments
Transactor (process or AMI)
Indexing
Trans-actions
Data Segments
Storage Service (Dynamo DB)
Redundant segment storage
OR
Storage Service (Infinispan Memory Cluster)
OR
SQL Storage Server (e.g. Postgresql)

Figure 2.7: Datomic architecture [Datomic, 2015a]

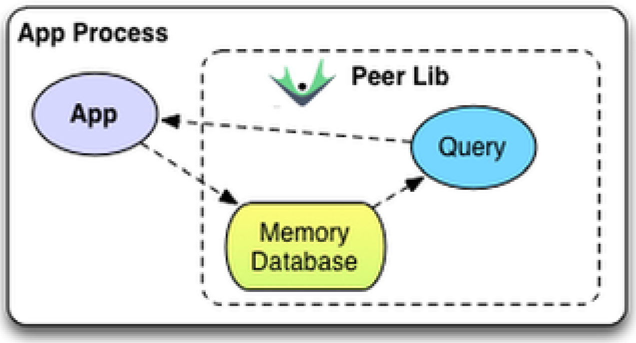


Figure 2.8: The Peer Library [Datomic, 2015b]

Also, another component very important in Datomic's architecture is the transactor as pictured in the figure 2.9. This part is requested when new facts are added into the Storage Service. Indeed, he stores them as transactions using ACID properties. Then, the transactor notifies all peers so that they can add the new facts into the cache managed by Datomic's Peer Library [Datomic, 2015a].

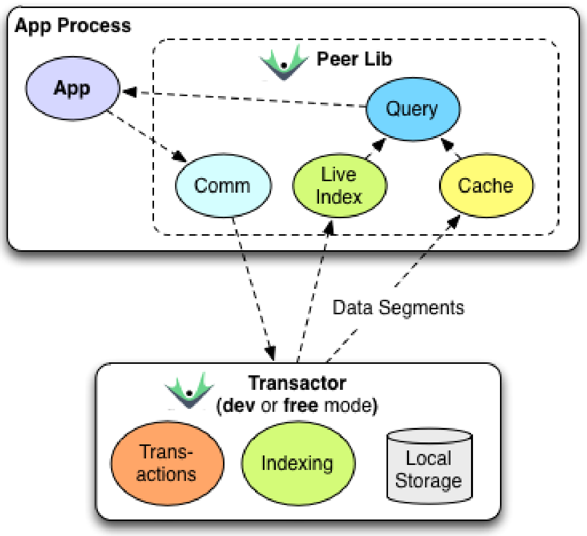


Figure 2.9: The Peer Library [Datomic, 2015b]

At this moment, all components of Datomic are described. In other words, Datomic is a kind of database management system without the database. This technology is not standalone and needs a storage service from outside except if the product is still in development. There is a special mode for that (i.e. dev mode) [Datomic, 2015b].

But when the application needs to be launched, at the same time higher performance and scalability are needed. There is no other way than integrating a storage service and Datomic supports a lot of them as SQL database, DynamoDB or a key/value store. Depending on configurations of the peer library and the transactor, they can bind to one of the external storage protocols [Datomic, 2015b]:

1. **ddb** by DynamoDB
2. **riak** by Riak
3. **cass** by Cassandra
4. **couchbase** by Couchbase
5. **inf** by Infinispan
6. **sql** by SQL database like PostgreSQL

In conclusion, there is a very particular concept which belongs to Datomic: Time. As a matter of fact, as explained in the sub-section 2.2.2.1.a, every fact is immutable. It means that there is no update but only adds which overpass the old facts. This immutability also provides a stable and consistent view as long as Peer needs one. Datomic allows keeping a complete versioning about all database states from the beginning. Though, the most recent set of facts is queried by default. But it is also possible to access data from the past. It is even possible to compare different facts coming from different times [Datomic, 2015a].

2.3 Sample of NoSQL technologies

This section gives a sample of NoSQL technologies which support Datomic. In that way, Amazon DynamoDB, Riak, Cassandra, Couchbase and MongoDB will be presented. A careful reader will note that MongoDB does not support Datomic. The reason of its presentation is that MongoDB is a reference for the document store family. So, it is an opportunity to be aware of what it is possible to do with it.

Moreover, those technologies are pretty young (less than 10 years) and may evolve since the writing of this thesis.

2.3.1 AmazonDynamoDB

Amazon is one of the leaders in terms of electronic commerce and cloud computing. The company provides its services to millions of users from around the world [Ama, 2015]. This requires high performance, reliability, efficiency, availability and being highly scalable in order to continuously expand the platform and the IT infrastructure [DeCandia et al., 2007b]. Naturally, all those aspects are extremely important for such a company because an issue with one of those aspects may have several negative impacts at the financial and customer levels. To avoid any problem, Amazon uses a highly decentralized and loosely coupled architecture. In that way, when a customer makes purchases on the platform and that the used cluster is failing, the customer can continue his shopping without realizing technical incidents.

Amazon offers a plethora of products and services in order to be ubiquitous on the electronic commerce, also called e-commerce. Over the years, the company expanded its activities. And it is in 2006 that the service called “Amazon Web Service” (AWS) was officially launched [His, 2015]. AWS has been created to provide online services to other websites or customer applications, by using cloud computing [Services, 2015c]. Moreover, AWS includes different services such as Amazon Simple Storage Service (Amazon S3) and amazon Elastic Compute Cloud (Amazon EC2) which are the most known. The first service offers secure, highly scalable and durable object storages to developers and IT teams [Services, 2015b]. While the second service is a web service that provides resizable compute capacity in the cloud [Services, 2015a]. Those different services can be used alone or with other AWS services. Furthermore, Amazon Web Services bills its services in accordance with the use and the rates may vary for each service. In 2014, AWS had a turnover of $4,6 billions.

In addition, Amazon Web Service has developed a NoSQL database management system, which is nothing other than Amazon DynamoDB. Unlike other NoSQL technologies presented in this thesis, this type of database is not open source[1]. Amazon DynamoDB is a part of the Key-Value store family which is fully managed in the cloud. This database is designed to provide fast performance, high availability, high scalability and reliability. Amazon developed the AWS Management Console to help its customers to easily create their databases [Services, 2012] [DeCandia et al., 2007b] [Nayak et al., 2013].

The availability of DynamoDB is achieved by the automatic data replication and data synchronization on three different data centers in different AWS regions. This allow accessing to data even when there are failures in the system. Due to the replication and synchronization of DynamoDB tables across different AWS regions, the data can be accessed from anywhere in the world in a minimum latency. Given that DynamoDB is designed to be an eventually consistent system, a data versioning is implemented [Services, 2012] [DeCandia et al., 2007b] [Nayak et al., 2013].

Moreover, DynamoDB provides a transparency in terms of storage and throughput to be highly scalable. This means that the amount of storage and throughput that you can dial up at a time are not limited. The database scales horizontally in order to add a new cluster to cope with increased demand of a particular service, which is allowed by the partitioning of data across multiple storage hosts. DynamoDB being schema-less provides the flexibility of the database [Services, 2012] [DeCandia et al., 2007b] [Nayak et al., 2013].

DynamoDB offers fast and predictable performance due to the use of Solid State Disks (SSDs) for all data. The access to the data is faster than if data are stored on traditional hard drives. DynamoDB is designed to maintain consistent at any scale.

Furthermore, DynamoDB's data model contains different related concepts such as tables, items and attributes. A database is a collection of tables, a table is a collection of

[1]Refers to Amazon's website to know the price

items and an item is a collection of attributes. An attribute can be single-valued or multi-valued. Due to the fact that DynamoDB is schema-less, it is not required to define all of the data types and attribute names in advance. This database only requires that tables have a primary key to create and maintain indexes for those primary key values. However, it is a benefit for some applications to have secondary keys in order to allow efficient access to data with attributes different than the primary key. This is the reason why DynamoDB provides secondary indexes [Services, 2012] [DeCandia et al., 2007b] [Nayak et al., 2013]. There are two types of secondary index:

Global secondary index: An index with a hash-and-range key (a key that is made of two attributes) that can be different from those on the table.

Local secondary index: An index that has the same hash key (a key that is made of one attribute which is generally the id) as the table, but a different range key.

Moreover, DynamoDB supports different data types:

Document types: List (which contains an ordered collection of values) and Map (which constains an unordered collection of attributes).

Multi-valued types: Binary Set, Number set and String Set.

Multi-valued types: Binary, Boolean, Null, Number and String.

2.3.2 Riak

Riak is an open source project developed by Basho and written in Erlang. This system has the particularity to be considered as a "key-value store" and a "document store". However, Basho mainly describes it as an advanced "key-value" store but it has more functionalities than the other key-value stores [Cattell, 2010]. Therefore, Riak associates a key with values such as a short string, a JSON object, video files, etc. This implies that Riak objects can have multiple fields like documents and objects are organized into buckets, which can be compared to a collection supported by document stores [Basho, 2015]. Unlike document stores, Riak has no query mechanisms and the only lookup can be made on primary key.

Moreover, Riak implements bucket types, which are essentially a flat namespace in Riak, in order to allow to store objects with the same key in different buckets. Buckets types are used for identical replication from different buckets to assign common configurations to those buckets. The replication is fundamental in Riak and is supported by a master-less mechanism, which means that all nodes play the same role in the system. Riak ensures the distribution of data across nodes using consistent hashing [Basho, 2015]. A map-reduce mechanism is also implemented in order to share the workload over all nodes in a cluster.

As seen before, NoSQL technologies can not guarantee the CAP theorem. Hence, Riak is designed to deliver a maximum read and write availability at the expense of strong consistency. So, the system offers an eventually consistency. In a highly-available system, it is unavoidable to have conflicts between replicas. The consistency can be adjusted by defining the minimum number of replicas that must successfully respond to a read or write request [Cattell, 2010]. Furthermore, Riak uses the vector clock mechanism in order to enable clients to increase the consistency of the system. So, the vector clock ensures at the read time that conflicts are resolved [Basho, 2015].

With regard to concurrency, Riak uses the MVCC mechanism to resolve conflicts. So, Riak determines, at the end of the operation, if there are potential conflicts between same stored data. The MVCC mechanism marks the new version of changed data and marks the old version as obsolete. Hence, there is no overwriting of the old data and multiple versions are stored in the system [Grolinger et al., 2013]. Moreover, the master-less mechanism enables each node to handle a read/write operation for any other node [Basho, 2015].

According to [Nayak et al., 2013], Riak is to be avoided for highly centralized data storage projects using unchanging data structures but is well suited to managing personal information of the user for social networking websites or Massively Multi-players Online Role Playing Games, to build mobile applications on cloud, etc.

2.3.3 Couchbase

In February 2011, Couchbase was designed as a result of the merger of Membase and CouchOne, founder of Apache CouchDB. Couchbase uses the forces of each of them and is an open source project. In that way, it is interesting to take a look at both of these NoSQL technologies.

Membase was developed as an open source project by the company bearing the same name. Membase provided a distributed key-value database with integrated memcached caching technology [Cou, 2011]. Membase provides persistence and replication to the memcached system, which is required to be qualified as a “data store” [Cattell, 2010].

The memcached system is a distributed memory object caching system offering high performance, designed for dynamic web applications. The particularity of the memcached system is that all clusters are into the same virtual pool of memory in order to optimally distribute memory. Without a memcached system, some clusters have more memory than needed. Therefore, this unused memory can be allocated to other clusters using a memcached system [Memcached, 2015].

Moreover, the force of this system is that Membase provides the ability to elastically add or remove servers in a running system, transferring data and dynamically redirecting requests in the meantime [Cattell, 2010]. This is provided by the fact that all nodes are equal into the cluster. Any node can replace any other node at any time. This allows to have consistent performance without additional cost.

With regard to the replication, Membase uses an asynchronous replication with a multi-master mechanism, where multiple nodes can process write requests, which are propagated to remaining the nodes. Unlike master-slave replication, all directions for the propagation are allowed [Grolinger et al., 2013].

Membase is used by some of world's busiest web applications, such as Zynga supporting several millions users per day.

Unlike Membase, Apache CouchDB is a document database developed as an open source project by CouchOne in order to facilitate the data synchronization across mobile, desktop and cloud platforms. Given that CouchDB is schema-free document model, it fits well for web applications, which may not have a fixed schema. CouchDB has the particularity that it uses JSON documents to store data and provides a RESTful interface to clients [Cattell, 2010].

Moreover, CouchDB uses the JavaScript language as query language [Nayak et al., 2013]. This database provides a "map-reduce" mechanism to distribute queries across several nodes but this mechanism has a significant cost. Concerning the scalability, CouchDB achieves it through asynchronous replication [Cattell, 2010]. This means that when a modification occurs, only changed fields are transferred across clusters and not the whole documents. While the durability is achieved due to the writing of all updates on documents and indexes to the end of a file on commit. The coupling of MVCC mechanism and the durability provides ACID semantics at the document level [Cattell, 2010].

Moreover, CouchDB does not guarantee the consistency because when the same document in two different clusters is modified, the system detects the conflict. To solve it, CouchDB saves the most recent version of the document and the other is saved in the document's history to allow the access to it if necessary [Couchbase,].

Therefore, Couchbase is a combination of a key-value store and a document store but is considered as a document store using JSON document format for storing data. Couchbase is designed for interactive web applications with low-latency requirements [Gosselé, 2014] [Couchbase, 2015]. Moreover, this system supports primary and secondary indexes on data using B-trees and implements almost all of the mechanism of Membase and CouchDB previously presented. In that way, Couchbase uses an asynchronous replication to distribute data across clusters with a multi-masters mechanism as used by Membase. The consistency is strong within one cluster but is not guaranteed with more than one cluster.

In addition, Couchbase implements optimistic concurrency control by using "Check and set", which performs a write operation only if the record has not been changed by another client since its last reading [Grolinger et al., 2013]. Couchbase does not implement multi-operations transaction while presented as an ACID-compliant.

2.3.4 MongoDB

According to the DB-Engines Ranking[2], MongoDB is the most popular NoSQL database. MongoDB is developed using C++ in an open-source project by the company 10gen Inc and released in 2009 [Nayak et al., 2013]. It is a schema-less document store database whose the main goal is to gather the fast and highly scalable key-value stores and feature-rich traditional RDBMSs such as secondary indexes, range queries and sorting [Strauch et al., 2011].

The architecture of MongoDB is based on the master-slave mechanism. This architecture allows the management of failover. When a master server goes down, MongoDB promotes a slave server, which is a backup server, to a master server. So, the master-slave structure is used for replication. The write operations are sent directly to the master server, which propagates modifications across slave serves of its cluster, while read operations can be sent to master and slave servers. While MongoDB does not provide consistency of a traditional DBMS, it can provide a "local" consistency on the modification of the document [Cattell, 2010]. According to [Hainaut, 2015], the distribution of documents uses a shard key, which is the "ID" key seen before and common to all documents of the same

[2]`http://db-engines.com/en/ranking`

collection. Thanks to a hash function, MongoDB is able to assign documents to the right node of the cluster [Hainaut, 2015].

With regard to the data model of MongoDB and as seen before, MongoDB stores documents with common characteristics in a collection and a set of collections forms the database. MongoDB server can contain several databases. Moreover, MongoDB stores data in a BSON format, which is a binary JSON format, and each document can have its own structure due to the fact that MongoDB is a schema-less database. Furthermore, a document can contain a combination of documents, so, a document can become very complex [V, 2014] [Strauch et al., 2011].

MongoDB provides indexes on document which avoid scanning all collections to find document that matches the query statement. Thus, indexes can improve the efficiency of read operations. Each collection has an index on the "ID" key and this index is unique [mongoDB, 2015].

2.3.5 Cassandra

Cassandra is a "Column-oriented" database system developped by Facebook. Facebook is the largest social networking platform with more than 1 billion of users each month [Statista, 2015]. Facebook has some strict requirements in term of performance, reliability and efficiency to meet the needs of users. Facebook must also be highly scalable because the number of users is steadily increasing. That was the leitmotif for the development of Cassandra by Facebook [Lakshman et Malik, 2008]. This system requires processing a large amount of write operations (update- and insert operations), becoming a requirement for Cassandra.

In 2008, the development of Cassandra was echoed by Apache Software Foundations to become an Open source project, developed using Java. Nowadays, Cassandra is still in charge of the management of database of more than 1,500 enterprises of all sizes.

Moreover, the data model of Cassandra is based on several denominations that differ from relational databases, based on "Big Table" by Google and Amazon's Dynamo system. In that way, Cassandra includes a set of column family which can be assimilated to the "table" concept of other databases. There are two types of column family: simple and super column families. The particularity of this super column family is that it may be seen as a column family within a column family. In other terms, this column family contains columns which are values of a column of a row [Hainaut, 2015].

Due to the Client API developed using Java, the data manipulations is done via 3 functions:

- get(table, key, columnName)
- insert(table, key, rowMutation)
- delete(table, key, columnName)

Regarding the architecture of Cassandra, it has been thought in order to offer a large availability, good performance and a good scalability, bearing in mind that the hardware and software may encounter failures. Cassandra offers a peer-to-peer architecture type with the particularity that all nodes are the same, meaning that the master-slave concept is not present. This allows to have no single point of failure like that can happen for architectures adopting the principle of master-slave [Datastax, 2013].

The data distribution is made across all nodes. It is automatic and transparent for users and developers. The data replication for each nodes is provided by Cassandra which are stored on other machines in the cluster. Thus, when the nodes of the system are down, it is possible to retrieve a copy of these nodes in the cluster in other to have a stable system. Moreover, when write operations are processed on a node A which is down, data are stored on another node B until the node A goes back in action [Datastax, 2013].

Write operations are cached in memory, which is flushed to disk [Cattell, 2010]. Thanks to this, the speed of write operations is better than the speed of read operations. That can be seen as a sort of disadvantage for Cassandra [Nayak et al., 2013].

According to [Datastax, 2013], Cassandra is more appropriate for the following cases:

- Real-time, big data workloads
- Social media input and analysis
- Online web retail
- Most write-intensive systems
- ...

2.4 Non-Relational versus Relational Databases

Having now explained each kind of database, their properties and the CAP theorem, here are all the differences on all levels from data model to data integrity explained in the table 2.6.

Features	**RDBMS**	**NoSQL**
Data model	Relational model	Domain driven
Data modeling drivers	Start from available data	Data access and update patterns
Transactions	Almost all support ACID	Atomic transactions at the aggregate level
Data types	Strongly typed	Loosely typed
Joins	Yes	Emulated at the application layer
Indexing	Primary, secondary and different storage types	Limited
Design Complexity	Persistence layer	Application layer
Role-Based access functionalities	Yes	No support
Data integrity	Responsible is Persistence layer	Shifted at the application layer
Consistency	Strong (also tunable by the application)	Eventual (also tunable by the application)
Schema mismatch detection	Database	Application/Data access layer
Query support	Complex and ad-hoc queries	Not suitable for ad-hoc and complex queries
Query language	SQL	REST, Client libraries, Protocol buffers
Query optimization	Responsibility of database	Responsibility is shifted to the application

Table 2.6: RDBMS versus NoSQL features comparison [Lamllari, 2013]

In addition, RDBMS, NoSQL and NewSQL can be compared in the light of this taxonomy in the table 2.7.

Features	**RDBMS**	**NoSQL**	**NewSQL**
Data model	Relational model	Domain driven	Relational model
Transactions	ACID (almost all)	BASE	ACID
Querying	SQL	REST, Client Libraries, Protocol buffers	SQL
CAP classification	CA	AP/CP	Mainly CP

Table 2.7: SQL vs NoSQL vs NewSQL according to CAP [Lamllari, 2013]

2.5 CASE Tools

CASE, which is an anagram for Computer Aided Software Engineering, tools are a set of application program in order to help development and maintenance of software projects [Tutorialspoint, 2015]. These projects are usually complex or on larger scale involving many software components and people [Rouse, 2015a]. Since the 1970's, CASE tools are playing a major role in improving software productivity and quality and in supporting the software development process [Baik, 2000]. The use of CASE tools allows developers, testers, managers,... to have a common view of the project at each stage of development. This is facilitated by a central repository which contains common, integrated and consistent information from any phase of the system life development cycle supported by the CASE tool [Not, 2008a]. This is why there are a large interest in such tools.

At the begin of the 1990's, the annual worldwide market for such tools was $ 4.8 billion and tripled in 5 years [Jarzabek et Huand, 1998]. It shows us that there was a real demand in these kinds of tools from companies, so, several CASE tools were developed. Naturally, several factors contribute to choose the appropriate CASE tools such as the economic status of the company or its area [Kemerer, 1992]. Although there was a large interest in CASE tools, a lot of companies had difficulties to use them because it required a large effort to learn its specific syntax.

2.5.1 CASE Tools classification

There are a plurality of classifications of CASE technologies in literature [Loucopoulos, 1995]. The first classified CASE in four categories, namely, language-centred (based on a programming language), structure-centred (based on the idea of an environment generation), toolkit environments (set of tools which supports the programming phase of the development) and method-based (focused on a specific method for the development of software systems).

A second classification [Sommerville, 2008], introduced by [Fuggetta 1993], proposes that CASE technologies should be classified into three categories, based on a framework and depicted in the figure 2.10 below:

- **Tools:** support only specific tasks in development of the system such as the verification of the consistency of a design, the compilation of a program, the comparaison of test results, etc. In general, Tools may be general-purpose, stand-alone tools (e.g. a word-processor) or may be grouped into workbenches.

- **Workbenches:** support one or more process phases or activities such as specification, design, etc. Workbenches generally integrate one or more tools with different degree of integration.
- **Environments:** support a large part software process. They normally include several integrated workbenches.

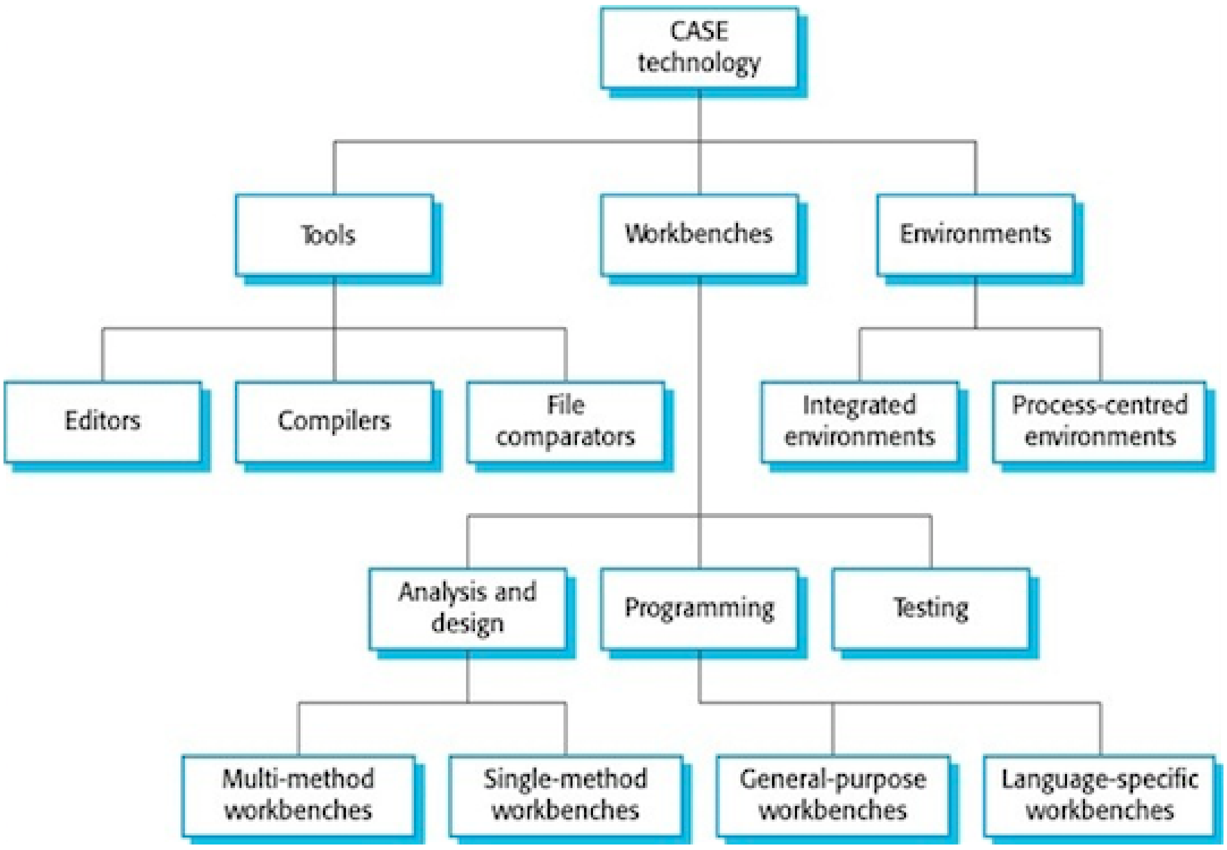


Figure 2.10: Tools, Workbenches and Environment [Fuggetta 1993]

The figure 2.10 shows us some examples of processes supported by each previous categories.

Nowadays, CASE tools are classified into 3 different categories according to their activities and in reference to the Waterfall Model in the figure 2.11 below and is the most popular classification [Loucopoulos, 1995] [Not, 2008b]:

- **UPPER CASE Tools:** Support the analysis and design phases of the system development life cycle and ignore the implementation of the system in an automatic manner. This name is due to the fact that these phases are at the top of the waterfall model.
- **LOWER CASE Tools:** Support the implementation and the maintenance phases of the system development life cycle and ignore the requirements and design of the

system. This name is due to the fact that these phase are at the bottom of the waterfall model.

- **INTEGRATED CASE Tools:** Support all stages of the lifecylce and provide the functionality of both upper and lower CASE tools. Hence, integrated CASE tools focus on all features of the system development life cycle.

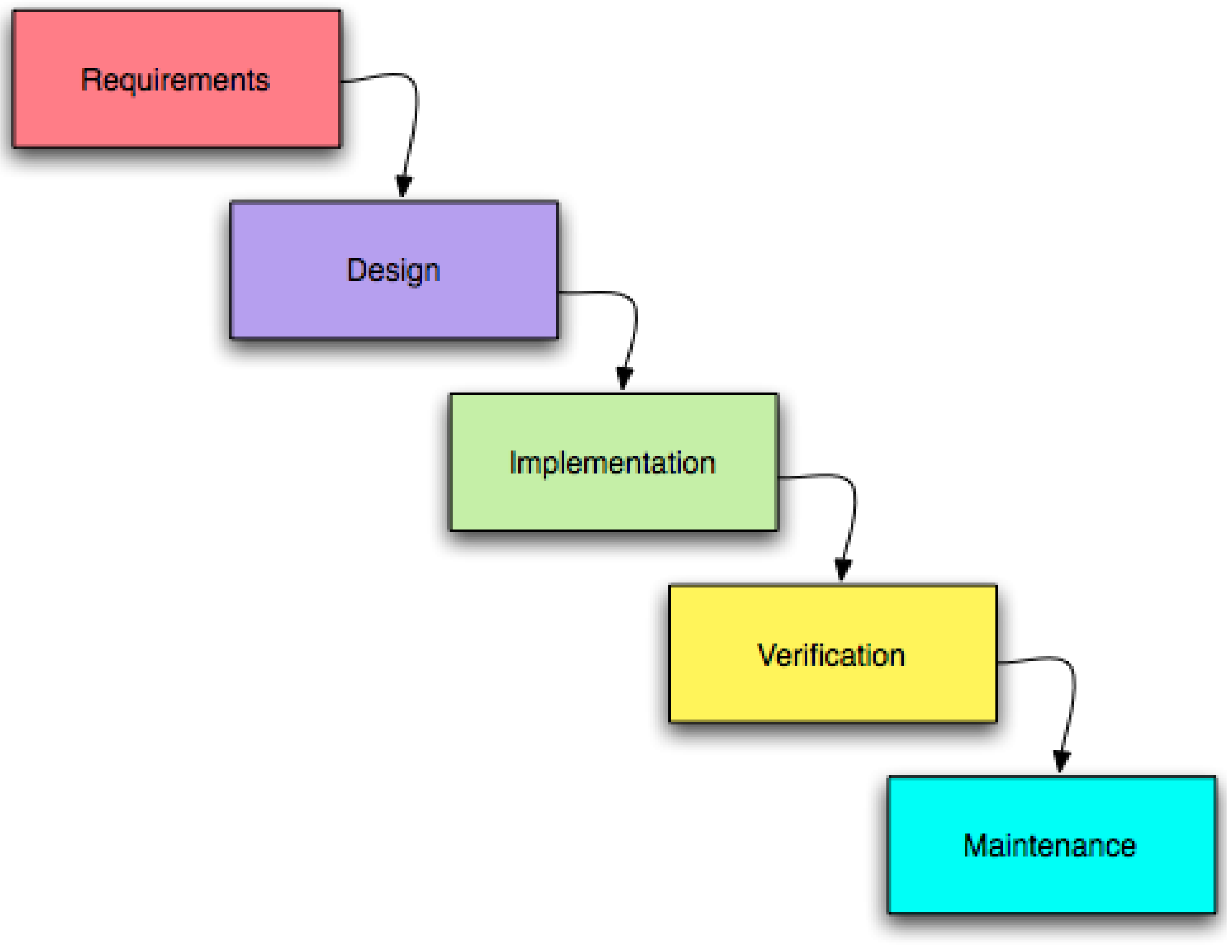


Figure 2.11: Waterfall Model

In short, the waterfall model (represented in the figure 2.11 [Rouse, 2007b]) is the first process model introduced in Software Engineering and ensures that the project will be a success. This model is divided into separate phases (see the figure 2.11), which are sequential and linear [ISTQB, 2015], and each phase must be completed before the beginning of the next phase.

The figure 2.11 shows us that upper CASE tools focus on the mission, objectives, strategies, resources, operational plans, etc of the system. These processes are only at the early stages of the development of the system.

Concerning lower CASE tools, they focus on the conceptual model of the system to generate an executable form of it from different algorithms. For example, it is possible to generate a relational database schema, to normalize database relations and to generate the SQL code from specifications. The next section will present a software called DB-Main, which is capable to do this kind of generation [Loucopoulos, 1995].

2.5.2 Kinds of CASE Tools

There are a plethora of different sorts of CASE Tools. This subsection will present some of these categories with a short explanation (all of these come from [Tutorialspoint, 2015]).

- **Diagram tools:** Diagram tools are used to graphically represent components, data and control flow of the system among various software components and system structure, generally without effort. These tools allow creating a flowchart which is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

- **Process Modeling Tools:** Process modeling is a method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it like the requirement of software product. For example, EPF Composer which is a framework for Eclipse. This framework aims at creating a customizable software process and supports a large variety of types of project [EPF, 2014].

- **Project Management Tools:** Project Management Tools are used for project planning, cost and effort estimation, project scheduling and resource planning. In project management, the execution of every mentioned step in software project management must be strictly respected by managers. Moreover, these tools help in storing and sharing project information in real-time throughout the organization. For example, GanttProject is a free project management software [Gan, 2015].

- **Documentation Tools:** Documentation in a software project contains all information of all phases of the system development life cycle from the beginning to the end of the project. Documentation tools are used to generate documents for (technical- and end-) users. Technical users refer to system manual, reference manual, training manual, installation manuals, etc. because they generally are in the development team. The end user documents describe the functioning and how-to of the system such as user manual for everyone who will use it. For example, Doxygen is a free licensed documentation generator able to produce software documentation from the source code of a program [Dox, 2015].

- **Analysis Tools:** Analysis tools are used to collect requirements from all placeholders of a project and to automatically check if there are any inconsistencies, data redundancies, inaccuracies in the diagram, etc. There are different tools for different purpose, for example, Accompa is used for requirement analysis whereas Visible Analyst is used for a total analysis.

- **Design Tools:** Design tools are used by the designer to conceptualize the structure of the software. It exists a "Computer-aided design" (CAD) to help a designer to create, modify, analyze or optimize the design. There are a plethora of design tools.

- **Programming Tools:** Programming tools are computer programs used by developers to create, modify, maintain, test, etc the source code of a project. These tools consist of programming environments like an integrated development environment (IDE). For example, Eclipse which is an essential tools for any Java developer [Ecl, 2015].

- **Prototyping Tools:** A prototype is used to provide an initial look of the product and how it will work by simulating of a few functionalities. Prototypes have different level of fidelity (low-, medium-, high-). Typically, the low-fidelity prototype is a paper prototype, focused on the interaction aspect. The medium-fidelity prototype is a clickable wireframe and the high-fidelity prototype is a faithful representation of the interaction behaviour and the look and feel of the system. This prototype can be fully programmed.
  Prototypes tools are used to help a team to quickly build prototypes due to the existing information. For example, Mockup Builder is a wireframe and create a prototype from requirements [Moc, 2013].

- **Web Development Tools:** Web Developments tools are used to help web developers to test and debug their source codes. Moreover, these tools assist web developers to design their interface. They can directly see what they are developing. For example, Fontello, Foundation 3 are Web Developments tools.

- **Quality Assurance Tools:** Quality assurance in a software organization is any process of checking to see if the software product being developed complies with requirements and organization standards [Rouse, 2007a]. Quality Assurance Tools is a set of control and software testing tools. For example, AppsWatch is a testing tool for performance, a web testing, etc [App, 2015].

- **Maintenance Tools:** After the delivery, a software need to be maintained and can need some modifications. Logging, error reporting techniques, etc. are useful for the maintenance of a software. Maintenance tools provide these techniques.

- **Data Modeling Tools:** Data modeling is often the first step in database design due to an analysis of data objects and their relationships to other data objects [Webopedia, 2015]. So, designers create a conceptual schema of data objects related to each other. The data modeling of a database starts from a conceptual model to a logical model to a physical schema. Data modeling tools are used to help designers to do this. For example, DB-Main is a free tool for data modeling and data architecture (See the next section to have more information about it).

This list of tools is not complete but provides an indication of the kinds of existing tools.

2.5.3 Advantages and Disadvantages

This subsection will present a non-exhaustive list of advantages and disadvantages of CASE tools [Petruska, 2011].

2.5.3.1 Advantages

- **Time saving:** CASE tools have an important advantage because they can save time to complete different tasks of the system development life cycle.
- **Better documentation:** Due to documentation generation tools, there is a better documentation which is produced at every phases of the system development life cycle.
- **Accurancy:** CASE tools can better meet the user needs and requirements. They can detect different errors and this is important because a correction requires less effort when it is done at an early stage.
- **Non-programmer:** CASE tools used in programming allow a person without programming background to develop a program.
- **Maintenance:** The cost in maintenance is reduced due to CASE tools used in other stages of the system development life cycle. Indeed, there are better analysis, automatic code generation, etc.

2.5.3.2 Disadvantages

- **Cost:** An important disadvantage of CASE tools is the cost which is not cheap. Concerning costs, many factors come into play such as hardware and systems, software, training and consulting.

- **Customization:** Some CASE tools can be difficult to customize because developers of these tools do not allow it.

2.6 DB-Main

Before the development of DB-Main [Englebert et al., 1995], CASE tools often provided partials solutions to the engineering problems of the databases. There were some weakness in those tools, e.g., ignorance of non-functional requirements, a lack of flexibility, ignorance of some processes, etc. The idea of a tool such as DB-Main was born from these gaps, to have a tool which will pay attention to non-functional requirements such as optimization, to flexibility and some key processes such as maintenance.

Nowadays [DB-, 2015], DB-MAIN was developed and it is a free[3] tool for data modeling and data architecture. This tool is developed in C++ with the widget toolkit “wxWidgets” and runs on Windows, Linux and Mac OSX. The purpose of this tool is helping developers and analyst in different data engineering processes. Here is a list of these processes:

- Design processes: requirement analysis, conceptual design, normalization, schema integration, logical design, physical design, schema optimization, code generation.
- Transformations: schema transformation, model transformation, ETL.
- Reverse engineering and program understanding: schema analysis (COBOL, CODASYL, IMS, IDMS, SQL, XML, ...), code analysis, data and data flow reverse engineering.
- Maintenance, evolution and integration: database migration, database evolution, impact analysis, database integration and federation, data wrapper design and generation.
- And many other domains like temporal and active databases, datawarehouse, XML engineering, ...

DB-Main allows users to develop new functions and extend its repository due to possibilities to add new plugins. Moreover, DB-Main contains a plethora of functions used in data modeling, such as code generator (SQL, MySQL, etc.) and a JDBC extractor (which extracts database structures to a JDBC driver), etc.

[3]It is possible to buy license for extra functionalities such as XML extractor, etc. Go to `www.db-main.be` for more information.

2.7 ETL tools

Many Extract-Transform-Load tools are currently available on the market, along with varied comparative reviews covering part of the tools. Since constituting an exhaustive comparative list of them would go beyond the scope of this thesis, we will present the principal tools the most often cited in the literature.

Whether in the scientific articles or in industrial papers, the following tools are considered the most common ETL tools systematically compared [Yousuf et Rizvi, 2011] [Thota,] [Jovanovic, 2015]:

- **Pentaho Data Integration**
- **Talend Open Studio**
- **Clover**
- **Jaspersoft**
- **Informatica PowerCenter**
- **Inaplex Inaport**
- **Oracle Warehouse Builder**
- **Oracle Data Integration**
- **IBM Information Server**
- **IBM DataStage**
- **Microsoft SQL ServerIntegration Services**
- **SAP BusinessObjects Data Integrator**

Among those tools, we only retained tools that provide a direct transition towards NoSQL storage services compatible with our target database technology Datomic (i.e. DynamoDB, Riak, Cassandra, Couchbase and Infinispan). Indeed, even though Datomic supports SQL storage systems, we do not consider this kind of system an ideal target for the type of migration depicted in this thesis and thus do not take them into account in this overview.

2.7.1 Pentaho Data Integration

Pentaho is a company specialized in open source Business Intelligence (BI). It provides multiple BI-related softwares in two editions : an Enterprise Edition and a Community Edition. Versions differ in the amount of functionalities they offer and the support of the product.

Pentaho Data Integration, also named Kettle, consists of multiple functionalities, among which a performing ETL tool relying on a Graphical User Interface (GUI). The ETL process is centred around two key-concepts : *transformations* and *jobs*. A transformation is an individual ETL process as depicted in the Data Conversion part. Transformations can be combined and scheduled inside jobs. Pentaho Data Integration is known for its wide range of connectors allowing data migration from and towards many types of sources [Pentaho, 2015].

2.7.2 Talend Open Studio

Talend is a company specialized in data management and data integration. It proposes a wide range of products in that field, including Big Data Integration, Cloud Integration, Data Integration, Application Integration and Master Data Management in two editions : an Enterprise Edition and a Talend Open Studio Edition. Versions differ in the amount of functionalities they offer and the support of the product.

Talend Open Studio Data Integration is among others an ETL tool focusing on ease of use and thus providing an intuitive interface combined with a lot of connectivity possibilities. The tool's key-concept is the representation of data through metadata stocked in a repository accessible to the user [Talend, 2015].

2.7.3 CloverETL

CloverETL is an ETL tool declined in multiple editions : the free Community Edition, the Designer Edition, the Server Standard Edition, the Server Corporate Edition and the Cluster Edition, each version increasing the number of functionalities included. CloverETL is based on Java and is consequently available as either an Eclipse plug-in or a standalone application. The tool represents data transformations as dataflows displayed in a graph composed of edges and pipes, the data being transferred and transformed from one component to another through a pipe [Clover, 2015].

2.7.4 Jaspersoft

TIBCO Software Inc. is a company specialized in Business Intelligence. It offers various products including reporting and analytics tools, a cloud solution for BI, mobile tools and a Big Data Business Intelligence software that provides ETL functionalities. Business Intelligence softwares are declined in five editions : the Community Edition, the Reporting Edition, the AWS Edition, the Professional Edition and the Enterprise Edition proposing an increasing range of functionalities.

The Big Data Business Intelligence tool proposing to combine multiple data sources in a data warehouse and perform real time analysis, it also allows using the ETL function to convert the data from one technology to another. Compatible database technologies are varied, encompassing MongoDB, Amazon Redshift, Hadoop, Google Big Query, Cassandra and Cloudera [Inc., 2015].

2.7.5 Informatica PowerCenter

Informatica Corporation is a software development company providing multiple products in various fields including Big Data, Cloud Integration, Data Integration, Data Quality, Data Security, Informatica Platform, Integration Platform-as-a-Service and Master Data Management.

Among Data Integration products is PowerCenter, a tool used for multiple purposes including ETL processes. The tool is declined in three editions : the Standard Edition, the Advanced Edition and the Premium Edition, all being paid solutions. Key features of the product encompass business and IT collaboration, reusability, automation, ease of use, metadata management, scalability, performance and zero downtime [Corporation, 2015].

Chapter 3

State of the art

3.1 Database migration

Firstly, any process of database engineering is located at three different levels [Hainaut, 2012]:

1. **Conceptual**: This is the most abstraction level. At the conceptual level, there is no technological dependency.
2. **Logical**: Just under the conceptual level, the logical level implements data structures and constraints according to the technology concerned.
3. **Physical**: The lowest level is purely physical and mainly focuses on what will support the database and its structure like storages, indexes and others.

Secondly, a migration between two different databases implies three different objects : the database schema, the data and the application programs using the database [Hainaut, 2006]. These objects are respectively involved by three different processes pictured in the figure 3.1.

1. **Schema conversion**: The bridge between two different data models has to ensure that both database schemas have at least the same semantics.
2. **Data conversion**: Data from the source database have to be rethought according to the target database, its structure and its data model.
3. **Program conversion**: Application programs have to be tailored to still communicate to the new database with the same behavior than before.

Each process has to be executed to consider the database platform migration as finished.

3.1.1 Transformational approach

The transformational approach is mainly used to operate a migration in the database area. This approach is a part of Model-Driven engineering and makes sense for us because

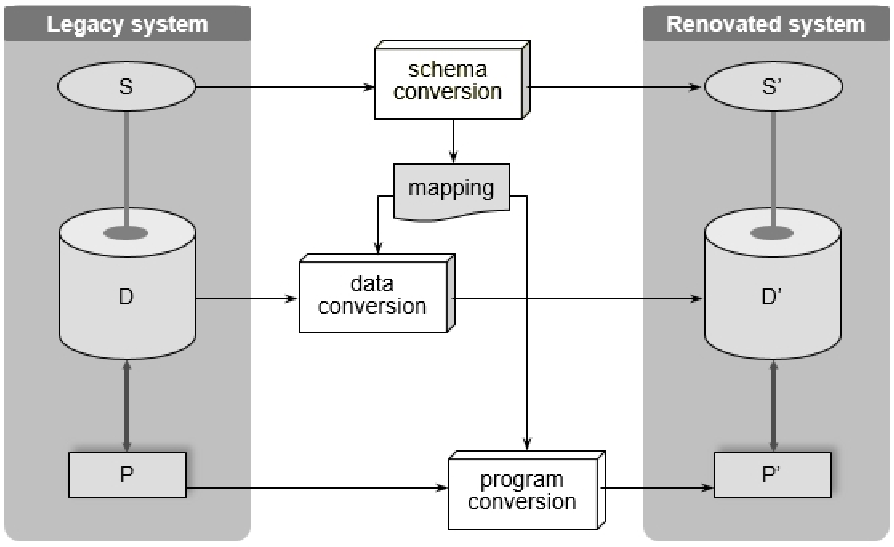


Figure 3.1: General migration schema [Clève, 2009]

this allows producing databases from abstract models in an efficient and an automatic way.

The transformational approach mainly focuses on the database schema. Its main goal aims to ensure that each transformation is correctness-preserving. In other terms, this approach guarantees keeping the same information content all along the process [Hainaut, 2006].

3.1.1.1 Schema conversion

There are two distinct processes to migrate the structure of a database. The first one is the database reverse engineering and the second is the database forward engineering.

3.1.1.1.a Database reverse engineering

The goal of reverse engineering aims to understand, to know how and why a system has been built. The reverse engineering process looks for implicit structures which led to this building and has to go beyond what looks obvious. There are a lot of methods and studies to operate this process.

These methods depend on the context in which the process is executed. Since our method of migration focuses on databases, its goal consists of documenting these by aggregating its schemas. These schemas are abstracts and made from source DDL code, programs and other information sources. Those shall correspond to the domain supported by the database concerned. And the database reverse engineering should under no circumstances alter data [Hainaut, 2012].

The database reverse engineering is based on three different processes as pictured in the figure 3.2.

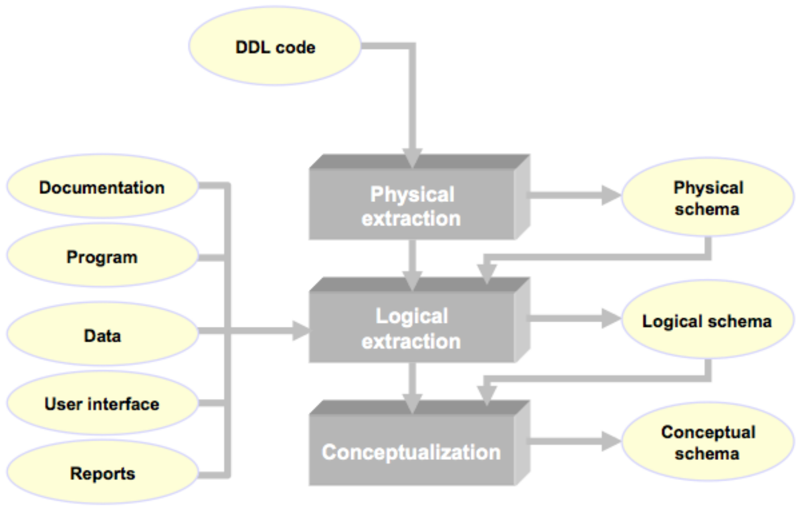


Figure 3.2: Database reverse engineering process [Clève, 2009]

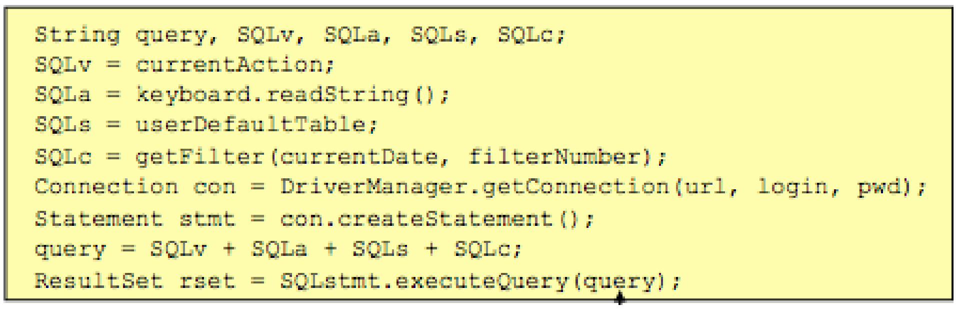
Physical extraction

The physical extraction is purely technical, capable of being automated and extract from DDL source code, the physical schema of the database. Only explicit structures like tables and columns are extracted. This first process is common and essential to build the base [Hainaut, 2012].

Logical extraction

From the whole of information sources (cf. Figure 3.2, p. 61), the physical schema is enriched by implicit structures. Even though it is already possible from the physical schema as it is to deduce some interesting properties like a primary key when there is an index on a non-identifying column for example, it is not enough to find all implicit structures essential to understand the domain of the database concerned [Hainaut, 2012].

As a matter of fact, another information source very interesting to derive properties is the program using the database. Several kinds of analysis are possible and are divided into two families: static and dynamic analysis [Hainaut, 2012].
The static analysis focuses on SQL statements which are not dynamically generated. But a lot of SQL statements are dynamically generated as pictured for exemple in the figure 3.3 [Clève, 2009]. In such case, dynamic analysis is more relevant. But no matter what the kind of analysis is chosen, both are based on finding dependencies between queries (or into only one) to derive foreign keys for example.



```
String query, SQLv, SQLa, SQLs, SQLc;
SQLv = currentAction;
SQLa = keyboard.readString();
SQLs = userDefaultTable;
SQLc = getFilter(currentDate, filterNumber);
Connection con = DriverManager.getConnection(url, login, pwd);
Statement stmt = con.createStatement();
query = SQLv + SQLa + SQLs + SQLc;
ResultSet rset = SQLstmt.executeQuery(query);
```

Figure 3.3: Dynamic SQL query [Clève, 2009]

The dynamic analysis is implemented in two phases. On the one hand, there is the tracing aiming to retrieve a SQL trace executed at runtime during the program execution. On the other hand, there is the analysis of this trace to derive interesting properties from the database and/or from the program.
Ideally, the capture has to be focused on the four phases of the dynamic generation of the query (i.e. statement preparation, value injection, query execution and result extraction). There are several methods and techniques to do it as summarized in the figure 3.4.

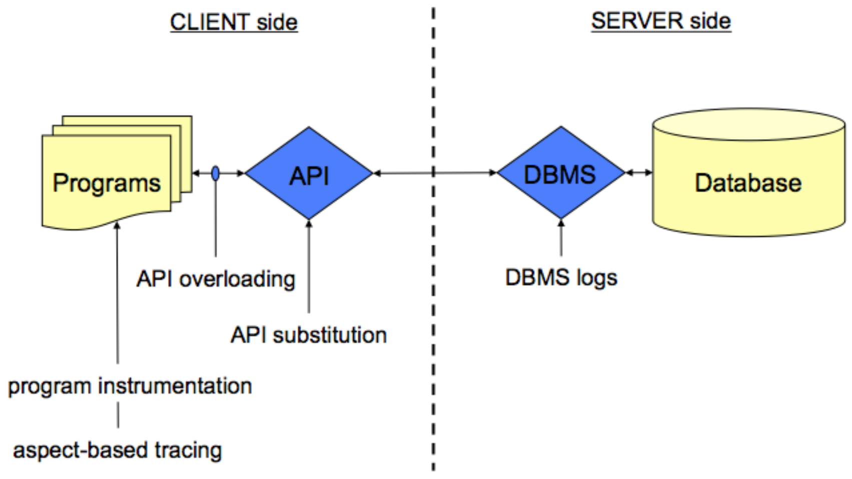


Figure 3.4: SQL tracing techniques [Clève, 2009]

In more details, here are presented each method of SQL tracing, their description, advantages and drawbacks [Clève, 2009].

DBMS logs

The best known database management systems provide logging. Thanks to this feature, it is possible to keep all queries executed against the database. This functionality is easy and free but does not cover all four phases for the dynamic generation of queries but just the query execution. Unfortunately, the recovered trace can also be polluted by system queries or others coming from programs others than those observed. But it is still convenient because there is no source code modification.

API substitution

As its name suggests, API (Application Program Interface) substitution consists of replacing the legacy API database access (like JDBC for example) by another one incorporating tracing statements. Thus, the four phases are executed and traced in parallel. There is no source code modification and the analysis is more accurate than by using DBMS logs. That being said, API substitution is more difficult to maintain, compile and modify.

API overloading

API overloading process wraps the legacy API database access into an intermediate one where tracing statements are implemented. This solution also provides accurate analysis like the previous one by covering all four phases about the dynamic generation of queries. But, there is source code modification and still no source code location.

Program instrumentation

This way of catching SQL trace involves a lot of modifications into the source code but allows getting source code points. In point of fact, program instrumentation consists of adding tracing statements directly inside the program. All four phases are covered but the source code points must be translated into initial source code location.

Aspect-based Tracing

Finally, Aspect-based Tracing is a paradigm which specifies separately the tracing functionality by means of aspects. This means that we state the functions corresponding to four phases (statement preparation, value injection, query execution and result extraction) as pointcuts. Then the program is running and when a function stated as pointcut is called, this program stops its linear execution to carry out what we have coded before according to each implementation. The analysis is complete, the program is more modular than program instrumentation for example but did not meet a great success. There are only a few languages allowing this paradigm (Java, C, C++, Cobol).

Then, the SQL trace analysis depends on the richness of the SQL trace retrieved before but also the final goals defined. This phase can be achieved in many ways. Actually, the SQL trace analysis can be carried out On-The-Fly or Off-Line, intra-execution or inter-execution, intra-query or inter-queries and based on trace only or more [Clève, 2009].
If the final goal aims to retrieve implicit foreign keys from a program, the SQL trace analysis can be executed Off-Line into an only trace execution and based on just the trace. It will analyse between several queries looking for output-input and input-input dependencies. If there are the same values into both queries, this could mean a relationship between tables referred on both sides. But it is also possible to infer foreign keys into an only query by looking at jointures [Clève, 2009].

To conclude about the logical extraction process of database reverse engineering, once the schema has been aggregated by all implicit structures found, it is important to remove other structures such as indexes, storages and other physical buildings to obtain the right logical database schema [Hainaut, 2012].

Conceptualization

Conceptualization takes as an input the logical database schema from the previous process (as described above) and creates the corresponding normalized conceptual database schema as an output. There are several conceptual schemas possible from one logical schema because one construction in this schema can have multiple interpretations at the conceptual level. Also, this process is not obvious because designers could have optimized the logical schema and have maybe used non-standard patterns to build it. But also databases have history, they can know different periods with different ways of doing. Though, this process can be subdivided into two subprocesses called "untranslation" and "unoptimization" [Hainaut, 2012] to manage all these difficulties. These two processes seek to produce a conceptual schema devoid of all technical constraints due to DBMS source.

First, the untranslation of a logical schema is the exact opposite of the logical design discussed later. This aims at choosing a sample of possible transformations from the conceptual to the logical level and to go in the opposite way, from logical to conceptual. For example, this process translates from foreign keys towards relationships. Also, this process can translate series of heterogeneous attributes with similar names to compound attributes and series of homogeneous attributes with similar names to multivalued attributes [Hainaut, 2012].

Then, the unoptimization of a logical schema aims to erase all logical constructions built before for the only reason to make the database more efficient. There is no standard for this process but there are three kinds of methods [Hainaut, 2012]:

- **Schemas restructuring**: Schemas restructuring joins tables which have been divided before into multiple ones.
- **Structural redundancy**: Structural redundancy deletes structures which are multiple and do not provide any value.
- **Internal redundancy**: Internal redundancy normalizes objects (entity type or/and relationships).

3.1.1.1.b Database forward engineering

The second process is the database forward engineering which produces the code from the conceptual schema as described in the figure 3.5. This is the opposite way of database reverse engineering.

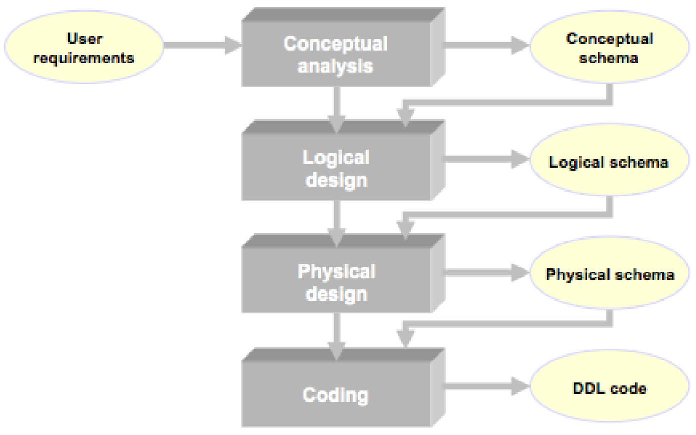


Figure 3.5: Database forward engineering [Clève, 2009]

Conceptual analysis

Understanding the domain supported by the database concerned is the main goal of the conceptual analysis. From users requirements, this analysis produces a complete, correct and normalized conceptual schema. But also, this analysis can produce a dynamic schema which highlights the behavior that the database will have to respect later [Hainaut, 2012]. A priori, in the context of a migration, the database reverse engineering process has already built the conceptual schema and this phase has no great interest anymore. Except if users and domain requirements require a shift at the conceptual level too. For example, if the migration is also the opportunity to comply the database structure to the standards.

Logical design

The logical design aims to produce a logical schema from the conceptual schema built before. This schema has to fit with the data model of the target RDBMS. In other words, this schema is located at a lower abstraction level since it is more technology-dependent than the conceptual schema. Usually, this process is automatable because it can be supported by a well-defined transformation plan. This transformation plan gathers all necessary rules to migrate from the conceptual schema devoid of all technological constraints and the logical schema which fits with a precise data model. Also, one of the most important properties that this plan has to respect is the semantics preserving all along the process. This means that it is important that every construction within the conceptual schema has its equivalent into the logical schema [Hainaut, 2012].

Physical design

The physical design process has the goal to enrich the input logical schema by technical mechanisms like indexes and storages. Though, each table is attached to a precise storage and some columns are indexed. But also, it is possible to modify the structure of logical schema according to some technical requirements to speed queries up. In short, the physical design aims to make applications using the database as efficient as possible.

Coding

Once the physical schema is built, the last phase aims to translate each construct of this into an understandable code for the target RBDMS. That being said, each construct is not always directly translatable. Though, this phase is semi-automatable. On the one hand, understandable constructs are translated but on the other hand, remaining ones need additional code. But in the end, every construct of the physical schema have to find their correspondent into the code to respect the original semantic structure.

Hence, all along these two processes, the semantic is the element to keep. And to carry out all these transformations between different technologies and abstraction levels, there is one solution to reduce the complexity, a pivot model between them.

3.1.1.2 Pivot model

Incorporate an intermediate between the different technologies involved by these transformations reduces the amount of mappings. Indeed, if we have to pass from M formalisms to N formalisms, instead of NxM mappings, there are just N+M mappings. Also, if we use a pivot model which has been approved, it serves as proof of transformations [Hainaut, 2006].

Furthermore, "Generic Entity-Relationship" (GER) formalism will be used in this thesis. It supports concepts as schemas, entity type, domains, attributes, relationship types, keys and other constraints [Hainaut, 2006].

This model is well defined for the relational databases because this fits very well with entity-relationship DBMS models as described in the table 3.1.

Relational constructs	**GER constructs**	**Assembly rules**
Database schema	Schema	
Table	Entity type	An entity type includes at least one attribute
Domain	Simple domain	
Nullable column	Single-valued and atomic attribute with cardinality [0-1]	
Not null column	Single-valued and atomic attribute with cardinality [1-1]	
Primary key	Primary identifier	A primary identifier comprises attributes with cardinality [1-1]
Unique constraint	Secondary identifier	
Foreign key	Reference group	The composition of the reference group must be the same as that of the target identifier
SQL names	GER names	GER names must follow the SQL syntax

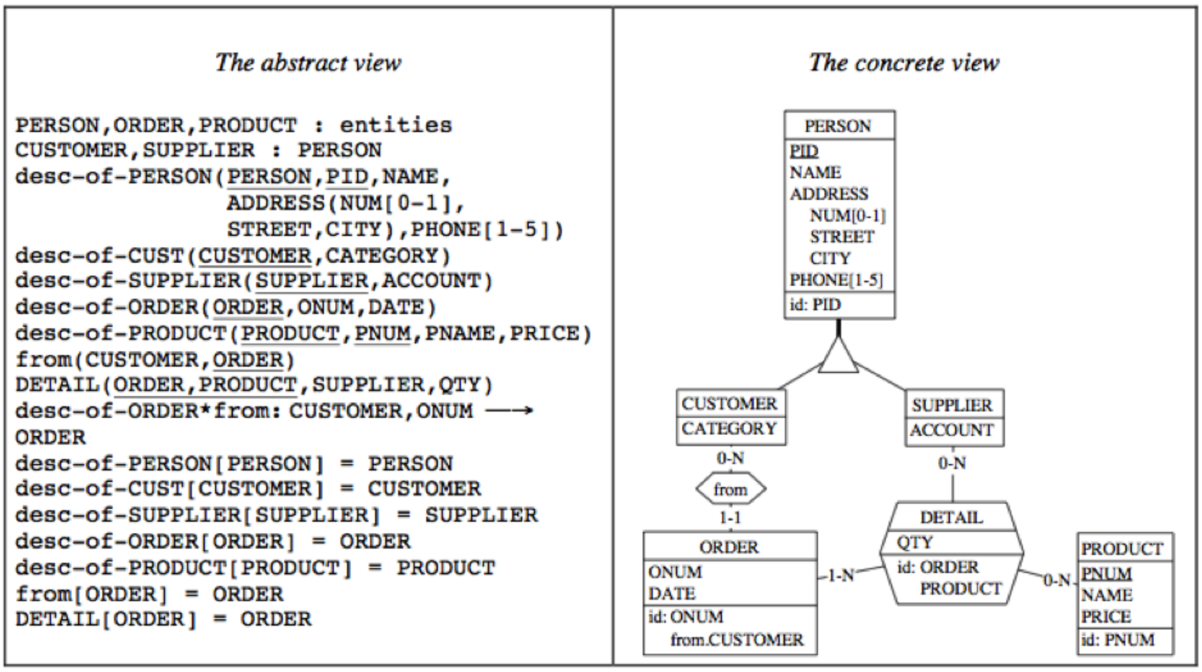
Table 3.1: Mappings of some concepts in the relational databases [Hainaut, 2006]

As a matter of fact, GER model has two ways to be defined, the first is the concrete view with ER diagrammatic connections and the second is the abstract view which is *"an extended N1NF model in which all GER constructs are given a uniform relational interpretation"* [Hainaut, 1996]. These two views can be compared and allow to prove properties about the database.

Also, GER formalism is made from data structures which can be specified at three different abstraction levels: the conceptual, logical and physical levels [Hainaut, 2006].

3.1.1.2.a Conceptual level

The first one is the conceptual level where *"a GER schema specifies entity types, relationship types and attributes. [...] The model also includes such advanced constructs as is-a relationships, multivalued and compound attributes, roles with cardinality constraints, entity and relationship identifiers, attribute domains,..."* [Hainaut, 1996] as pictured in the figure 3.6.


The abstract view

```
PERSON,ORDER,PRODUCT : entities
CUSTOMER,SUPPLIER : PERSON
desc-of-PERSON(PERSON,PID,NAME,
               ADDRESS(NUM[0-1],
               STREET,CITY),PHONE[1-5])
desc-of-CUST(CUSTOMER,CATEGORY)
desc-of-SUPPLIER(SUPPLIER,ACCOUNT)
desc-of-ORDER(ORDER,ONUM,DATE)
desc-of-PRODUCT(PRODUCT,PNUM,PNAME,PRICE)
from(CUSTOMER,ORDER)
DETAIL(ORDER,PRODUCT,SUPPLIER,QTY)
desc-of-ORDER*from: CUSTOMER,ONUM ——→
ORDER
desc-of-PERSON[PERSON] = PERSON
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPPLIER[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PRODUCT[PRODUCT] = PRODUCT
from[ORDER] = ORDER
DETAIL[ORDER] = ORDER
```

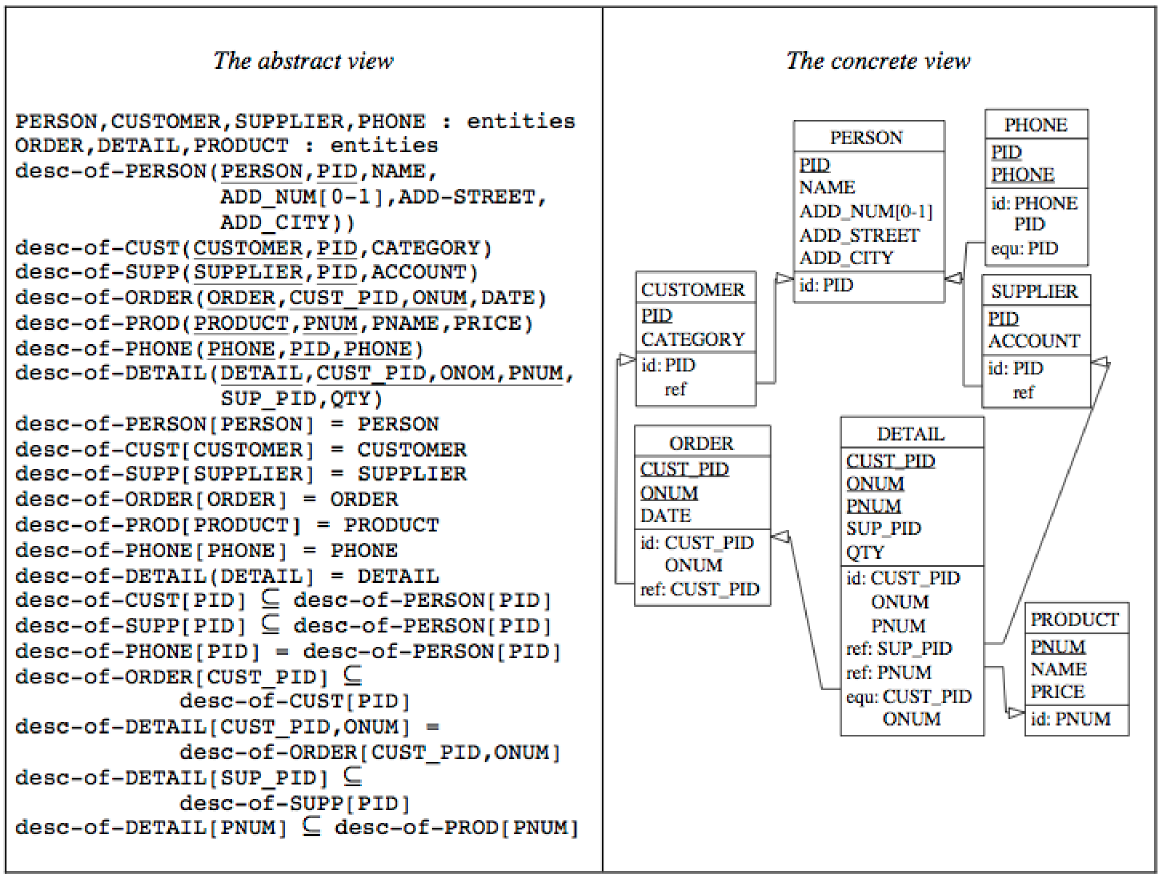
Figure 3.6: GER constructs at conceptual level [Hainaut, 1996]

Though, the abstract view formally describes what is graphically pictured in the concrete view. Here, an entity type is pictured by an entity subset or another entity domain. An attribute is described by a relation "desc-of-X" and a relationship type is represented by a relation [Hainaut, 1996].

3.1.1.2.b Logical level

Then, there is the logical level where GER aims to define logical structures. It means that GER adapts its specifications according to the chosen DBMS model. *"A logical entity represents a record, an object class, a segment type or a table, a logical attribute represents a field or a column, a relationship type represents a set type or a parent-child relationship"* [Hainaut, 1996], etc.

The figure 3.7 represents the fact that an inclusion constraint is used to represent a foreign key in the abstract view. Whereas in the concrete view, a foreign key is represented *"by the ref keyword, and by an arc toward the referenced identifier"* [Hainaut, 1996].



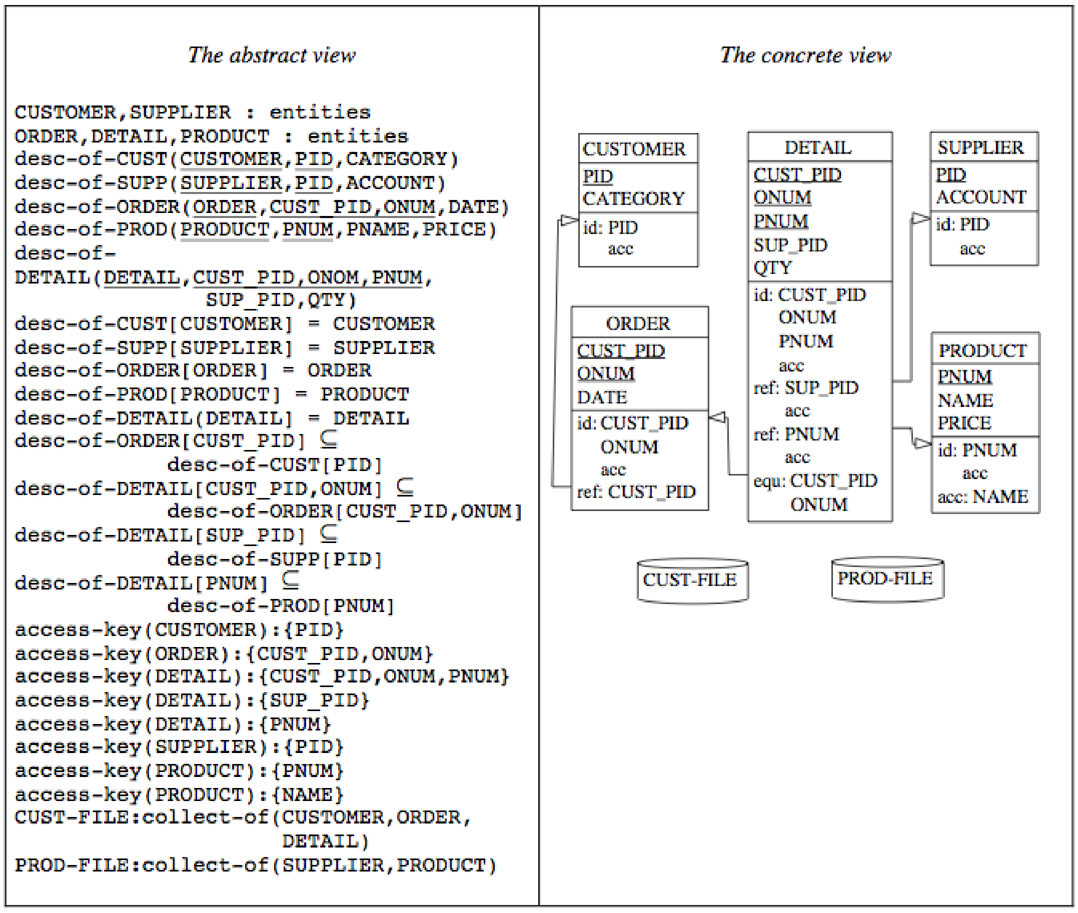
```
PERSON,CUSTOMER,SUPPLIER,PHONE : entities
ORDER,DETAIL,PRODUCT : entities
desc-of-PERSON(PERSON,PID,NAME,
               ADD_NUM[0-1],ADD-STREET,
               ADD_CITY))
desc-of-CUST(CUSTOMER,PID,CATEGORY)
desc-of-SUPP(SUPPLIER,PID,ACCOUNT)
desc-of-ORDER(ORDER,CUST_PID,ONUM,DATE)
desc-of-PROD(PRODUCT,PNUM,PNAME,PRICE)
desc-of-PHONE(PHONE,PID,PHONE)
desc-of-DETAIL(DETAIL,CUST_PID,ONOM,PNUM,
               SUP_PID,QTY)
desc-of-PERSON[PERSON] = PERSON
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPP[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PROD[PRODUCT] = PRODUCT
desc-of-PHONE[PHONE] = PHONE
desc-of-DETAIL(DETAIL] = DETAIL
desc-of-CUST[PID] ⊆ desc-of-PERSON[PID]
desc-of-SUPP[PID] ⊆ desc-of-PERSON[PID]
desc-of-PHONE[PID] = desc-of-PERSON[PID]
desc-of-ORDER[CUST_PID] ⊆
            desc-of-CUST[PID]
desc-of-DETAIL[CUST_PID,ONUM] =
            desc-of-ORDER[CUST_PID,ONUM]
desc-of-DETAIL[SUP_PID] ⊆
            desc-of-SUPP[PID]
desc-of-DETAIL[PNUM] ⊆ desc-of-PROD[PNUM]
```

Figure 3.7: GER constructs at logical level [Hainaut, 1996]

3.1.1.2.c Physical level

The third and the lowest level is the physical level. A physical GER schema includes concepts like entity collection which represents record repositories as files but also access keys specify access mechanisms like indexes.

In the figure 3.8, on the one hand, the collect-of clauses in the abstract view and cylinder symbols in the concrete view represent collections. On the other hand, the access-key keyword in the abstract view and the acc keyword in the concrete view represent access keys [Hainaut, 1996].


The abstract view

```
CUSTOMER,SUPPLIER : entities
ORDER,DETAIL,PRODUCT : entities
desc-of-CUST(CUSTOMER,PID,CATEGORY)
desc-of-SUPP(SUPPLIER,PID,ACCOUNT)
desc-of-ORDER(ORDER,CUST_PID,ONUM,DATE)
desc-of-PROD(PRODUCT,PNUM,PNAME,PRICE)
desc-of-
DETAIL(DETAIL,CUST_PID,ONOM,PNUM,
                SUP_PID,QTY)
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPP[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PROD[PRODUCT] = PRODUCT
desc-of-DETAIL(DETAIL] = DETAIL
desc-of-ORDER[CUST_PID] ⊆
            desc-of-CUST[PID]
desc-of-DETAIL[CUST_PID,ONUM] ⊆
            desc-of-ORDER[CUST_PID,ONUM]
desc-of-DETAIL[SUP_PID] ⊆
            desc-of-SUPP[PID]
desc-of-DETAIL[PNUM] ⊆
            desc-of-PROD[PNUM]
access-key(CUSTOMER):{PID}
access-key(ORDER):{CUST_PID,ONUM}
access-key(DETAIL):{CUST_PID,ONUM,PNUM}
access-key(DETAIL):{SUP_PID}
access-key(DETAIL):{PNUM}
access-key(SUPPLIER):{PID}
access-key(PRODUCT):{PNUM}
access-key(PRODUCT):{NAME}
CUST-FILE:collect-of(CUSTOMER,ORDER,
                     DETAIL)
PROD-FILE:collect-of(SUPPLIER,PRODUCT)
```

Figure 3.8: GER constructs at physical level [Hainaut, 1996]

3.1.1.3 Schema transformation

3.1.1.3.a Definition

The schema transformation in this thesis is seen as an operator T replacing a construct C by another C' from a schema S to another S'. The schema transformation is not just about structural mapping but also instance mapping. Indeed, the structural mapping (T) is about the syntax whereas the instance mapping (t) is about the semantics. Though, a schema transformation is defined by its mappings: **Σ** = <T,t> [Hainaut, 1996].

$$\begin{tikzcd}
\mathbf{C} \arrow[r, "\mathbf{T}"] & \mathbf{C'=T(C)} \\
\mathbf{c} \arrow[u, "\text{instance-of}"] \arrow[r, "\mathbf{t}"] & \mathbf{c'=t(c)} \arrow[u, "\text{instance-of}"']
\end{tikzcd}$$

Figure 3.9: Schema transformation definition [Hainaut, 1996]

There are several ways to define the structural mapping of a schema transformation and one of them is the predicative approach. In this approach, T is defined as a couple of predicates <P,Q> where P determines the minimal precondition that C must respect and Q is the maximal postcondition which must be met for C'. In other terms, this couple defines what C and C' should respect before and after the transformation: P(C) ⇒ Q(T(C)) [Hainaut, 1996].

Therefore, the schema transformation can be rewritten like this: **Σ** = <P,Q,t> where P and Q are both second-order predicates. These predicates can be expressed with GER formalism and t can be expressed in pseudo-code as described in the table 3.2.

P	CUSTOMER,STOCK:entities PURCH(CUSTOMER,STOCK)

Q	CUSTOMER,STOCK,PURCH:entities
	CP(CUSTOMER,PURCH)
	SP(STOCK,PURCH)
	CP[PURCH]=PURCH
	SP[PURSH]=PURCH
	CP*SP:CUSTOMER,STOCK –>PURCH

t	for each p = (c,s) of the current instance of PURCH do generate arbitrary entity p ' of PURCH insert (p',c) in the current instance of CP insert (p',s) in the current instance of SP

Table 3.2: Predicative and procedural expression of the structural and instance mapping of the transformation figure 3.10

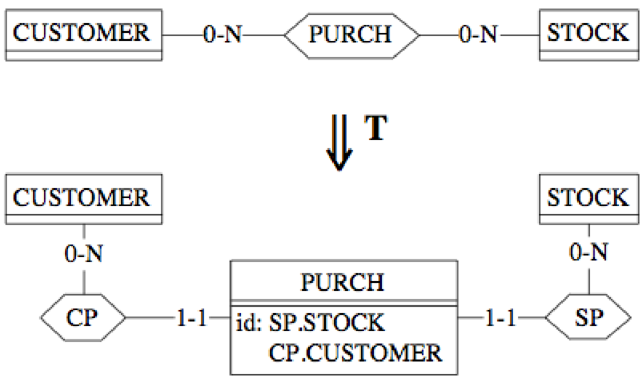


Figure 3.10: Transformation of a relationship type into an entity type [Hainaut, 1996]

That being said, this previous table describes only one transformation. Indeed, this transformation belongs to a class of them. This class is specified by a transformation scheme as we can see in the table 3.3.

P	E1,E2:entities R(E1,E2)

Q	E1,E2,R:entities
	R1(E1,R)
	R2(E2,R)
	R1[R]=R
	R2[R]=R
	R1*R2:E1,E2 –>R

t	for each p = (c,s) of the current instance of PURCH do generate arbitrary entity p ’ of PURCH insert (p’,c) in the current instance of CP insert (p’,s) in the current instance of SP

Table 3.3: The transformation scheme from which the transformation of this table 3.2 has been instantiated [Hainaut, 1996]

It is trivial to directly see the instantiation which has been operated.

3.1.1.3.b Semantical issues about schema transformations

Now about the semantics of these transformations, one of the most important properties aims to ensure that the target schema can substitute the source one with the same information content capacity. This property is named semantics preservation or reversibility. So, there is only one thing to prove, the fact that the transformation is reversible to prove the semantics preservation. Here is the definition:

"A transformation **Σ***1 = <T1,t1> = <P1,Q1,t1> is reversible, iff there exists a transformation* **Σ***2 = <T2,t2> = <P2,Q2,t2> such that, for any construct C, and any instance c of C: P1(C) ⇒ ([T2(T1(C))=C] and [t2(t1(c))=c]).* **Σ***2 is the inverse of* **Σ***1, but the converse is not true. For instance, an arbitrary instance c' of T(C) may not satisfy the property c'=t1(t2(c'))"* [Hainaut, 2006]

For example, transformation schemes described in these tables 3.4 and 3.5 are reversible transformations.

P	R(U) I J K = U ; I J K I

Q	R1(I,J)
	R2(I,K)

t	for each r = (r1,r2) of the current instance of R do generate arbitrary entity r' of R insert (r',e1) in the current instance of R1 insert (r',e2) in the current instance of R2

Table 3.4: Project-join decomposition [Hainaut, 1996]

P	R($\underline{I}$,$\underline{K}$,M); I,K not empty S(K,L); K not empty S[K] ⊆ R[K]
Q	R($\underline{I}$,$\underline{K}$,M); I,K not empty T(I,L); I not empty T[I] ⊆ R[I]
t1	let r be the current instance of R, let s be the current instance of S, let t b an instance of T, t = (r*s)[I,L]
t2	let r be the current instance of R, let t be the current instance of T, let s b an instance of S, s = (r*t)[K,L]

Table 3.5: The composition transformation [Hainaut, 1996]

Indeed, these transformations are reversible because *"the instance of R can always be recovered by the natural join f the corresponding instances of R1 and R2"* [Hainaut, 1996]. They also allow to prove other ones because they represent a scientific truth which cannot be disproved.

3.1.2 Co-Transformational approach

3.1.2.1 Overview

Previous steps described how to design a target structure capable of holding the information from the source database. So far, this new system is empty and cannot communicate to the application program(s) since the technology used has changed. A co-transformational approach consists of impacting the database schema modifications to both the data and the application program(s). Thus, this approach extends the transformational approach described in the previous section. Thereby, to perform a full database migration including not only the structure but also the data itself and the application program(s), we will follow in this thesis a transformational approach, which is also a co-transformational approach and vice versa [Cleve et Hainaut, 2006].

The goals of the co-transformational approach are highlighted in yellow in the figure 3.11.

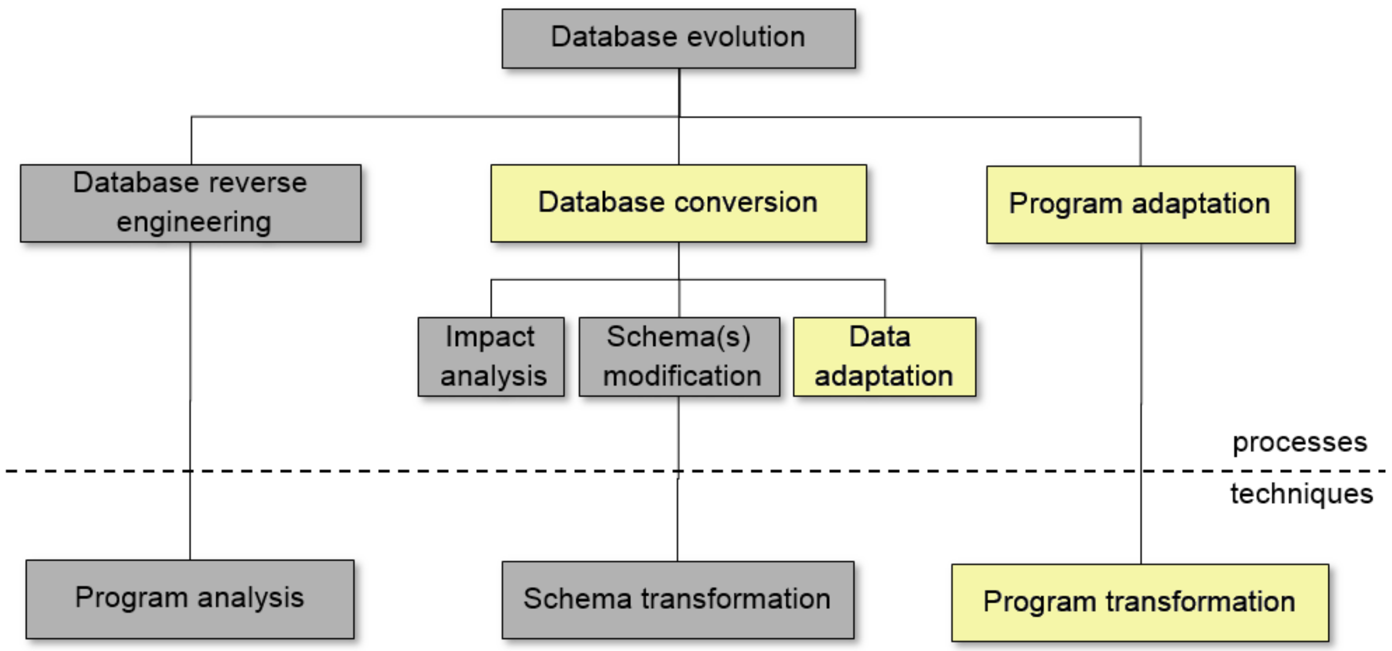


Figure 3.11: Database evolution processes and supporting techniques [Cleve, 2015]

Multiple processes exist to suit those needs and will be described in this section. They are separated in two categories:

1. **Data**: Processes related to data conversion that transform the data structure in respect to the target database's specificities and data model
2. **Application programs**: Processes related to program conversion that modify

(parts of) programs so they can maintain the same behaviour while communicating with the new database

3.1.2.2 Data conversion

Data conversion consists of migrating the data existing within the source database into the target database, while respecting its structure and data model. This procedure allows avoiding starting over with a brand new and empty database, and keep maintaining and using existing data. There are two paradigms when it comes to migrate data [Klaus, 2009]:

1. **Target-pull**: Only the data necessary to the target system is migrated, which is simpler and cheaper to perform
2. **Source-push**: The whole data from the old system is migrated, which is more complex and expensive to perform

Since only a part of the source attributes generally need to figure in the target database, the target-pull paradigm is the most often chosen option, even though the source-push would be ideal. Data conversion has a generic architecture composed of three main phases depicted in figure 3.12 [Klaus, 2009].

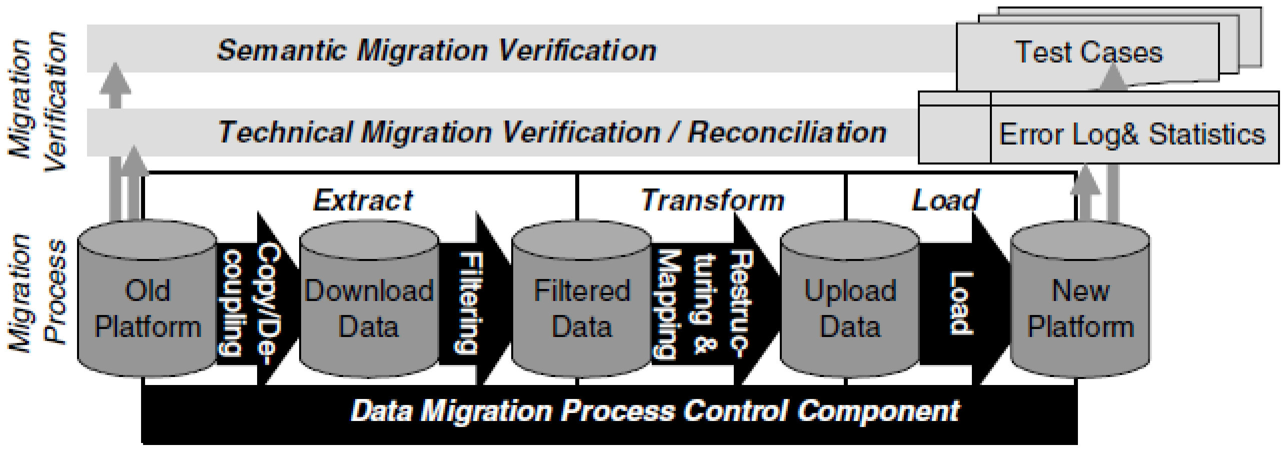


Figure 3.12: Generic Migration Architecture [Klaus, 2009]

1. **Extract**: At first, the whole data is copied in order to be filtered to only keep the data to migrate and reject the rest
2. **Transform**: Then, the data to be migrated is restructured and mapped to fit in the target schema
3. **Load**: At last, the transformed data is loaded within the target database, thus completing the data migration

During those steps, other actions like calculating statistics can be performed. Once the data is migrated, there is a verification phase insuring the correctness of the data loaded within the target database [Klaus, 2009].

3.1.2.3 ETL tools

ETL tools allow the user to define data-flows using a visual programming language. Those data-flows represent the mapping between one or more data sources and their destination in the target database structure [Klaus, 2009]. Originally, ETL tools were conceived as a way to automate Data Warehousing processes. Thus, ETL were - and are - mostly used in the context of business intelligence and more recently big data. Typically, an ETL tool is used for each data type to be extracted, transformed and loaded in a data warehouse [Simitsis et Vassiliadis, 2003]. In the case of database migration, data will be loaded in a target database rather than in a data warehouse.

3.1.2.3.a Extract step

The extract step aims at collecting the data that will later be transformed and loaded in the target database. To do so, this step has two objectives and is consequently divided in two sub-steps [Klaus, 2009].

Download sub-step The first objective is to decouple the data from the project itself to keep only raw data without any interference. Most to all data is downloaded on an external copy on which transformations will be performed without risking perturbations from the project's daily operations.

Filtering sub-step This sub-step consists of choosing objects that will make it to the target system. As precised earlier, most data migration opt for a *target-pull* approach rather than a *source-push*, thus rejecting part of the data that is not relevant or necessary to the operation of the target system. Haller identifies three main filtering patterns that are used during data conversion:

1. **Attribute value based filtering**: Filtering a row based on a given criteria, typically the value of an attribute, independently of other rows and tables
2. **Selection table based filtering**: Filtering a row based on the value of one or more attribute(s) in another table, typically using a foreign key or a join
3. **Aggregation based filtering**: Filtering a row based on the information in multiple rows and multiple tables, typically using aggregation functions

The result of this step is a set of data independent from the project and ready to be transformed and restructured. Figure 3.13 gives an example of the complete extract step for a banking data system [Klaus, 2009].

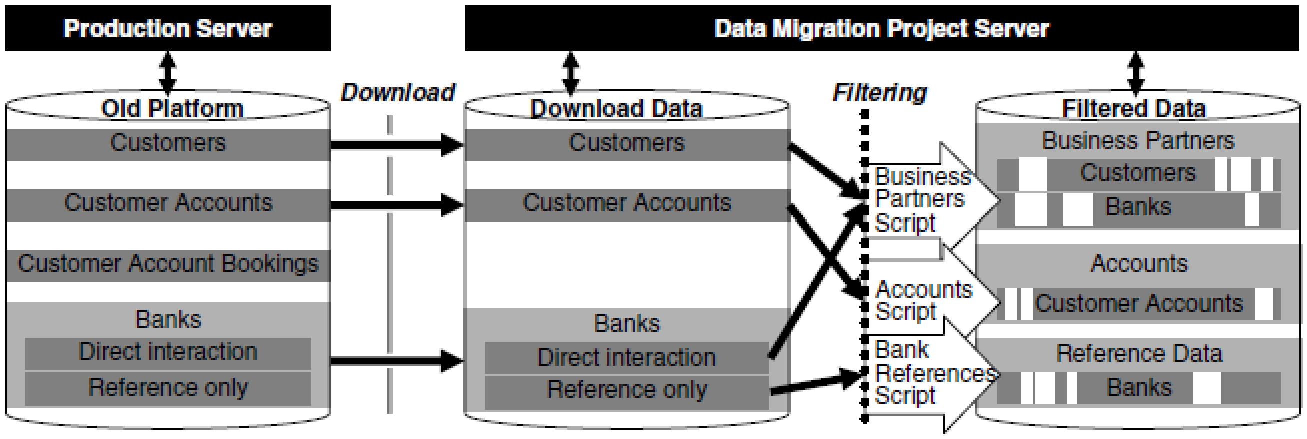


Figure 3.13: Extract step [Klaus, 2009]

3.1.2.3.b Transform step

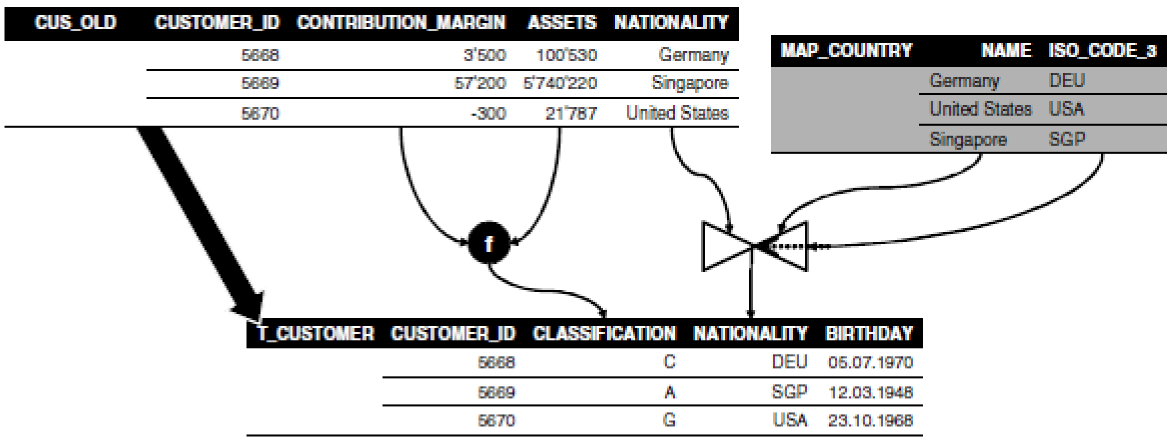
The transform step aims at converting and restructuring the data in order to fit in the target database's structure. Depending on the objects being migrated, two *pattern groups* can be employed to transform the data.

Mapping The first pattern group is used when objects have similar data-models in both databases and focuses on mapping source attributes to their counterpart in the target database. Haller distinguishes two patterns within this group [Klaus, 2009]:

1. **Mapping table**: The mapping is represented in an intermediate table that contains the source database attribute's value(s) and their equivalent in the target database

2. **Mapping function**: The mapping is too complex to be simply displayed in a translation table, and thus is represented by a function that takes one or more source attribute(s) as an input and produces one or more target attribute(s)

Figure 3.14 depicts the *mapping* pattern group and provides an example of both patterns' usage. We can observe the translation table MAP_COUNTRY mapping countries' names to their ISO code counterpart used in the target database. Furthermore, CONTRIBUTION_MARGIN and ASSETS attributes are used as an input of a function F that produces a CLASSIFICATION for each customer in the target database.



CUS_OLD	CUSTOMER_ID	CONTRIBUTION_MARGIN	ASSETS	NATIONALITY
	5668	3'500	100'530	Germany
	5669	57'200	5'740'220	Singapore
	5670	-300	21'787	United States

MAP_COUNTRY	NAME	ISO_CODE_3
	Germany	DEU
	United States	USA
	Singapore	SGP

T_CUSTOMER	CUSTOMER_ID	CLASSIFICATION	NATIONALITY	BIRTHDAY
	5668	C	DEU	05.07.1970
	5669	A	SGP	12.03.1948
	5670	G	USA	23.10.1968

Figure 3.14: Sample Tables *Mapping* Pattern Group [Klaus, 2009]

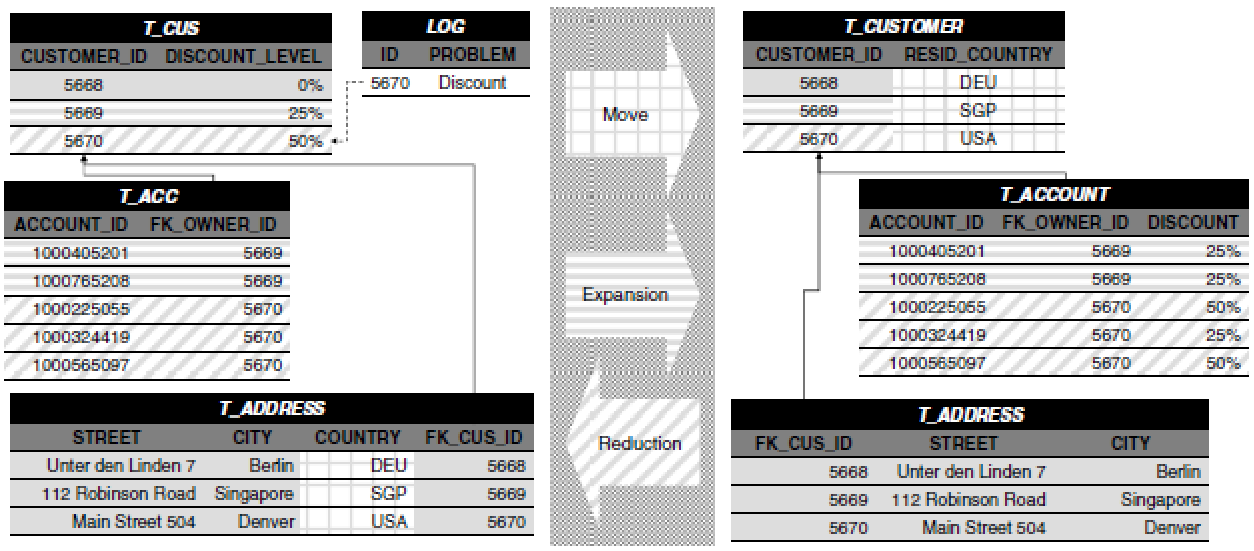
Restructuring The second pattern group is used when objects have different data-models in both databases, and the source structure has to be consequently reworked to fit in the target database. This group contains three restructuring patterns [Klaus, 2009]:

1. **Simple Attribute Move**: An attribute remains the same during the transformation but is transferred from one table to another one

2. **Expansion**: The target database relies on enhanced semantics and thus one or more attributes are added to the target structure

3. **Reduction**: On the opposite, the target database relies on lessened semantics and thus one or more attributes disappear, which may be mandatory logged in another table depending on the context

Figure 3.15 depicts the *restructuring* pattern group and provides an example of the three patterns' usage. We can observe the attribute COUNTRY incurring a name change and moving from the source table T_ADDRESS to the target table T_CUSTOMER. The source table T_ACC, becoming the target table T_ACCOUNT, gains an attribute DISCOUNT which has the consequence to add a level of granularity to the data model. On the other way around, considering T_ACC as the target table, a granularity level is lost from the data model and so a LOG table is added to save the information.



T_CUS

CUSTOMER_ID	DISCOUNT_LEVEL
5668	0%
5669	25%
5670	50%

LOG

ID	PROBLEM
5670	Discount

T_ACC

ACCOUNT_ID	FK_OWNER_ID
1000405201	5669
1000765208	5669
1000225055	5670
1000324419	5670
1000565097	5670

T_ADDRESS

STREET	CITY	COUNTRY	FK_CUS_ID
Unter den Linden 7	Berlin	DEU	5668
112 Robinson Road	Singapore	SGP	5669
Main Street 504	Denver	USA	5670

T_CUSTOMER

CUSTOMER_ID	RESID_COUNTRY
5668	DEU
5669	SGP
5670	USA

T_ACCOUNT

ACCOUNT_ID	FK_OWNER_ID	DISCOUNT
1000405201	5669	25%
1000765208	5669	25%
1000225055	5670	50%
1000324419	5670	25%
1000565097	5670	50%

T_ADDRESS

FK_CUS_ID	STREET	CITY
5668	Unter den Linden 7	Berlin
5669	112 Robinson Road	Singapore
5670	Main Street 504	Denver

Figure 3.15: Restructuring Pattern Group Examples [Klaus, 2009]

3.1.2.3.c Load step

Once the data has been correctly extracted and transformed, the load step aims at concluding data conversion by inserting that data in the target database. To do so, three different patterns exist [Klaus, 2009].

Direct approach The data is directly inserted in the target tables without the use of an API, which is at first sight the cheapest approach but also the most risky considering that migration errors, apart from severe ones, will not be detected, resulting in the need of a highly trained migration team with deep knowledge and understanding of internal tables and manually implemented checks.

Simple API approach The data is transferred to an API (within API tables) that performs loading procedures which also check failures or non-compliances with the data-model, thus decreasing the need of a highly trained migration team that may still want to manually implement checks in addition to existing checks.

Workflow-based API approach The data is transferred to an API (within API tables) that performs procedures used for inserting data. Contrarily to the *simple API approach*, the *workflow-based API approach* invokes the workflow separately for each object to load them one by one in the target database, leading in the use of an external process that does not need any additional cost to be implemented and checked.

Figure 3.16 details differences between the three approaches.

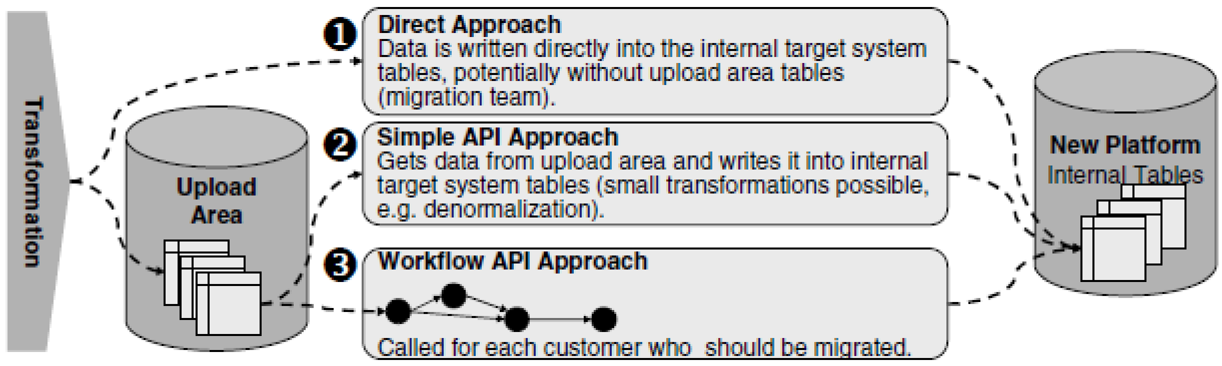

Transformation
Upload Area
1
Direct Approach
Data is written directly into the internal target system tables, potentially without upload area tables (migration team).
2
Simple API Approach
Gets data from upload area and writes it into internal target system tables (small transformations possible, e.g. denormalization).
3
Workflow API Approach
Called for each customer who should be migrated.
New Platform
Internal Tables

Figure 3.16: Data Loading Approaches [Klaus, 2009]

3.1.2.3.d Reconciliation

Despite the use of ETL tools ensuring data consistency and being careful when choosing which patterns to perform during the conversion, the process of data migration remains challenging and potentially risky in terms of data loss. Data being a precious resource, particularly for data-intensive systems, it is important to ensure the integrity of migrated data by verificating data and testing validation [Paygude et Devale, 2011].

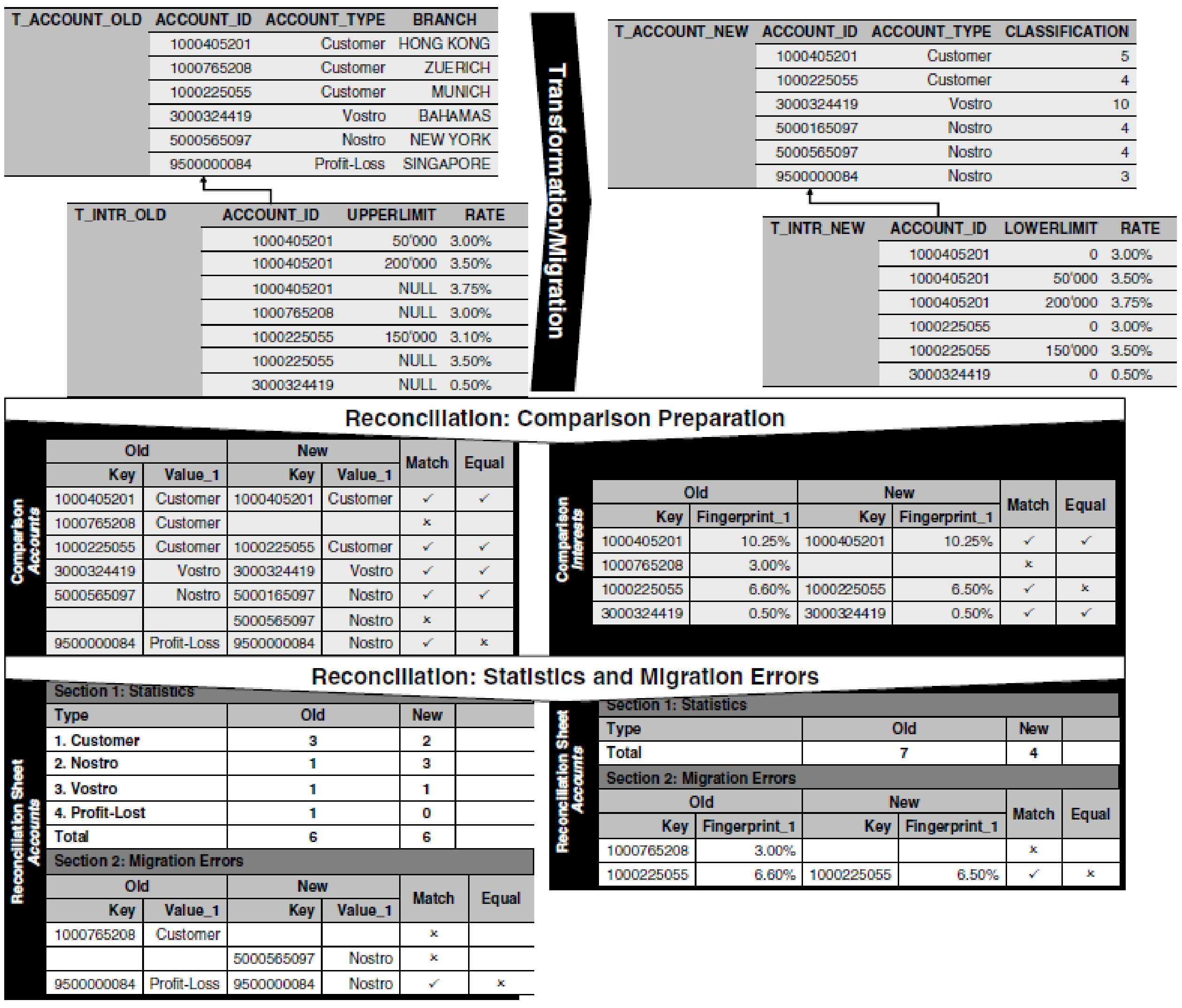
Haller designates this process as the *Reconciliation Process* and focuses on checking selected attributes of all objects in an automated way. Using this method, a *reconciliation sheet* is constituted and is typically composed of two parts : **statistics** and **migration errors**. Three patterns are distinguished to derive this sheet [Klaus, 2009]:

Top-down pattern In this simple approach, the number of objects in both databases is counted and statistics are established based on this census.

Bottom-up equivalence pattern This approach, the most useful when no restructuring is involved, goes further than pure statistics and allows identifying which objects got lost by selecting a unique key for each row of a table in both databases and checking if there is a match for all rows.

Bottom-up fingerprint pattern This approach, adapted from the previous one in case a restructuring is involved, allows to identify which objects got lost by selecting a fingerprint for one or more rows of a table in both databases based on the values of one or more attributes and checking if there is a match between the source set of rows and the target one.

Figure 3.17 gives an example of a reconciliation sheet composed using both the bottom-up equivalence pattern to determine which rows were added and deleted and the bottom-up fingerprint pattern to determine which rows differ in their values. In this case, the bottom-up equivalence pattern is not sufficient to determine which rows differ regarding the interest rate since the ideal key <ACCOUNT_ID, LIMIT> cannot be used due to a restructuring of the attribute LIMIT which is maximal in the source and minimal in the target. Thus, the selected fingerprint is the sum of the RATE attribute for a unique ACCOUNT_ID, which leads to conclude that the object 1000225055 changed and the object 1000765208 disappeared.


Transformation/Migration
Reconciliation: Comparison Preparation
Comparison Accounts
Comparison Interests
Reconciliation: Statistics and Migration Errors
Reconciliation Sheet Accounts
Reconciliation Sheet Accounts

T_ACCOUNT_OLD	ACCOUNT_ID	ACCOUNT_TYPE	BRANCH
	1000405201	Customer	HONG KONG
	1000765208	Customer	ZUERICH
	1000225055	Customer	MUNICH
	3000324419	Vostro	BAHAMAS
	5000565097	Nostro	NEW YORK
	9500000084	Profit-Loss	SINGAPORE

T_INTR_OLD	ACCOUNT_ID	UPPERLIMIT	RATE
	1000405201	50'000	3.00%
	1000405201	200'000	3.50%
	1000405201	NULL	3.75%
	1000765208	NULL	3.00%
	1000225055	150'000	3.10%
	1000225055	NULL	3.50%
	3000324419	NULL	0.50%

T_ACCOUNT_NEW	ACCOUNT_ID	ACCOUNT_TYPE	CLASSIFICATION
	1000405201	Customer	5
	1000225055	Customer	4
	3000324419	Vostro	10
	5000165097	Nostro	4
	5000565097	Nostro	4
	9500000084	Nostro	3

T_INTR_NEW	ACCOUNT_ID	LOWERLIMIT	RATE
	1000405201	0	3.00%
	1000405201	50'000	3.50%
	1000405201	200'000	3.75%
	1000225055	0	3.00%
	1000225055	150'000	3.50%
	3000324419	0	0.50%

Reconciliation: Comparison Preparation

Old		New		Match	Equal
Key	Value_1	Key	Value_1		
1000405201	Customer	1000405201	Customer	✓	✓
1000765208	Customer			×	
1000225055	Customer	1000225055	Customer	✓	✓
3000324419	Vostro	3000324419	Vostro	✓	✓
5000565097	Nostro	5000165097	Nostro	✓	✓
		5000565097	Nostro	×	
9500000084	Profit-Loss	9500000084	Nostro	✓	×

Old		New		Match	Equal
Key	Fingerprint_1	Key	Fingerprint_1		
1000405201	10.25%	1000405201	10.25%	✓	✓
1000765208	3.00%			×	
1000225055	6.60%	1000225055	6.50%	✓	×
3000324419	0.50%	3000324419	0.50%	✓	✓

Reconciliation: Statistics and Migration Errors

Section 1: Statistics			
Type	Old	New	
1. Customer	3	2	
2. Nostro	1	3	
3. Vostro	1	1	
4. Profit-Lost	1	0	
Total	6	6	

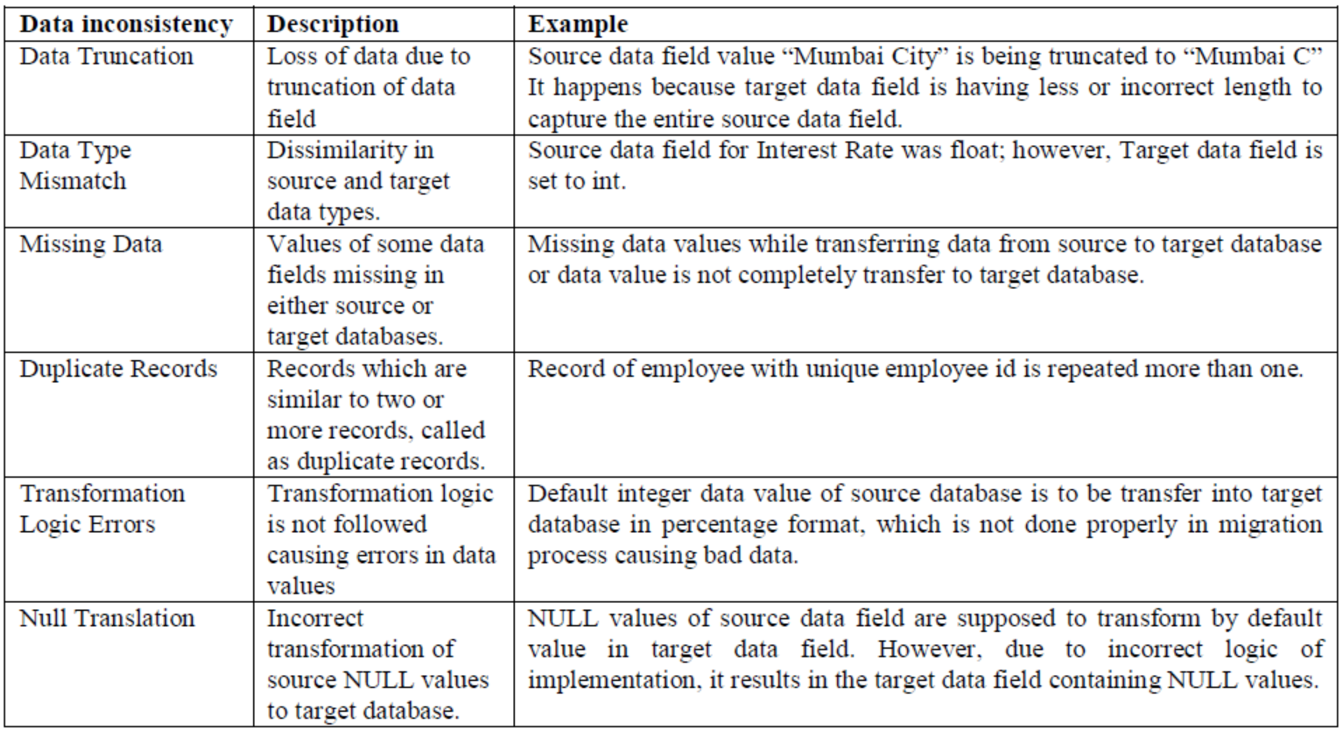
Section 2: Migration Errors					
Old		New		Match	Equal
Key	Value_1	Key	Value_1		
1000765208	Customer			×	
		5000565097	Nostro	×	
9500000084	Profit-Loss	9500000084	Nostro	✓	×

Section 1: Statistics			
Type	Old	New	
Total	7	4	

Section 2: Migration Errors					
Old		New		Match	Equal
Key	Fingerprint_1	Key	Fingerprint_1		
1000765208	3.00%			×	
1000225055	6.60%	1000225055	6.50%	✓	×

Figure 3.17: Reconciliation Sheet Generation Process [Klaus, 2009]

Paygude and Devale propose to automate those verifications within a single tool that takes the mapping between the source and the target database as an input alongside with user queries in order to produce standardized reports. In figure 3.18, they detail the types of inconsistencies that might occur during the data conversion phase [Paygude et Devale, 2011].



Data inconsistency	**Description**	**Example**
Data Truncation	Loss of data due to truncation of data field	Source data field value "Mumbai City" is being truncated to "Mumbai C" It happens because target data field is having less or incorrect length to capture the entire source data field.
Data Type Mismatch	Dissimilarity in source and target data types.	Source data field for Interest Rate was float; however, Target data field is set to int.
Missing Data	Values of some data fields missing in either source or target databases.	Missing data values while transferring data from source to target database or data value is not completely transfer to target database.
Duplicate Records	Records which are similar to two or more records, called as duplicate records.	Record of employee with unique employee id is repeated more than one.
Transformation Logic Errors	Transformation logic is not followed causing errors in data values	Default integer data value of source database is to be transfer into target database in percentage format, which is not done properly in migration process causing bad data.
Null Translation	Incorrect transformation of source NULL values to target database.	NULL values of source data field are supposed to transform by default value in target data field. However, due to incorrect logic of implementation, it results in the target data field containing NULL values.

Figure 3.18: Data inconsistencies [Paygude et Devale, 2011]

3.1.2.4 Program conversion

Program conversion consists of creating the bond between the target database and the source application program(s), thus enabling communication and overall system operation, completing the database migration process. To do so, the source program application(s) must be modified according to two principles [Hainaut et al., 2008].

1. **Comply with the target API of the DMS**: In order to be able to send queries to the database, the adequate data manipulation language and interaction protocols have to be used

2. **Comply with the target database's structure**: In order for the queries sent to be relevant and produce results, the data has to be manipulated in its adequate format which may have changed in regards to its original format during the migration

Three general strategies have been identified to perform program conversion and will be detailed in the next sections.

3.1.2.4.a Wrapper strategy

The wrapper strategy allows the source program application(s) to communicate with the database while incurring very little alterations. The principle is that a new element, the wrapper, is introduced between the program and the database. Wrappers simulate the behaviour of the legacy DMS, translate queries from the application program and map them to the target DMS technology. Figure 3.19 illustrates this concept [Hainaut et al., 2008].

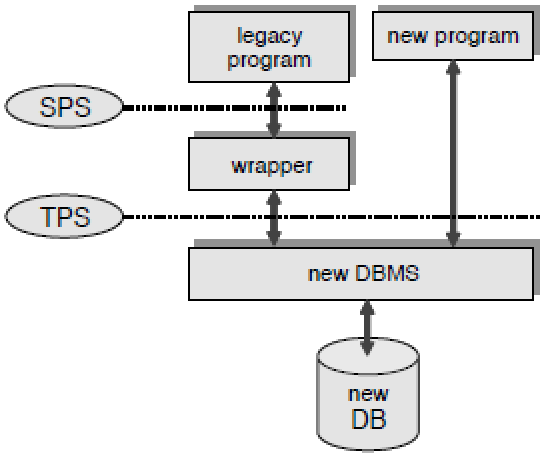
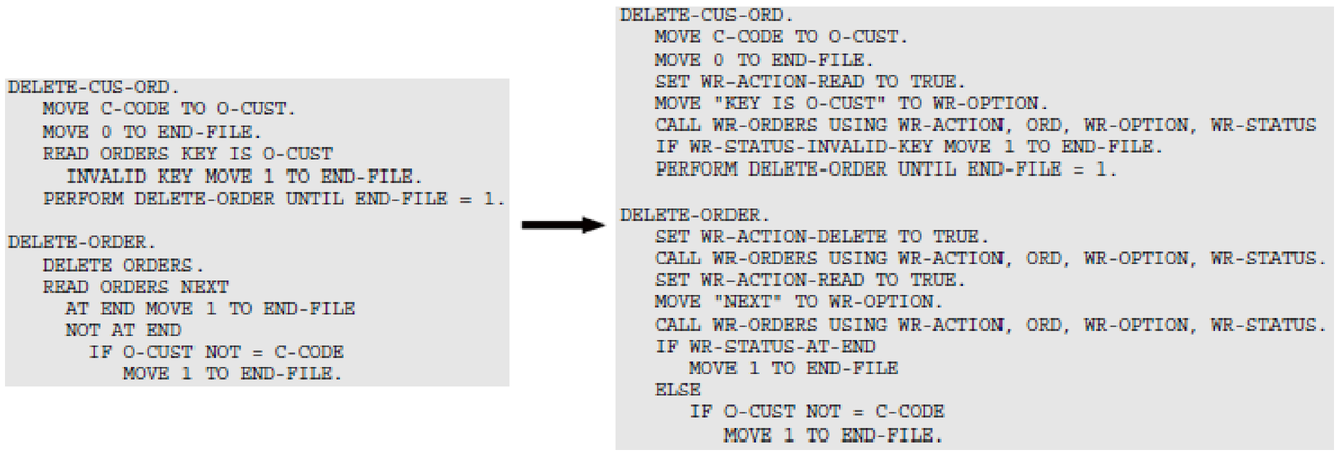


Figure 3.19: Wrapper-based migration architecture: a wrapper allows the data managed by a new DMS to be accessed by the legacy programs [Hainaut et al., 2008]

Figure 3.20 gives an example of a program conversion using this strategy.



```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.

DELETE-ORDER.
   DELETE ORDERS.
   READ ORDERS NEXT
     AT END MOVE 1 TO END-FILE
     NOT AT END
       IF O-CUST NOT = C-CODE
          MOVE 1 TO END-FILE.
```

```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   SET WR-ACTION-READ TO TRUE.
   MOVE "KEY IS O-CUST" TO WR-OPTION.
   CALL WR-ORDERS USING WR-ACTION, ORD, WR-OPTION, WR-STATUS
   IF WR-STATUS-INVALID-KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.

DELETE-ORDER.
   SET WR-ACTION-DELETE TO TRUE.
   CALL WR-ORDERS USING WR-ACTION, ORD, WR-OPTION, WR-STATUS.
   SET WR-ACTION-READ TO TRUE.
   MOVE "NEXT" TO WR-OPTION.
   CALL WR-ORDERS USING WR-ACTION, ORD, WR-OPTION, WR-STATUS.
   IF WR-STATUS-AT-END
      MOVE 1 TO END-FILE
   ELSE
      IF O-CUST NOT = C-CODE
         MOVE 1 TO END-FILE.
```

Figure 3.20: Legacy Cobol code fragment converted using the *Wrapper* strategy[Hainaut et al., 2008]

In order to conserve all manipulations' possibilities, a wrapper has to be built for each source data record type, simulating the behaviour of the legacy DMS for manipulating this object. Thus, the wrapper strategy does not imply deep modification to the source code since only calls to the legacy DMS are replaced by wrapper invocations. The complexity of the wrappers depends on the differences between the source and target physical schemas. On the one hand, if structures are alike, each query will be translated in the target DMS language ; and on the other hand, if structures differ, a query can potentially have an impact on multiple entities. This approach is automatable, wrappers generators already exist for a few DMS.

3.1.2.4.b Statements rewriting strategy

The statements rewriting strategy consists of replacing DML statements of the legacy system by DML statements of the new DMS, providing a direct communication with the database without changing the program's logic. Each DML statement must be located in the source code and translated, thus leading to a complexity and length of the process directly related to the complexity of the application program. This approach is particularly efficient if a Data Access Object or a similar structure is employed to access the data. Figure 3.21 gives an example of a program conversion using this strategy.

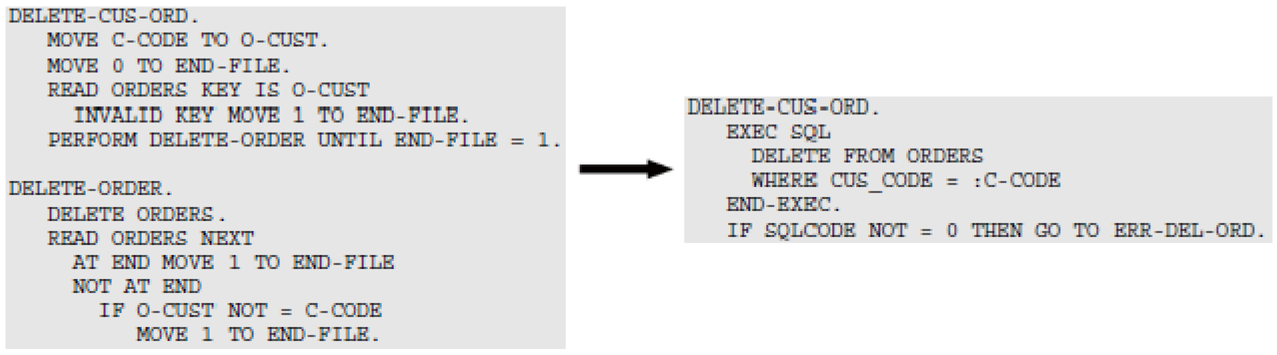
Similarly to the wrapper strategy, the complexity of the translated statements depends on the structural difference between the source and target physical schemas. On the one hand, if structures are alike, each query will be translated in the target DMS language ; and on the other hand, if structures differ, a query can potentially have an impact on multiple entities, thus leading to a more complex translation. This approach is also fully automatable [Hainaut et al., 2008].

3.1.2.4.c Logic rewriting strategy

The logic rewriting strategy differs from the others in the sense that the application program is deeply modified to make the most of the target DMS' possibilities. Thus, the logic of the application is reviewed and adapted to the target DML. The strategy consists of [Hainaut et al., 2008].

1. **Identifying the file access statements**
2. **Identifying and understanding the statements and the data objects that depend on these access statements**
3. **Rewriting these statements as a whole and redefining these data objects**

Figure 3.22 gives an example of a program conversion using this strategy:



```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.

DELETE-ORDER.
   DELETE ORDERS.
   READ ORDERS NEXT
     AT END MOVE 1 TO END-FILE
     NOT AT END
       IF O-CUST NOT = C-CODE
          MOVE 1 TO END-FILE.
```

```
DELETE-CUS-ORD.
   EXEC SQL
     DELETE FROM ORDERS
     WHERE CUS_CODE = :C-CODE
   END-EXEC.
   IF SQLCODE NOT = 0 THEN GO TO ERR-DEL-ORD.
```

Figure 3.22: Legacy Cobol code fragment converted using the *Logic Rewriting* strategy[Hainaut et al., 2008]

The process is complex but produces a system in a total harmony with its new attached database. Automation of this task is not possible, but tools facilitating the work can be developed. The results of this strategy is optimal if the database migration process is complex and the structures differ, due to a deep reverse and forward engineering phases [Hainaut et al., 2008].

```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.

DELETE-ORDER.
   DELETE ORDERS.
   READ ORDERS NEXT
     AT END MOVE 1 TO END-FILE
     NOT AT END
       IF O-CUST NOT = C-CODE
          MOVE 1 TO END-FILE.
```

```
EXEC SQL DECLARE CURSOR ORD_GE_K1 FOR
  SELECT CODE, CUS_CODE
  FROM ORDERS WHERE CUS_CODE >= :O-CUST
  ORDER BY CUS_CODE
END-EXEC.
...
EXEC SQL DECLARE CURSOR ORD_DETAIL FOR
  SELECT PROD_CODE, QUANTITY
  FROM DETAIL WHERE ORD_CODE = :O-CODE
END-EXEC.
...
DELETE-CUS-ORD.
  MOVE C-CODE TO O-CUST.
  MOVE 0 TO END-FILE.
  EXEC SQL
    SELECT COUNT(*) INTO :COUNTER
    FROM ORDERS WHERE CUS_CODE = :O-CUST
  END-EXEC.
  IF COUNTER = 0
     MOVE 1 TO END-FILE
  ELSE
     EXEC SQL OPEN ORD_GE_K1 END-EXEC
     MOVE "ORD_GE_K1" TO ORD-SEQ
     EXEC SQL
       FETCH ORD_GE_K1
       INTO :O-CODE, :O-CUST
     END-EXEC
     IF SQLCODE NOT = 0
        MOVE 1 TO END-FILE
     ELSE
        EXEC SQL OPEN ORD_DETAIL END-EXEC
        SET IND-DET TO 1
        MOVE 0 TO END-DETAIL
        PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
     END-IF
  END-IF.
  PERFORM DELETE-ORDER UNTIL END-FILE = 1.
DELETE-ORDER.
  EXEC SQL
     DELETE FROM ORDERS
     WHERE CODE = :O-CODE
  END-EXEC.
  IF ORD-SEQ = "ORD_GE_K1"
     EXEC SQL
       FETCH ORD_GE_K1 INTO :O-CODE,:O-CUST
     END-EXEC
  ELSE IF ...
    ...
  END-IF.
  IF SQLCODE NOT = 0
     MOVE 1 TO END-FILE
  ELSE
     IF O-CUST NOT = C-CODE
        MOVE 1 TO END-FILE.
...
FILL-ORD-DETAIL SECTION.
  EXEC SQL
     FETCH ORD_DETAIL
     INTO :REF-DET-PRO(IND-DET),:ORD-QTY(IND-DET)
  END-EXEC.
  SET IND-DET UP BY 1.
  IF SQLCODE NOT = 0
     MOVE 1 TO END-DETAIL.
```

Figure 3.21: Legacy Cobol code fragment converted using the *Statements Rewriting* strategy[Hainaut et al., 2008]

Chapter 4

General overview

We have introduced this thesis as being for a part a migration methodology from relational towards non-relational databases and more particularly those which respect Datomic's data model.

Thus, this methodology describes a database platform migration and aims at standardizing the transition from the relational to the non-relational world. This process is intended to support all relational databases in input and several non-relational databases in output. Because of the lack of standardization among non-relational databases, we have decided to select a data model more abstract at logical level to cover more than just one NoSQL database technology.

Figure 4.1 illustrates the different processes described in the methodology and necessary to perform in order to successfully migrate a database composing a system with application program(s).

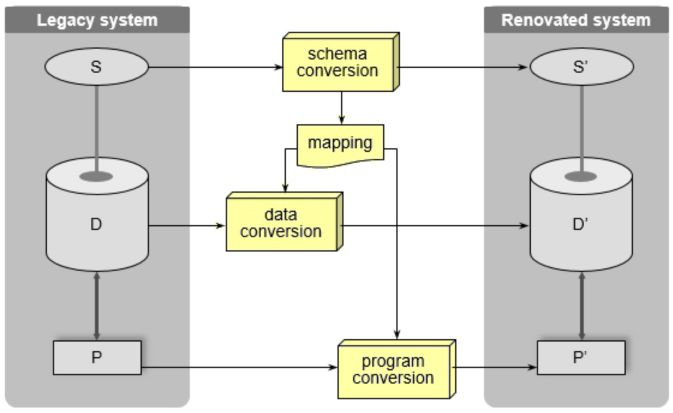


Figure 4.1: Migration schema of our methodology

Indeed, this methodology will be divided in two distinct parts, one dedicated to the schema conversion process and the other to both data and program conversion processes.

4.1 Schema conversion

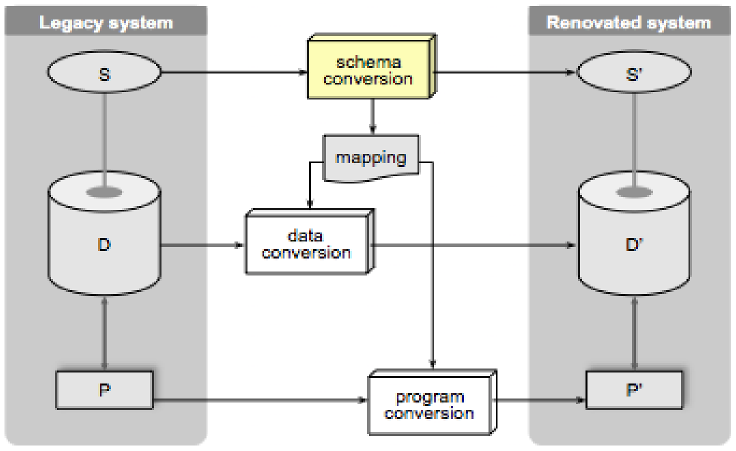


Figure 4.2: Migration schema [Clève, 2009]

First, the schema conversion will be discussed. This is the first phase of our migration methodology and this one involves processes and techniques tinted in yellow in the figure 4.3.

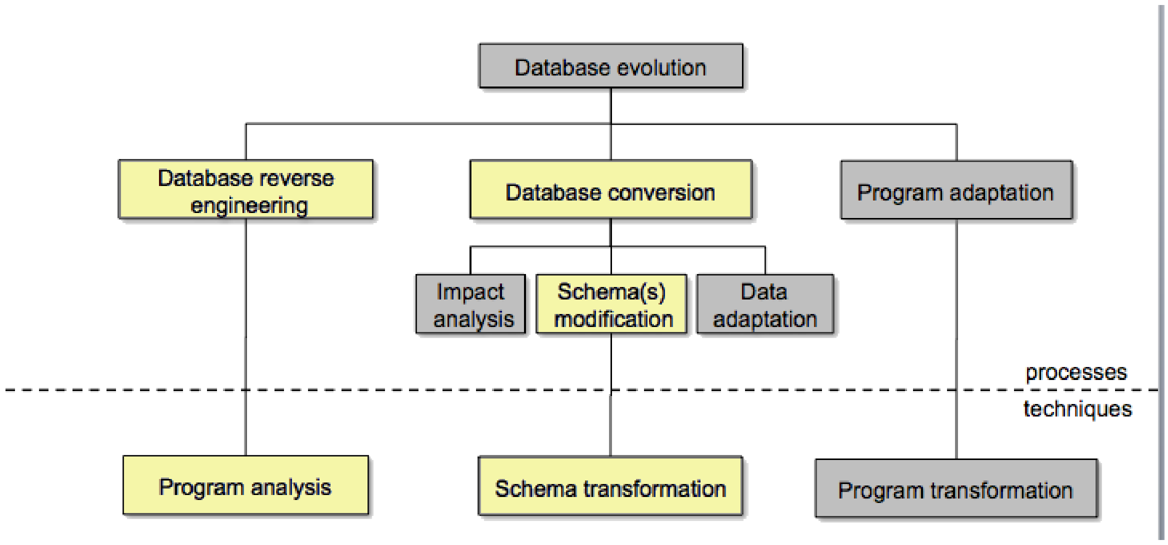


Figure 4.3: Database evolution processes and supporting techniques [Clève, 2009]

Figure 4.4 pictures the general approach chosen for the schema conversion from the source to the target data model in our migration methodology.

This approach is called "semantic" because this allows getting an in-depth understanding about the database and provides high-quality results [Clève, 2009].

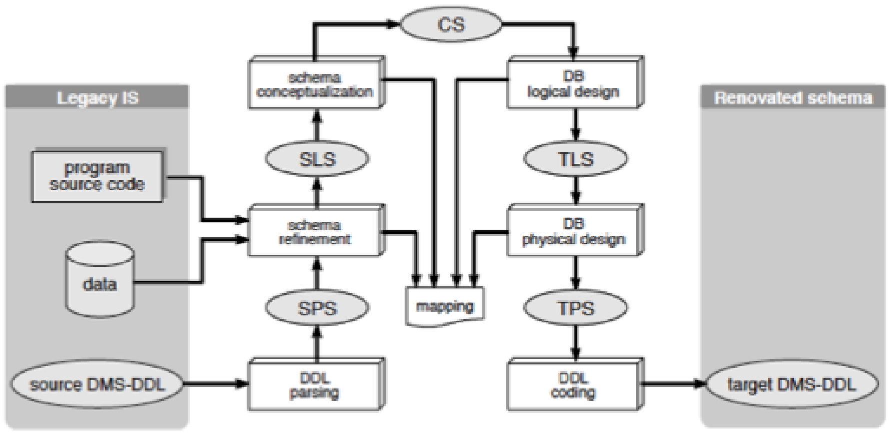


Figure 4.4: Semantic approach [Clève, 2009]

It is performed in successive steps and goes through three different abstract levels: conceptual, logical and physical. This approach is further divided itself into two parts. First, the database reverse engineering which produces the database conceptual schema from the source DDL code and other information sources. Second, the database forward engineering which carries out the transformation in the opposite way. It produces the target DDL code from the database conceptual schema made by the database reverse engineering previously.
By the way, the chapter 6 will be sub-divided according to these two processes.

4.2 Data synchronization

Successfully transform the source database's schema towards a target schema is a critical step in the database migration process but does not mark the end of it. Indeed, the information stored in the source database cannot directly be used in the context of the target database, and at this point the application program is not able to communicate with the target database.

Thus, in order to complete the process, a ***data synchronization phase*** has to occur. This phase consists not only of migrating the content of the source database to obtain a target database up with the state of the system, but also of giving the means to the application program(s) to query and update the target database to maintain it up to date. Figure 4.5 illustrates the situation by highlighting the steps to be applied to complete the migration:

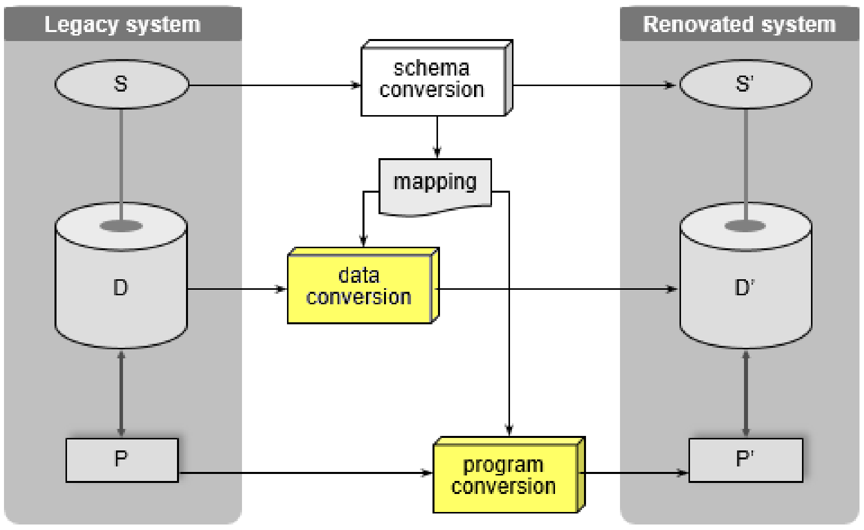


Figure 4.5: Database platform migration [Cleve, 2009]

Two distinct processes can be distinguished and compose the *data synchronization phase*. On the one hand, the *data conversion* process aims at extracting the existing data stored in the source database, transforming it according to the target database's standards, and loading it into it. One the other hand, the *program conversion* process aims at adapting the source code to ensure the communication between the target database and the application program.

In this part, we will cover both *data conversion* and *program conversion* processes. In a first time, we will detail possible approaches applicable to the processes, and we will in a second time suggest a general methodology suitable to the specific case of database migration from relational to NoSQL and specifically Datomic. We will also illustrate this methodology by applying it to our study case: the migration of a subset of the demographic information of the Oscar system towards Datomic.

Chapter 5

Tools support

5.1 Schema conversion

As seen in the previous chapter "General Overview", schema conversion is carried out by successive steps.

The first one is the *database reverse engineering* and there are already a lot of technologies that support this process. Our method does not impose anything. That being said, in the context of our case study, we only used DBMS logs to get traces and DB-Main for building the conceptual schema from the DDL code.

The second step is the *database forward engineering.* From the conceptual schema built by the previous step, the migration method generates corresponding understandable code for Datomic thanks to DB-Main 9.1.6. This is a good choice because it allows integrating plugins for adapting the transformation plan. In order to materialize the transformation plan of our migration method, we developed our own plugins on Eclipse Kepler with Java 6. These plugins are indeed two java classes for the two phases logical design and coding. The physical design is semi-automatable and the automatable part is already implemented by DB-Main.

5.2 Data synchronization

Similarly to the schema conversion, the data synchronization phase is decomposed in two steps.

The first step is the *data conversion* for which we employ an ETL tool (as described in the "State of the art") to migrate data from a source to a target database. Many tools exist, but we narrowed the selection in chapter 2 to those directly supporting migration towards NoSQL. To illustrate the operation of this kind of tools, we used Pentaho Data Integration on our case study: the OSCAR system.

The second step is the *program conversion*, for which we employ both *wrappers* and *logic rewriting* as described in chapter 3. While *logic rewriting* does not necessitate the support of an external tool but the modification of the existing code, *wrappers* do consist of self-developed or generated softwares. However, in the context of this thesis, we remain at a conceptual level for this step and thus do not use nor developed concrete software(s).

Chapter 6

Schema Conversion

6.1 Database reverse engineering

Although names differ between the figure 6.1 and the ascending part of the semantic approach chosen for the schema conversion in the migration method (cf. figure 4.4, p. 89), these picture the same processes.

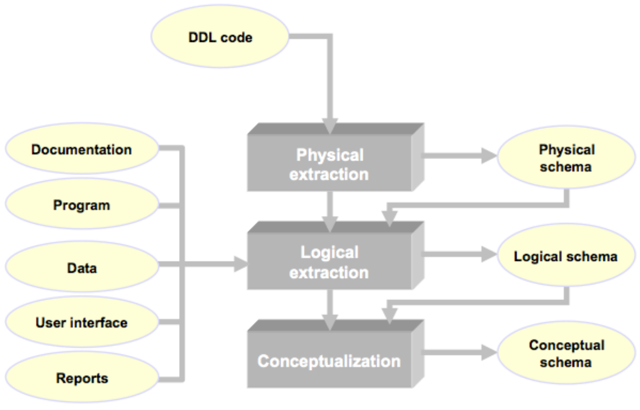


Figure 6.1: Database reverse engineering process [Clève, 2009]

In our case, this process does absolutely not depend on NoSQL, the target technology of our migration method but only focuses on the source technology which is a relational database. It is important to know that the database reverse engineering process has already been experienced a lot of times on this kind of technology. It means that there are already a lot of case studies.

We have already explained what is the goal of the database reverse engineering process and its different phases in the chapter 3. However, these need to be adapted to our migration method.

6.1.1 Physical extraction

First, the physical extraction being automatable does not require a lot of time. This process only needs to chose the right software to extract from the DDL source code, the right physical schema. Many softwares allow that. Nevertheless, we advise to directly use a software being a part of CASE tools and which has this extraction feature. Indeed, the following processes will need a modeling tool and therefore it is more relevant to do everything with the same software.

6.1.2 Logical extraction

Once the physical schema is produced, the logical extraction process enriches it by all implicit structures and namely foreign keys missing. Since we do not want to be dependent on one context, any method described sooner (cf. Chapter 3, Section 3.1.1.1.a, p. 60) can be used depending on the cost allowed. For example, if there is not a lot of money to do the reverse engineering, we advise to use DBMS logs for the SQL tracing and to do the analysis based on heuristics. But clearly, the best way to carry out this process is to use other methods like API substitution or overloading at the logical extraction. But if the program is coded in Java for example, Aspect-based tracing is still a good solution. About the tracing analysis, more is better.

6.1.3 Conceptualization

About the first step of the conceptualization process which is the untranslation, this must first define a range of logical design transformations as pictured for example in the figure 6.2. But instead of going from left to right, this step goes from right to left. The complexity is to chose the right range of transformations according to the logical schema in input. There is no guideline and this step needs a good experience.

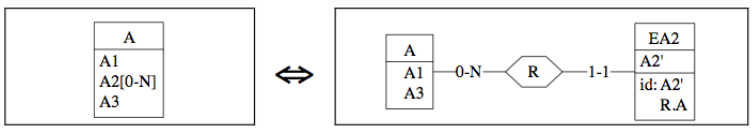


Figure 6.2: Transformation from an attribute to a type of entity [Hainaut, 1996]

Then the second step also needs well trained professionals. As a matter of fact, this step goes far beyond the schema as such because to be correctly executed, this step requires to become familiar with the database history and all people who have worked on this before.

.

That being said, the complexity of the conceptualization process mainly depends on the time allotted on it and goals defined. But more is obviously better.

6.2 Database forward engineering

Now that the database reverse engineering process has been presented as the upstream part of the figure 4.4, p. 89, it is time to describe the downstream part. This section is about the database forward engineering which is already explained in the state of the art (cf. figure 6.3, p. 97). This process is particularly essential to go from the source to the target databases. This aims at proving the possibility of a migration from a relational database towards a non-relational database which implements Datomic as data model. As much as possible, this database forward engineering process has to guarantee that all schema transformations all along this process preserve the initial semantics.

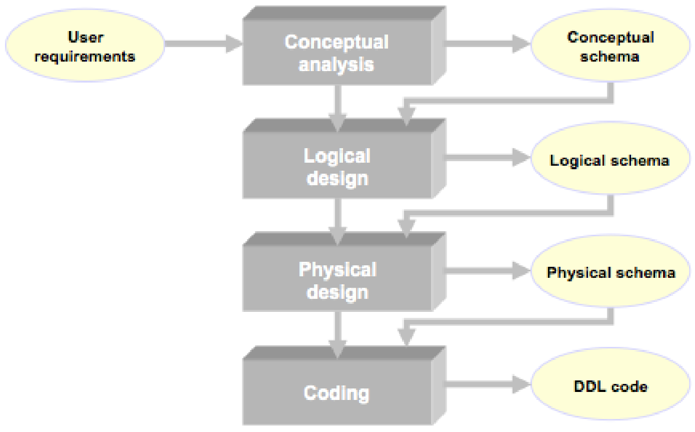


Figure 6.3: Database forward engineering [Clève, 2009]

6.2.1 Conceptual analysis

Since the migration method is about a database platform migration, there is no conceptual modification. The given conceptual schema made by the database reverse engineering before remains unchanged. Though, the database conceptual design is of no benefit.

6.2.2 Logical design

A conceptual schema gives a clear view about the scope studied. But this kind of schema is not understandable for a database management system and its data model, Datomic in this case. The goal of this logical design is to adapt the conceptual schema according to Datomic's data model. It is possible thanks to the elaboration of an action/transformation plan gathering the whole necessary transformations. These transformations must be semantics preserving. In order to prove that, GER formalism will be used as pivot model and it will be essential to show as much as possible that these transformations are reversible. As much as possible because along the establishment of the transformation plan several assumptions will have to be made to ensure the same information capacity.

6.2.2.1 Adapt the attribute names (and quit types of entities)

According to Datomic's data model defined before and the relational databases, the differences are such big between these two worlds that it is important keeping in Datomic some relational concepts to make the transition between them possible. The first concept is about the organization of attributes. In Datomic, there is no attachment to any concept like tables in SQL. The notion of "type of entity" has no mapping in Datomic. But related attributes still need to be considered together. Those are named clusters and are graphically pictured like tables in SQL. About the name of these attributes, it is the first action carried out by the transformation plan because this does not change the semantic, only the syntax. It only means adapting names to map to rules of the nomenclature in Datomic as pictured in the figure 6.4.

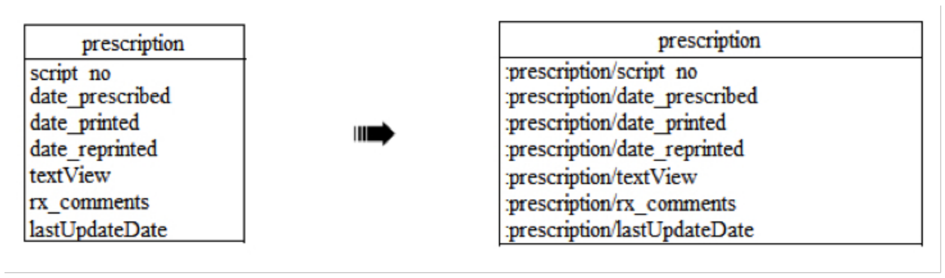


Figure 6.4: Example of names adaptation

Though, the architecture of an attribute name in Datomic is [:<X>/<Y>]. X is replaced by the name of the type of entity where the attributes come from. It allows to keep track of the origin semantic about the type of entity of each attribute. Then, Y is replaced by the same attribute name.

6.2.2.2 Change attributes data type

After adapting attribute names, the transformation plan needs to change data types of the conceptual model to map to Datomic's data types. This is the second step of the transformation plan and the table 6.1 describes the mappings between the different data types keeping as much as possible the same range of values.

There are several possibilities to change data types but only of one them has to be chosen in the context of an automation of the transformation plan. In the cases of "Varchar", "Numeric" and "Float", these choices are underlined and totally arbitrary. Nevertheless,

Data type of DB-Main	Data type of Datomic
Char	:db.type/string
Varchar	:db.type/string
	:db.type/uuid
	:db.type/uri
Numeric	:db.type/long
	:db.type/bigint
Date	:db.type/instant
Boolean	:db.type/boolean
Float	:db.type/float
	:db.type/double
	:db.type/bigdec
Compound	/
Object type	/
User defined	/
Sequence	/
Index	/

Table 6.1: Data types mapping

the range of values is bigger for each data type in Datomic. Thus, the process can import all data from source to target databases. This is lossless but if there is any change about data into the target database, the migration in the opposite way could fail if the data values exceed the range.

6.2.2.3 Transform IDs

The third step is about identifier (ID) in the context of databases. *"An identifiant is a particular property of an object such that no two occurrences of this object exist with the same value"* [Merise, 2014]. In the case of a conceptual schema, IDs may be composed of one or more attributes. On the one hand, if there is only one attribute, it is directly translated towards a Datomic attribute declared as unique. In fact, there is no primary id in this data model. Graphically, this constraint is represented by the symbol “id’” (cf. figure 6.5). However on the other hand, there is no direct mapping if the identifier has more than one attribute. It is impossible to set the uniqueness constraint about a group of attributes in Datomic. Thus either the migration method limits the range of conceptual schema as inputs in the database forward engineering. Or, this constraint is implemented at the applicative layer in order to compensate what Datomic does not implement itself.

That being said, each fact (Datom) added into a Datomic database is accompanied by a unique number (entity id). This unique number is similar to an auto-increment technical ID through all the tables in a relational logical schema to further understand what it is. This identifier is not represented in the conceptual schema and is implicit in the concrete operation of Datomic. But this is represented in the logical schema and considered as primary ID of the cluster because it has a real importance for the relationships. Graphically, it is represented as a primary ID (in the world of relational databases). The transformations about IDs can be seen in the figure 6.5.

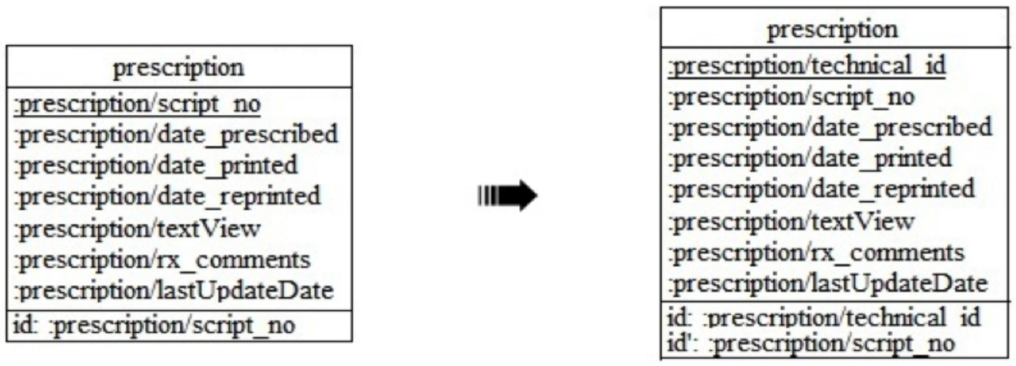

prescription
:prescription/script_no
:prescription/date_prescribed
:prescription/date_printed
:prescription/date_reprinted
:prescription/textView
:prescription/rx_comments
:prescription/lastUpdateDate
id: :prescription/script_no
prescription
:prescription/technical_id
:prescription/script_no
:prescription/date_prescribed
:prescription/date_printed
:prescription/date_reprinted
:prescription/textView
:prescription/rx_comments
:prescription/lastUpdateDate
id: :prescription/technical_id
id': :prescription/script_no

Figure 6.5: Example of IDs transformation

Let us now prove that these transformations do not lose any information capacity and the example pictured into the figure 6.5 is used. The attribute called ":prescription/script_no" is the ID of the type of entity "prescription" at the conceptual level. By definition, it means that it is not possible to have more than one entity with the same value about this attribute. If an attribute is defined at logical level in Datomic as "unique", it means exactly the same thing (c.f. Chapter 2, Section 2.2.2.1). Then adding a technical attribute considered as universal ID is justified because it works that way in Datomic and it does not affect the semantic because it is totally out of the scope. Of course, any other version of the database that the current one cannot be considered because Datomic keeps several states of database content and structure.

6.2.2.4 Transform relationships

This step is the last but not the least. It takes care of relationships. Each relationship in Datomic is represented by a reference attribute (with data type :db/ref) which contains the entity id of fact referred. This entity id is the technical id added at the previous step. But there is a data type problem that arises. Indeed, there is no constraint about what the reference attribute points. It could reference any other cluster than the one intended. Though, the transformation plan assumes that each reference attribute points the right cluster and entities into this cluster. But also, there is another problem in Datomic's data model: all attributes are optional. Indeed, if the range of conceptual schemas is retrained to those with only optional attributes, there will not be anything left. Though, this migration method has to assume in the context of this transformation plan that there is a required attribute constraint in Datomic.

Also, it is important to know that all relationships in Datomic are bi-directional. In other words, for example, an order's :order/customer attribute is a relationship between an order entity and a customer entity and can be retrieved in either direction. It means that the order can be found from the customer or the customer from the order.

That being said, in the context of the transformation plan of the database forward engineering, only these kinds of relationship are taken in consideration:

1. One-to-Many
2. One-to-One
3. Many-to-Many

6.2.2.4.a One-to-Many

Assuming R, a relationship between two types of entity X and Y so that there is only one entity of Y related to each entity of X, and several entities of X related to each entity Y. The transformation method is the following:

1. The relationship R is translated into a Datomic attribute with ":db.type/ref" as data type
2. This attribute is named «:X/Y » or « :Y/X » according to the cluster where it is attached to
   (a) **X/Y** - The attribute is attached to cluster X and its cardinality is one (":db.cardinality/one")
   (b) **Y/X** - The attribute is attached to the cluster Y and its cardinality is many (":db.cardinality/many")

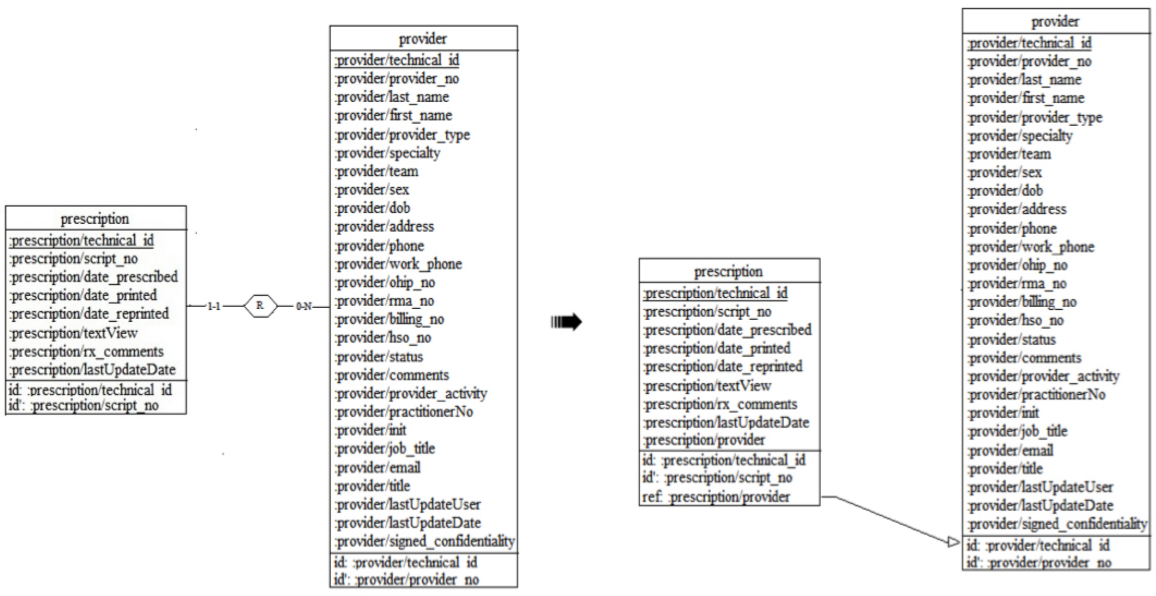


Figure 6.6: One-to-Many transformation: First case

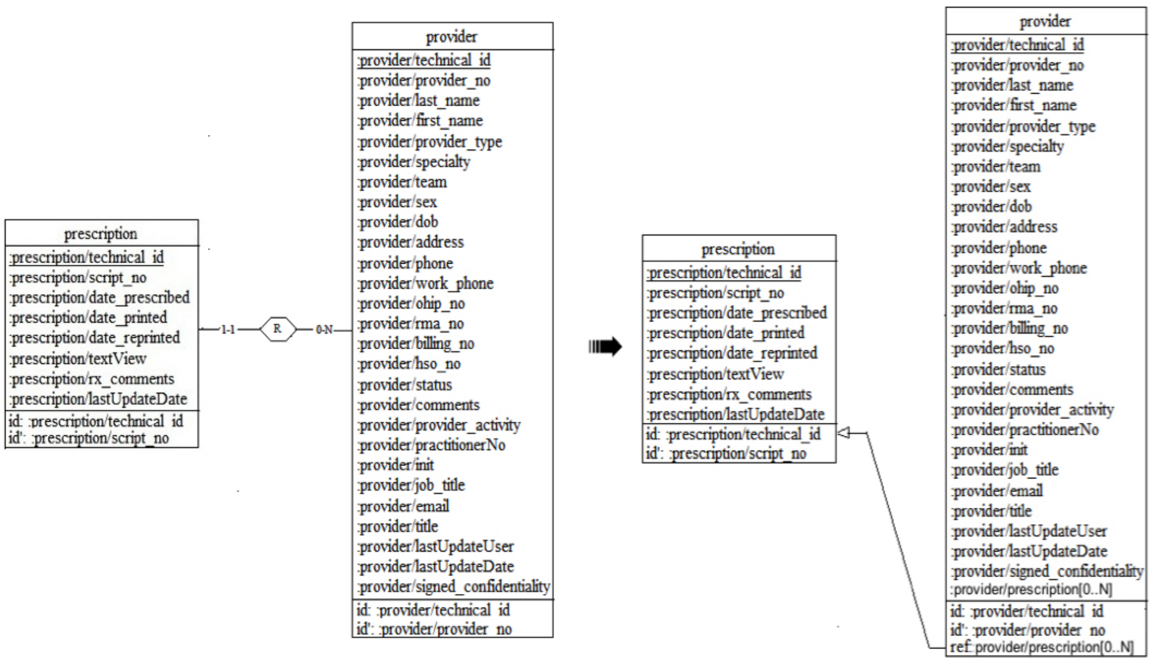


Figure 6.7: One-to-Many transformation: Second case

But is this transformation semantics-preserving in both cases? Let us use the example in the figure 6.6 to prove the first case. Indeed, this operation aims at transforming a relationship into an attribute (foreign key). There is a class of these transformations which is already proved by successive applications of the project-join transformation and the composition transformation (cf. tables 3.4 and 3.5). Hence, here is the transformation T according to the example where P represents the abstract view of the left schema and Q represents the abstract view of the right schema. Also, L and I are subsets of attributes representing the rest.

T	P	prescription,provider:entities
		desc-of-prescription(prescription,prescription/technical_id,L)
		desc-of-provider(provider,provider/technical_id,I)
		R(prescription,provider)
		desc-of-prescription[prescription]=prescription
		desc-of-provider[provider]=R[provider]=provider

	Q	precription,provider:clusters
		desc-of-prescription(prescription,prescription/technical_id,prescription/provider,L)
		desc-of-provider(provider,provider/technical_id,I)
		desc-of-prescription[prescription/provider] ⊆ desc-of-provider[provider/technical_id]
		desc-of-prescription[prescription] = prescription
		desc-of-provider[provider] = provider

Table 6.2: One-to-Many relationship transformation: First case

Since the second case (c.f. figure 6.6) is also a transformation from a relationship to an attribute, this one applies successively the composition and the project-join transformations too.

T	P	prescription,provider:entities
		desc-of-prescription(prescription,prescription/technical_id,L)
		desc-of-provider(provider,provider/technical_id,I)
		R(prescription,provider)
		desc-of-prescription[prescription]=prescription
		desc-of-provider[provider]=R[provider]=provider

	Q	precription,provider:clusters
		desc-of-prescription(prescription,prescription/technical_id,L)
		desc-of-provider(provider,provider/technical_id,provider/prescription,I)
		desc-of-provider[provider/prescription] ⊆ desc-of-prescription[prescription/technical_id]
		desc-of-prescription[prescription] = prescription
		desc-of-provider[provider] = provider

Table 6.3: One-to-Many relationship transformation: Second case

Though, these two cases are semantically equivalent. The first one is children pointing ancestors and the second one is the opposite. However, in the second case, there is a problem about the normalization. Indeed, all prescriptions should be checked to see if each one is different from the others. Even though there is no normalization issue in NoSQL, the first one has been chosen in the context of this transformation plan for its promiscuity to relational model. Indeed, there is no difference about efficiency between both since reference attributes are bi-directional.

In conclusion of One-to-Many relationship, [1..1] –> [0..N] has just been covered and there are other ones. Assuming the first case, let us explain each of it:

- **[0..1] -> [0..N]**: Since all attributes in Datomic are not required, this kind of relationship is easier to translate. According to our example, the only difference about the transformation T is that some prescriptions have no provider. It means that the reference attribute is optional.
- **[0..1] -> [1..N]**: This relationship means that each provider has at least produced one prescription. In the first case, the only difference is that the reference attribute is optional again.
- **[1..1] -> [1..N]**: Even though the abstract views would change to represent that all providers have at least produced one prescription, there is no final difference with the first case.

6.2.2.4.b One-to-One

Assuming R, a relationship between two types of entity X and Y so that there is only one entity (at most) of Y related to each entity of X, and one entity (at most) of X related to each entity Y. The transformation method is the following:

1. The relationship R is translated into a Datomic attribute with ":db.type/ref" as data type. This attribute is mono-value with the uniqueness constraint to meet maximum limits (1) on both sides of the relationship.
2. This attribute is named «:X/Y » or « :Y/X » according to the cluster where it is attached to
   (a) **X/Y** - The attribute is attached to cluster X
   (b) **Y/X** - The attribute is attached to the cluster Y

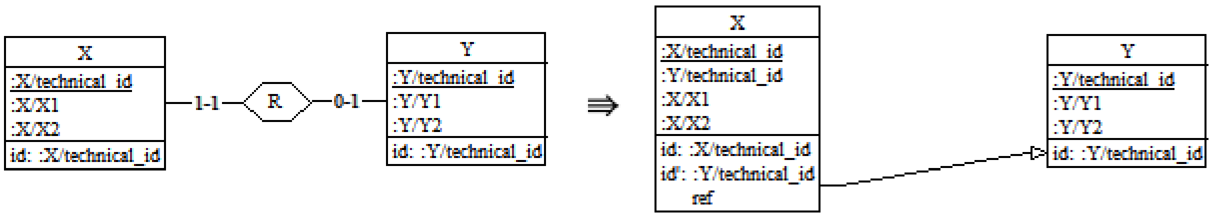


Figure 6.8: One-to-One transformation: First case

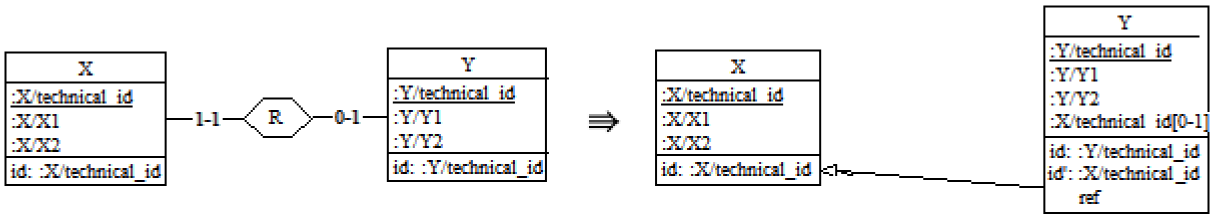


Figure 6.9: One-to-One transformation: Second case

Again the goal aims at proving that this transformation method is semantics-preserving in both cases. The examples pictured in the figures 6.8 and 6.9 are used to demonstrate the case of one optional side whereas the other is required. Once again, these transformations go from a relationship to an attribute (foreign key). According to the application of the project-join and the composition transformations as described in the tables 3.4 and 3.5, both preserve the semantics in the light of their respective abstract view (cf. tables 6.4 and 6.5).

T	P	X,Y:entities
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,I)
		R(X,Y)
		desc-of-Y[Y]=Y
		desc-of-X[X]=R[X]=X

	Q	X,Y:clusters
		desc-of-X(X,X/technical_id,X/Y,L)
		desc-of-Y(Y,Y/technical_id,I)
		desc-of-X[X/Y] ⊆ desc-of-Y[Y/technical_id]
		desc-of-X[X] = X
		desc-of-Y[Y] = Y

Table 6.4: One-to-One relationship transformation: First case

Thus, these two ways to transform One-to-One relationship work and are semantically equivalent. The first case involves the introduction of a required attribute. Without any

T	P	X,Y:entities
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,I)
		R(X,Y)
		desc-of-Y[Y]=Y
		desc-of-X[X]=R[X]=X

	Q	X,Y:clusters
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,Y/X,I)
		desc-of-Y[Y/X] $\subseteq$ desc-of-X[X/technical_id]
		desc-of-X[X] = X
		desc-of-Y[Y] = Y

Table 6.5: One-to-One relationship transformation: Second case

assumption, there is no constraint like that in Datomic. But even though the second case seems easier to implement, this one involves also a big constraint. Indeed, each entity of X has to correspond with one of Y. This constraint cannot be implemented without the applicative layer. In any case, the program will need adaptation and this is the first case which has been chosen in the context of an automation for its promiscuity with the relational world.

To cover all One-to-One relationships, other ones that only [1..1] <–> [0..1] have to be discussed and here are some differences according to the first case:

- **[1..1] <-> [1..1]**: In this case, the relationship is equivalent. In any way the referential attribute is, there will not be any difference between this one and the first case about semantics.
- **[0..1] <-> [0..1]**: Unlike other One-to-One relationships, a referential optional attribute has to be added. There is no implementation of the required attribute constraint in this way. But still needs the implementation of the constraint explained just above.

6.2.2.4.c Many-to-Many

Assuming R, a relationship between two types of entity X and Y so that there are several entities of Y related to each entity of X, and several entities of X related to each entity Y. The transformation method is the following:

1. The relationship R is translated into a Datomic attribute with ":db.type/ref" as data type. This attribute is multi-valued.

2. This attribute is named «:X/Y » or « :Y/X » according to the cluster where it is attached to

   (a) **X/Y** - The attribute is attached to cluster X

   (b) **Y/X** - The attribute is attached to the cluster Y

Creating another type of entity and two references pointing to the other two types of entity like in the relational model could be also considered. But that would be totally contrary to the idea of NoSQL. Indeed, NoSQL has no interest in keeping the normalization. And more than that, this transformation involves a decline in performance.

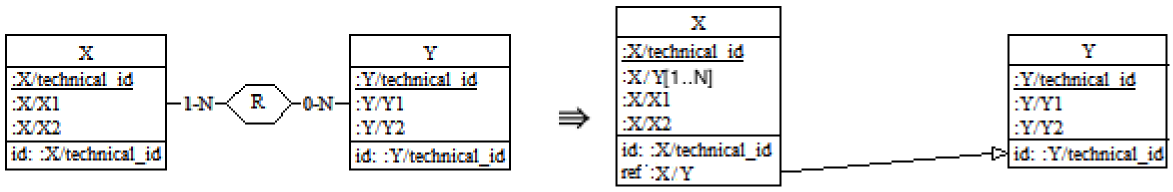


Figure 6.10: Many-to-Many transformation: First case

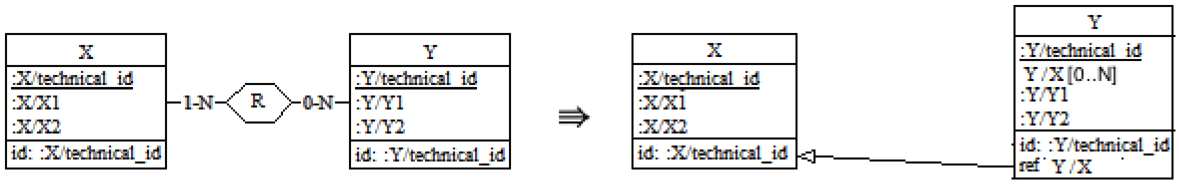


Figure 6.11: Many-to-Many transformation: Second case

Nevertheless, this transformation also belongs to the same class as others because this one involves likewise a relationship into an attribute (foreign key) as pictured in figures 6.10 and 6.11. Again, thanks to these project-join and composition transformations (cf. tables 3.4 and 3.5), these two ways to adapt Many-to-Many relationships are correct according to the initial assumptions.

Since the automation is needed, there is a choice to make. The first case involves a required attribute but the second one involves a big constraint too like in the context of One-to-One relationships. Indeed, it is essential that each add of an entity X is subject of a transaction where an entity Y is added too to preserve the semantic of R. Once again, this transformation plan considers the first case because the mandatory attribute is easier to implement at the applicative layer.

T	P	X,Y:entities
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,I)
		R(X,Y)
		desc-of-Y[Y]=Y
		desc-of-X[X]=R[X]=X

	Q	X,Y:clusters
		desc-of-X(X,X/technical_id,X/Y,L)
		desc-of-Y(Y,Y/technical_id,I)
		desc-of-X[X/Y] ⊆ desc-of-Y[Y/technical_id]
		desc-of-X[X] = X
		desc-of-Y[Y] = Y

Table 6.6: Many-to-Many relationship transformation: First case

T	P	X,Y:entities
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,I)
		R(X,Y)
		desc-of-Y[Y]=Y
		desc-of-X[X]=R[X]=X

	Q	X,Y:clusters
		desc-of-X(X,X/technical_id,L)
		desc-of-Y(Y,Y/technical_id,Y/X,I)
		desc-of-Y[Y/X] ⊆ desc-of-X[X/technical_id]
		desc-of-X[X] = X
		desc-of-Y[Y] = Y

Table 6.7: Many-to-Many relationship transformation: Second case

Finally there are other kinds of Many-to-Many relationships than [1..N] <–> [0..N]. Here are some differences according to the first case:

- **[0..N] <-> [0..N]**: Unlike our choice to take the first case, here, we have to add a referential attribute optional. But that is the only difference.
- **[1..N] <-> [1..N]**: In this case, there is no difference in the final result with respect to the first case chosen when one part is optional and the other is required. That being said, this case involves also implementing the other constraint as explained above.

6.2.2.5 Transformation plan

All the transformations described so far can be gathered into one transformation plan. This one does not cover all possible constructions of a conceptual schema. However, it is already enough to adapt the case study explained later from the conceptual to the logical level according to Datomic's data model.

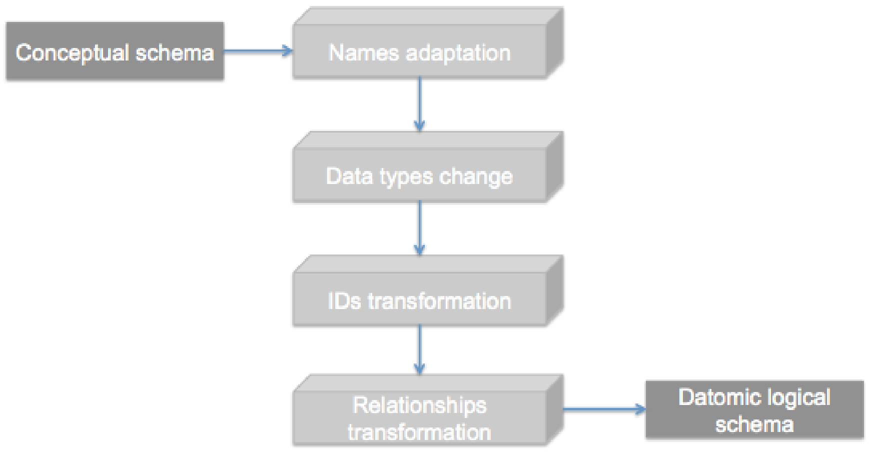


Figure 6.12: Simple transformation plan

That being said, this logical step mainly highlights all the difficulties to adapt something very structured to a technology which is "schema-less". A lot of assumptions have been made to keep proving the semantics-preserving like entity concept, required attribute and others. However, the chapter about program conversion will make sense of this transformation plan by exposing under which context this can be considered as lossless.

6.2.3 Physical design

The physical design is the opportunity to discuss about the optimal implementation of Datomic. This step has two distinct goals. First, this is the last step before the translation into an understandable code. Thus, the physical design needs to produce an understandable physical schema according to the target technology. Although with Datomic, the target technology is already chosen at the logical level because unlike in the relational world, NoSQL is not standardized and there is no common logic between all the non-relational databases. Second, this step has for goal to enrich the logical schema according to some technical requirements to improve the queries speed.

Since the first point is already treated, let us explain the second point, which focuses on one and only issue: cost-benefit. The physical design is guided by this question regarding technical requirements. In other words, is that particular mechanism worth to be implemented according to the performance gain provided? There are two different ways to consider these mechanisms.

6.2.3.1 Internal structure

The internal structure focuses on the relevance of adding some indexes on some attributes. In Datomic, there are already indexes on the technical id and all attributes are declared as unique. Since these attributes are the most often queried, this feature allows speeding the requests up. However, if in the definition of any attribute, ":db/index" is true, this attribute will be indexed too. Also, all referential attributes have to be indexed. In this manner a query which involves more than one cluster will be faster. Also, some other attributes could also be indexed depending on the study of the context concerned.

Despite the system is faster thanks to these indexes, those are purely technical. It means that in each index, there is no value but just reference to it. But Datomic implements fulltext indexes too. It is another kind of index where the value is directly store in it. This index is particularly interesting when it is declared on an attribute where the data type is string. Indeed, it allows to speed queries up on the name searched. Though, this fits very well with other attributes than technical ids, unique and referential attributes.

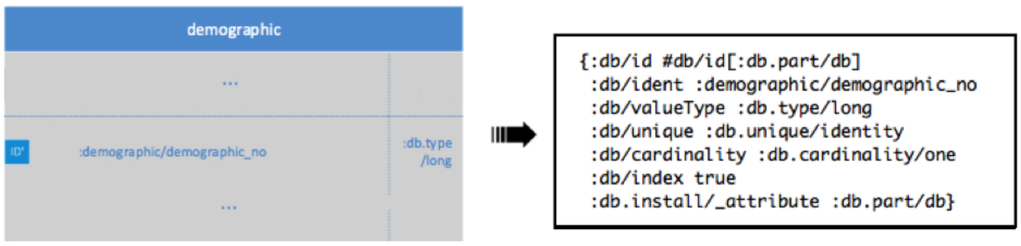
6.2.3.2 External structure

This is not the schema which is concerned when it is about the external structure but everything around Datomic. As described sooner (cf. Chapter 2, Section 2.2.2.1), Datomic is a database management system without database and it needs a storage service to work in production. Each storage service supported by Datomic has been already explained. But which one would be optimal? This is the question of the external structure. And the answer will be discussed in the Performance and Benchmarking chapter of this thesis.

6.3 Coding

Finally, Coding is the last step of database forward engineering which produces understandable code for Datomic's data model from the physical schema.

Datomic only understands data structures as lists of maps. These structures are composed of only one element: attribute. From the physical schema, the Coding process translates every attribute and some of their constraints (uniqueness, cardinality, data type, access key and component) as pictured in the figure 6.13.



```
{:db/id #db/id[:db.part/db]
 :db/ident :demographic/demographic_no
 :db/valueType :db.type/long
 :db/unique :db.unique/identity
 :db/cardinality :db.cardinality/one
 :db/index true
 :db.install/_attribute :db.part/db}
```

Figure 6.13: Example of an attribute generation

The table 6.8 explains each line of this example in more details.

	Attribute properties into the physical diagram	**Attribute properties into Datomic**
Name	:demographic/demographic_no	:db/ident :demographic/demographic_no
Type	:db.type/long	:db/valueType :db.type/long
Uniqueness	Yes	:db/unique :db.unique/identity
Cardinality	[1..1]	:db/cardinality :db.cardinality/one
Index	No	/
Fulltext	No	/
Component	No	/
Attribute location	-	:db/id #db/id[:db.part/db]
Attribute installation	-	:db.install/_attribute :db.part/db

Table 6.8: Explanation about each row of the attribute pictured in this figure 6.13

That being said, the other constraints need to be implemented at the application layer or another one than data as already exposed during the logical design explanation. Here are some features/constraints in 6.9 that there is no way to represent into the Datomic's data model.

Constraints/Features	**Idea of resolution**
Required attribute	Rigour of developer and/or a mechanism that allows rejecting a transaction if it does not insert a fact without an a attribute required into a cluster
Composed id (with multi-attributes)	/
Check	After adding any fact, a daemon could check if the last transaction did not violate the different clauses defined
Trigger	After each transaction, a daemon goes trough the modifications coming from this transactions and checks if the trigger clause is still true
Stored procedures	A function into the program

Table 6.9: Additional constraints/features which need applicative implementations

6.4 Case study

We have chosen a subset of the OSCAR System related to prescriptions area as case study. There was no documentation at all. Though, we had to recover this documentation and as explained before, the best way to do that was to apply the database reverse engineering process.

First, we did not need to extract a physical schema because the one related to prescriptions was already available (cf. figure 6.16). Even if this was a subschema of what we have discovered later during the logical extraction, we will keep this one as the reference because the rest does not introduce any new construction.

Next, we carried out the logical extraction to recover implicit foreign keys and to get rid of physical constructions such as indexes. To do so, we ran the system and executed all operations related to prescriptions while keeping the trace of execution into the logs of the databases chosen. Indeed, several storage spaces are involved by prescriptions into this system. Also, we had no contact nor the budget to apply more methods. Then based on some heuristics, we retrieved a lot of foreign keys which were not declared into the DDL source code as we can see in the figure 6.16.

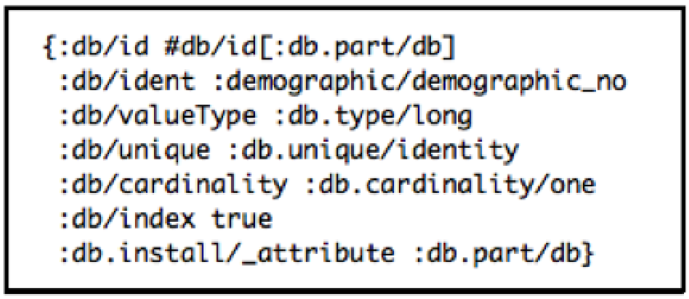
From this logical schema, we have applied the conceptualization process by using the software "DB-main" which already implements the transformation plan for this kind of database reverse engineering, and we obtained this conceptual schema as depicted in the figure 6.17.

Finally, we reduced the conceptual schema (cf. figure 6.17) according to the subschema (cf. figure 6.15) to obtain the conceptual subschema about prescriptions as pictured in the figure 6.18.

Once the conceptual schema was produced, the database forward engineering process must be applied to complete the schema conversion of our migration method.

This process includes the successive application of three steps (i.e. logical design, physical design and coding). Indeed, the conceptual analysis had no interest since this is a database platform migration.

Though, we used the DB-Main plugin that we have developed to automate the database forward engineering process, which can be download on “`http://crespeigneromain.wix.com/thesis`”. This plugin implements the transformation plan described in the section 6.2. By the successive application of the three steps, we have obtained the logical and physical schema of prescriptions area of OSCAR system represented respectively in the figures 6.19 et 6.20. But also, the plugin has generated an EDN file from the physical schema. This file consists of lists of maps which is understandable for Datomic and as much as possible true to the original semantics of physical schema in input. Here is a sample of the EDN file generated from the physical schema. The figure 6.14 pictures the attribute ":demographic/demographic_no" in Datomic.



```
{:db/id #db/id[:db.part/db]
 :db/ident :demographic/demographic_no
 :db/valueType :db.type/long
 :db/unique :db.unique/identity
 :db/cardinality :db.cardinality/one
 :db/index true
 :db.install/_attribute :db.part/db}
```

Figure 6.14: Example of an attribute generation

Since the database forward engineering process described in the context of this migration method is able to produce an understandable code for Datomic from the conceptual schema produced by the database reverse engineering process before, it is allowed to state that the schema conversion covers prescriptions area of the OSCAR system. Even though, this statement has to be qualified with all assumptions taken to reach this point.

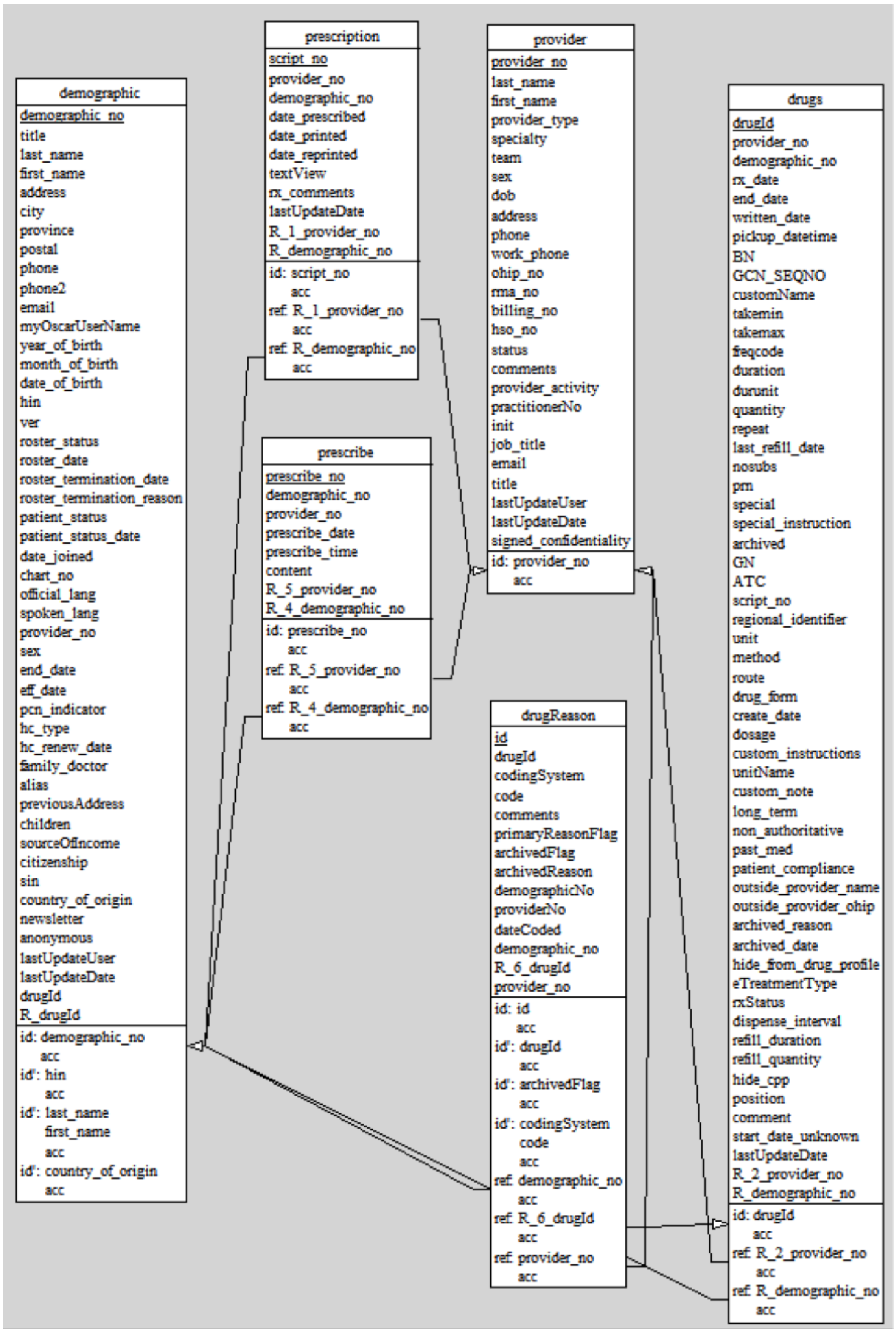


Figure 6.15: Physical subschema of OSCAR database about prescribing

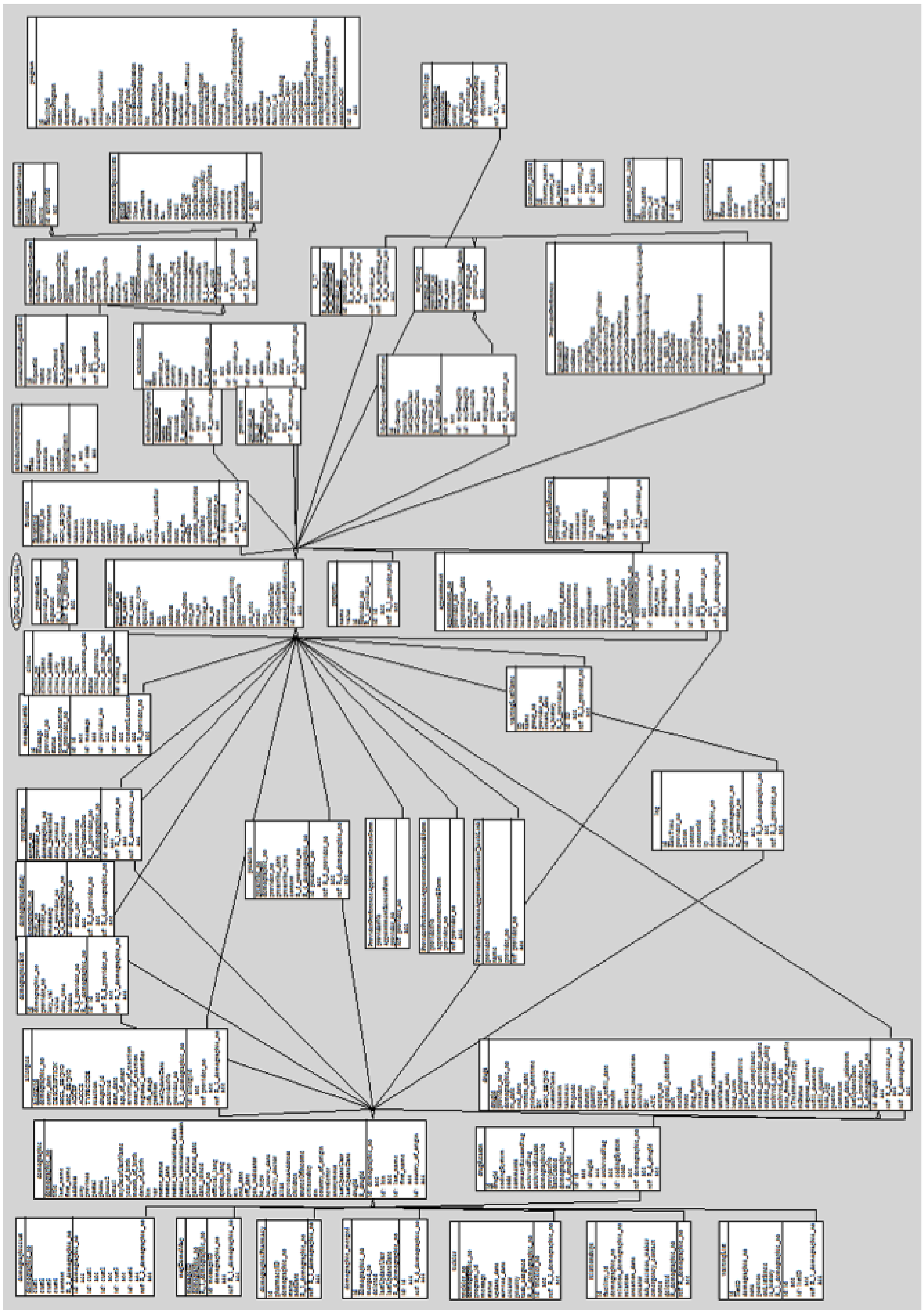


Figure 6.16: Logical schema of OSCAR database about prescribing

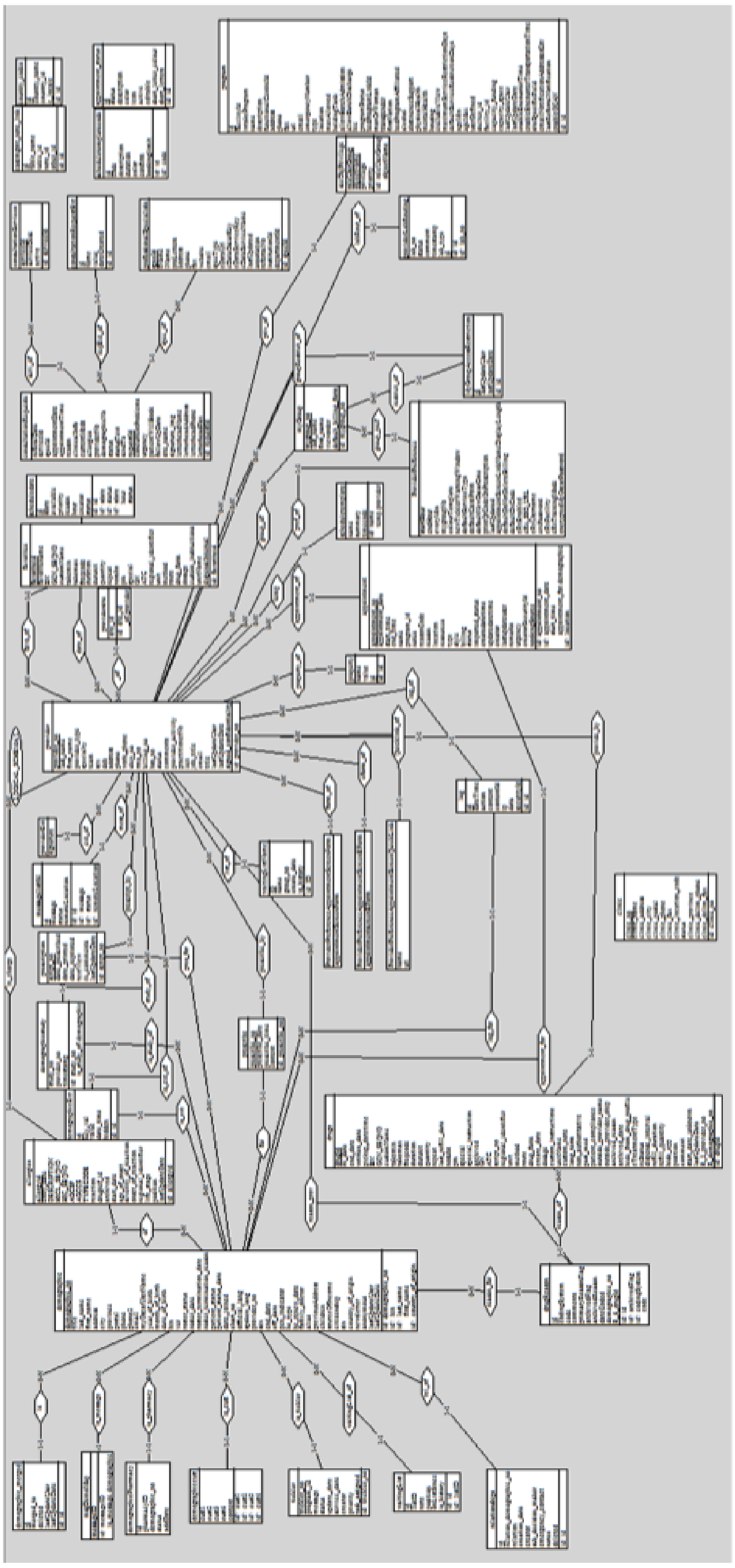


Figure 6.17: Conceptual schema of OSCAR database about prescribing

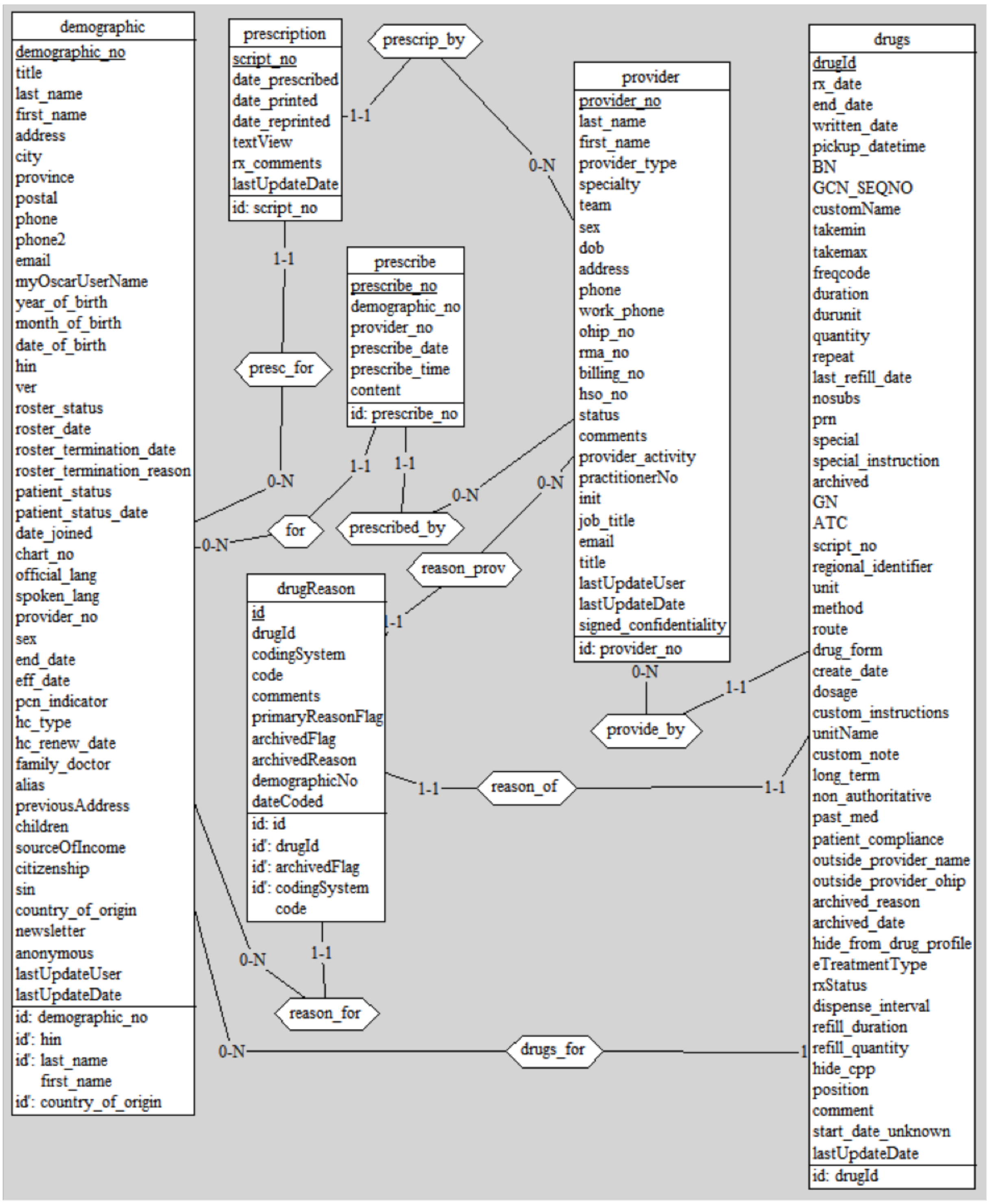


Figure 6.18: Conceptual subschema of OSCAR database about prescribing

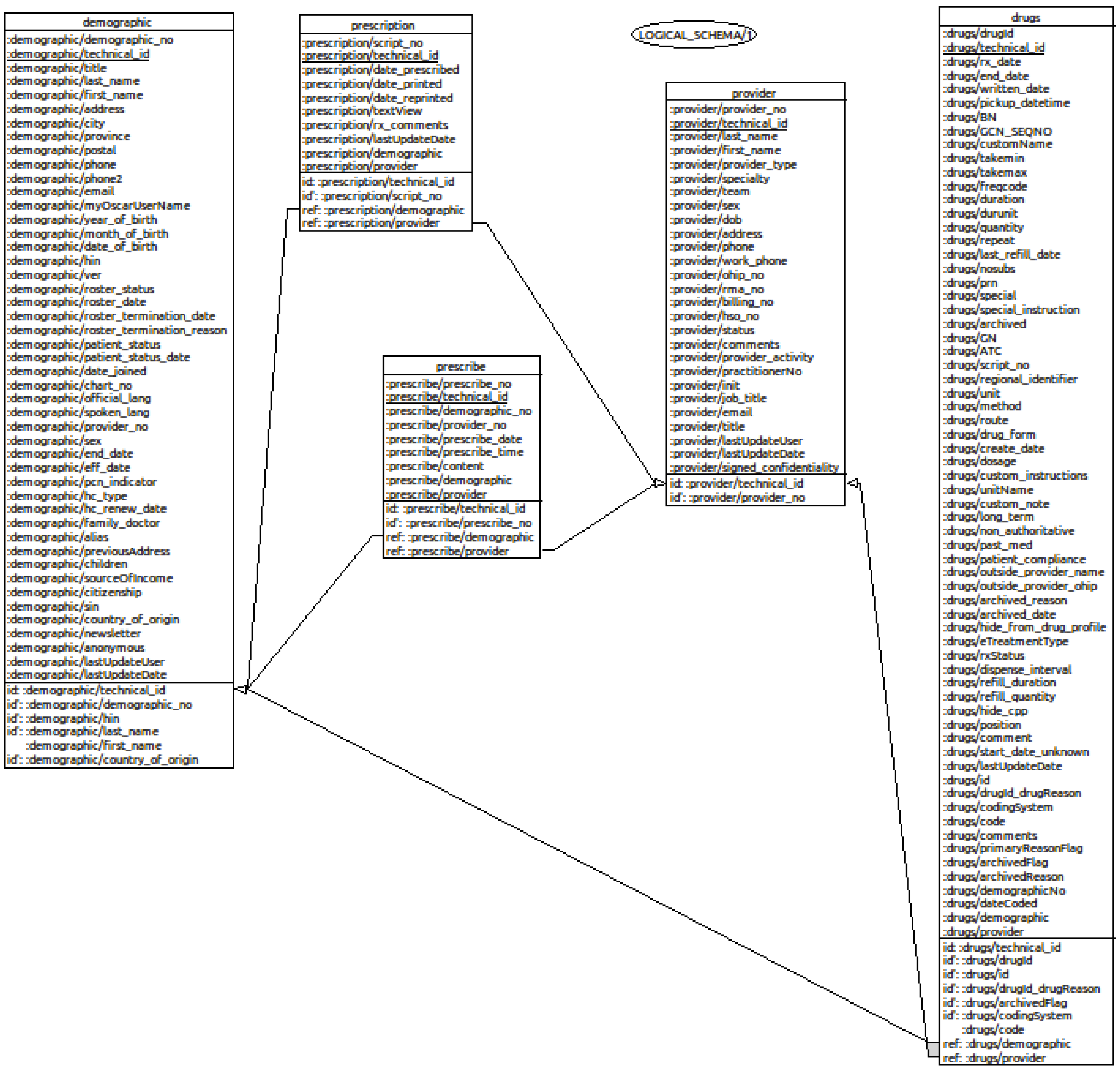


Figure 6.19: Logical schema of OSCAR database about prescribing according to Datomic

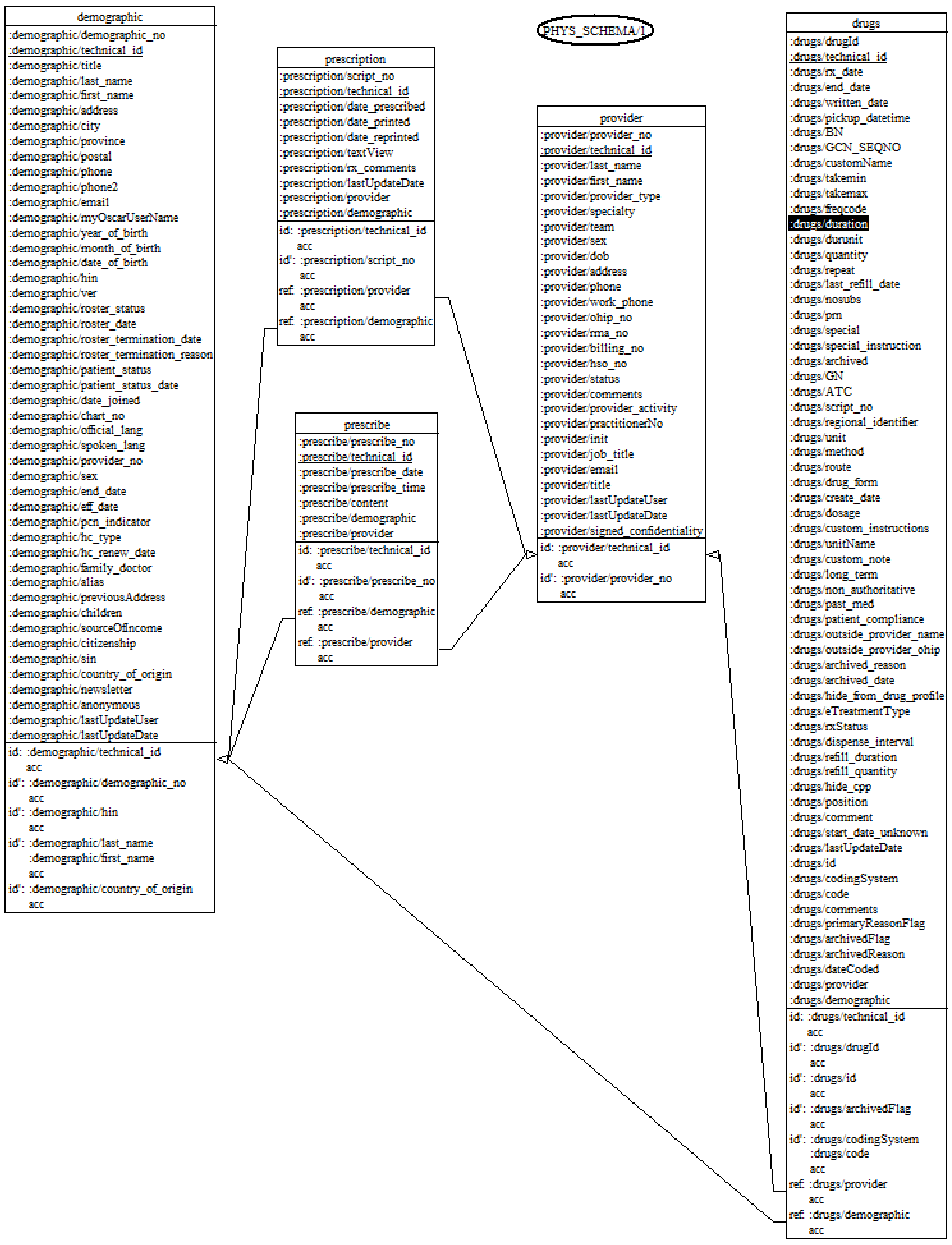


Figure 6.20: Physical schema of OSCAR database about prescribing according to Datomic

Chapter 7

Data conversion

7.1 Overview

This phase takes place during the database migration process after the target database schema has been determined. Thus, we dispose of an existing database containing data, an empty target database and source program(s). The objective of this phase is to convert the data contained in the source database to fit the structure of the target database and insert it into it. As observed in the state of the art, the data conversion phase of a database migration relies systematically on an Extract-Transform-Load process which we represented by the figure 3.12:

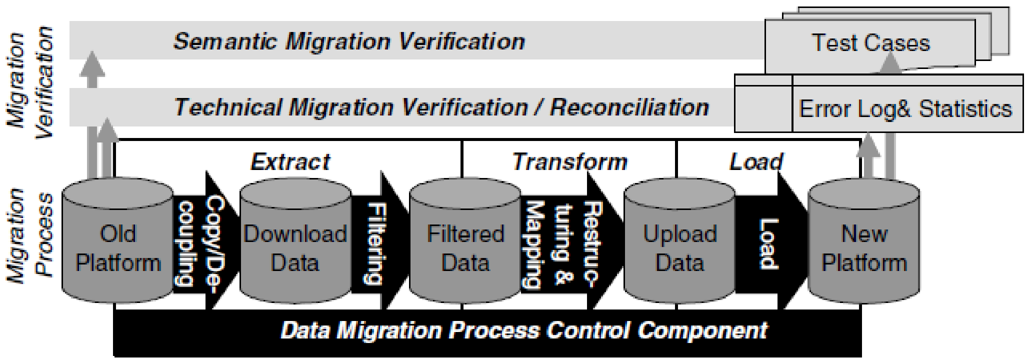


Figure 3.12: Generic Migration Architecture (repeated from page 75)
[Klaus, 2009]

The *extract process* will first copy the data to decouple it from the application and then filter it to transform and load only relevant and necessary information.

The *transform process* will adapt the extracted data's structure in order to fit to the schema obtained after the schema transformation. In the context of this thesis, our case study aims at adapting the original relational data to the Datomic schema we obtained and that is described in the previous chapters.

The *load process* will insert the transformed data into the target database, thus completing the data conversion phase.

This pattern is very general and systematically applied when it comes to data migration. However, multiple strategies exist and extend the global and abstract concepts of Extract, Transform and Load. In this chapter, we will first detail the different existing data migration methods, then motivate our choice of one of them and furnish a way to select the appropriate tool related to this method in the case of migration towards NoSQL. We will at last illustrate this method by applying it to our case study, the migration of a subset of the demographics information from the OSCAR system.

7.2 Data conversion approaches

We have given a general pattern of data conversion for database migration. In its concrete application, multiple different strategies can be distinguished. We will classify here general groups of data conversion methods, then propose a general method that will be detailed when applied to our case study. Shweta, Pratiksha and Bendre identify three different methods to migrate the data [Shweta et al., 2014]:

7.2.1 Data migrated manually beforehand

This method is performed when the old system cannot provide data required by the new system. The raw data has to be reworked before switching to fit in the target database. It is very costly and has a high error ratio [Shweta et al., 2014].

The Row-Oriented Implementation Paradigm A script is written for each object type and encloses the mapping between the object in the old database and its counterpart structure in the target database within a loop. Thus, the migration is oriented on each row sequentially. The paradigm specifies the migration in an imperative way, easily understandable and hiding data complexity [Klaus, 2009].

The Set-Oriented Implementation Paradigm Data are treated by sets, which are data structures more complex than simple rows. A script is written for each set and defines the mapping between the set in the old database and its counterpart structure in the target database [Klaus, 2009].

7.2.2 Data generated by new system afterwards

This method designates the generation of data by the new system after switching from the old database to the new one [Shweta et al., 2014].

7.2.3 Data migrated by tools beforehand

This method consists of using a data migration tool called *ETL* (Extract, Transform, Load) that will perform the steps described earlier. Multiple tools of this kind proposing a wide range of functionalities exist [Shweta et al., 2014]. Those kinds of tools will be detailed in the next subsection.

7.2.4 Comparison

In general, those approaches are not necessarily mutually exclusives and we will now discuss of a way to apply and combine them in a strict method. In order to compare relevant approaches, we will first sort them. In the context of this thesis, we will put aside for now the approach 7.2.2 since it typically takes place once the migration has been successfully performed, after the program transformation phase, which goes further than the subject we are working on here.

Moreover, for more clarity, we propose to decompose the *Data migrated manually beforehand* in two groups of methods, and to rename the *Data migrated by tools beforehand*, which results in the following classification:

1. **Manual data migration**: Designates data migration methods that manually transfer data from one database to the other, possibly copying and pasting information or literally typing each row
2. **Semi-automatic data migration**: Encompasses Haller's *Row-Oriented Implementation Paradigm* and *Set-Oriented Implementation Paradigm*, designates data migration methods that are realized by implementing one or multiple scripts that will be manually orchestrated and called
3. **Automatic data migration**: Designates data migration methods that are realized automatically by the use of an ETL tool

To compare those concepts, we build a comparative table 7.1 of those sorted methods that will guide the construction of this method.

All those approaches are relevant in the context of migration towards NoSQL and thus are all candidates for being a part of a data conversion method. The important costs and risks of potential errors and data loss that can be observed for the manual and semi-automatic approaches tend to suggest the use of an automatic approach (i.e. an ETL tool) self-developed or already existing, at least for relatively large databases. This choice is comforted by both academical world and business experiences [Clève et al., 2010] [Devart, 2013]. That said, it is conceivable that some data structures might be too complex to be parsed by an automatic tool. In this case, we propose using of a manual or semi-automatic approach for those data types only. The next section will detail the choice between a self-developed solution and an existing one, and in this case chosen on which criteria. In all cases, a verification and validation phase is required in order to ensure the integrity of migrated data. Thus, the method we suggest for relational data conversion towards a Datomic structure is the following:

1. **Select or develop an ETL tool**
2. **Perform data migration using this tool**
3. (If some data types are too complex to be directly migrated using the tool) **Employ the manual (purely manual or semi-automatic) approach for those types**

Method group	Cost	Risk	Details
Manual data migration	Very high	Very high	The cost is very high for each data conversion and since there is a heavy human involvement, the risk of errors and data loss is very high too. Thus it is a suitable solution for small databases very well known by the team and whose manual migration cost is lower than the cost of understanding/developing a tool or implementing a semi-automatic solution.
Semi-automatic data migration	High	High	There is still the need for important human involvement and high understanding of the database structure for the team, but the partially automatic approaches slightly reduce costs and risks. Indeed, the cost will now reside in the implementation of scripts and possibly in understanding the database.
Automatic data migration	Variable	Reduced	Depending on the use of an existing tool or the internal development of a full ETL tool, the cost will reside in understanding the tool or developing it. Risks of data loss and errors remain present but are reduced by the fully automatic approach.

Table 7.1: Data migration methods comparison

7.3 ETL tools

In this section, we assume that the choice of relying principally on an ETL tool is made. At this step of the data conversion phase, two options are possible and will be discussed now:

1. **Developing an ETL tool**
2. **Employing an existing ETL tool**

ETL tool development is a complex field of study and has been the subject of multiple research papers. Example of those research papers are the work of P. Vassiliadis who contributed at the redaction of *A Methodology for the Conceptual Modeling of ETL Processes* [Simitsis et Vassiliadis, 2003], *A generic and customizable framework for the design of ETL scenarios* [Vassiliadis et al., 2007] and *Deciding the Physical Implementation of ETL Workflows* [Tziovara et al., 2007]. Those articles describe a complete process of ETL-scenario implementation, from the conceptual design to the physical implementation. Since a self-developed ETL tool is not the approach we suggest in our method, the development process of an ETL will not be detailed in this work and we will focus on benefits and costs of the global implementation.

In *Data Warehousing Tool Evaluation – ETL Focused* [Rodriguez et al., 2012], Rodriguez, Lawson, Molina and Gutierrez confirm the heavier cost of self-developing a solution in comparison to the understanding and use of an existing tool. Furthermore, most of the experiences and testimonies concerning this matter come from the industry, scientific papers generally assuming the choice is already made and preferring to analyse how to design and implement a tool or how to select an existing tool. Yet, we have taken care to avoid ETL vendors' articles and to rely only on independent experts' or customer-side opinions.

Beyond the complexity drawback, we can also identify advantages of the implementation of a self-developed solution. Indeed, a custom tool allows more freedom in its design and permits narrowing the development complexity to the source and target data structures of the migration. Furthermore, depending on the users' needs, typical ETL tool features such as a Graphical User Interface might be skipped. These characteristics, added to the synthesis of existing tools' strengths and weaknesses realized by Joy Mundy [J., 2008], Data Warehouse/Business Intelligence consultant for the Kimball Group, allow us to erect the following table 7.2:

	Self-developed ETL tool	Existing ETL tool
Advantages	- Warranty of the understanding of the solution - Freedom in the development - Solution suited to the exact needs	- Visual flow and self-documentation - Structured system design - Operational resilience - Data-lineage and data-dependency functionality -Advanced data cleansing functionality -Performance
Drawbacks	- Complexity and costs of development - Possibly less advanced functionalities	- Potential software licensing cost - Uncertainty - Reduced flexibility

Table 7.2: ETL solutions comparison

In general, we can conclude that self-developed solutions tend to suit small and simple projects (i.e. few data types, simple structures, small-sized databases), recurrent migrations and ETL-experienced users. Indeed, small and simple projects will lead to less and simpler transformations to implement. What is more, a self-developed tool is the most profitable when it is used multiple times, and recurrent migrations will lead to a team experienced in migrating data and familiar with ETL concepts. In those cases, the cost of developing a custom solution will be smaller than the cost of understanding and/or paying a fee for an existing solution. The use of an existing ETL tool will be more interesting for punctual and/or complex migrations, performed by less experienced teams, thanks to its advanced and more accessible functionalities.

We make the assumption in this thesis that the system to migrate has a certain level of complexity, comparable to the complexity level of our case study (the OSCAR system) and that users are not ETL experts. Thus, we will opt in the next steps for the use of an existing tool. Furthermore, this thesis aims at migrating towards a NoSQL database, more specifically a Datomic database. Those technologies being relatively young, not all ETL tools can be used for data conversion. However, the proportion of ETL tools proposing a NoSQL output is increasing. We depicted in the Technological Background the main ETL tools supporting migration towards NoSQL.

7.3.1 ETL tool selection

There is a wide range of existing ETL tools, each with its strengths and weaknesses. Some tools will perform better than others depending on the type, amount or structure of data they take as an input or produce as an output. In this section, we will define a selection method, and then follow it to select a suitable ETL tool for data migration from a relational database to a NoSQL and Datomic database.

In their article *Engineering trade study : extract, transform, load tools for data migration*, Henry, Hoon, Hwang, Lee and DeVore defined a strict method for comparing and selecting ETL tools. This approach consists of three key elements : *Figures of Merit*, *Criteria* and *Test scenarios* [Henry et al., 2005].

Figures of Merit Figures of Merit represent the general measures of quality that will allow the user to evaluate the ETL tool. There are six of them [Henry et al., 2005]:

1. **Cost** The cost of the use of a system represents both the eventual subscription or usage fee and all implicit using costs
2. **Ease of use** The easier a tool will be to use, the more profitable it will be for users
3. **Flexibility** Flexibility represents the width of the range of features and functionalities and the ability of a tool to be customized for specific uses
4. **Robustness** Robustness represents the ability of a tool to react to unplanned environment changes
5. **Scalability** Scalability represents the ability of a tool to cope with different volumes of data and remain reactive
6. **Speed** The speed represents the overall performance of the tool

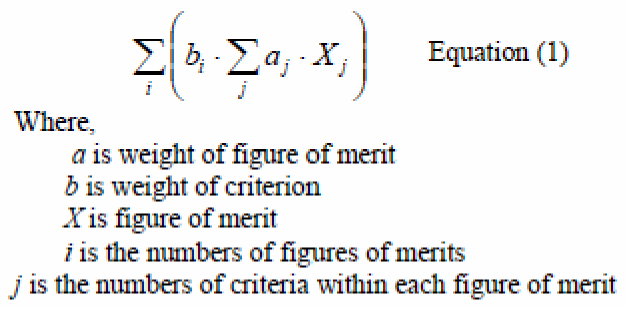
Criteria Criteria are the concrete elements of the tool that will be evaluated regarding to the figures of merit. Because of the complexity of ETL tools, criteria will be gathered in eight general groups [Henry et al., 2005]:

1. **Product Architecture** Related to the implementation of the product, encompasses the installation process, the platform support, the recovery logic, the restart logic, the intermediate storage, the parallel support and the documentation
2. **Data Support** Related to the types of data supported by the tool, encompasses data formats support, data type support and real-time data

3. **Data extraction** Related to the operation of the data extraction phase of the ETL tool
4. **Data Transformation** Related to the operation of the data transformation phase of the ETL tool, encompasses pre-built rules and transformations, rule-based transformations, support for all basic mathematical and statistical functions, basic data cleansing functionality availability, recursive processing support and code/scripting support
5. **Data Loading** Related to the operation of the data loading phase of the ETL tool
6. **Matching** Related to the ability to match and merge information between multiple data sets
7. **Metadata Management** Related to the management of Metadata, encompasses extensibility, open storage format, metadata sharing and content reporting
8. **Development Environment** Related to the quality of the environment the user will use to perform the migration, encompasses the Graphical User Interface support, the command line support, the integrated toolset, the sequential processing, the debugging support, the ETL reporting, the centralized administration and the scheduling

Test scenarios The ETL tool will be tested by following defined test scenarios that will challenge criteria regarding figures of merit. Those scenarios should ideally be similar in terms of data and functionalities to the final ETL processing. Also, they have to be effective in testing each criterion and scalable [Henry et al., 2005].

Method Depending on the needs of the users, more or less importance will be given to each figure of merit and criterion. To do so, weights will assigned to each element, thus representing its relative importance in the general evaluation. The sum of the weights of all criteria must equal 1, and the same goes for the sum of the weights of all figures of merit. Put in another way, the weight repartition follows this equation [Henry et al., 2005]:

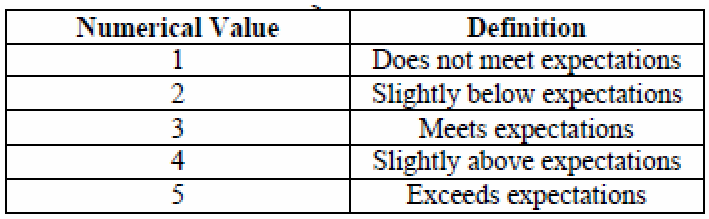


$$\sum_i \left(b_i \cdot \sum_j a_j \cdot X_j \right) \qquad \text{Equation (1)}$$

Where,
a is weight of figure of merit
b is weight of criterion
X is figure of merit
i is the numbers of figures of merits
j is the numbers of criteria within each figure of merit

Figure 7.2: Weights repartition [Henry et al., 2005]

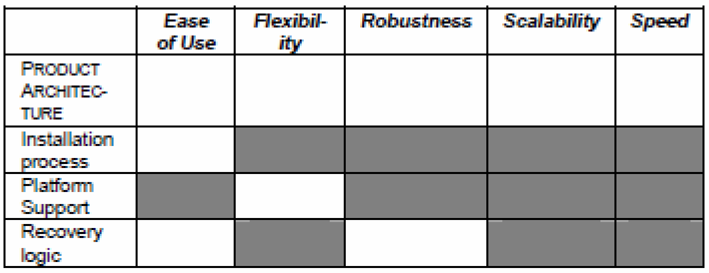
Users will then follow the test scenarios and progressively attribute values from 1 to 5 to each criterion from each figure of merit. Values represent the degree of satisfaction according to the following scale [Henry et al., 2005]:



Numerical Value	Definition
1	Does not meet expectations
2	Slightly below expectations
3	Meets expectations
4	Slightly above expectations
5	Exceeds expectations

Figure 7.3: Quantitative scale [Henry et al., 2005]

Scores will then be stored in an evaluation matrix such as depicted in fig 7.4 [Henry et al., 2005]:



	Ease of Use	*Flexibil-ity*	*Robustness*	*Scalability*	*Speed*
PRODUCT ARCHITEC-TURE					
Installation process		▨	▨	▨	▨
Platform Support	▨		▨	▨	▨
Recovery logic		▨		▨	▨

Figure 7.4: Evaluation matrix [Henry et al., 2005]

The fulfilled evaluation matrix will then be used to compare the tested ETL tools' results and select the most fitting one for the data migration in its particular context.

In the context of defining a database migration from relation towards NoSQL (Datomic) method, we will now use those concepts and assign weights to figures of merit and criteria in order to ease the comparison between ETL tools. To do so, we have to go further than the technological context and make multiple assumptions. We suppose the migration corresponds to a generic case we specify and is performed by a generic user profile we also define. Here are the assumptions we make:

1. The migration is a one-time migration of a rather complex system including a wide database structure and a consistent volume of data from a relational database to a NoSQL database
2. The user is not an expert in database migration and has limited resources (i.e. the user cannot or does not aspire to invest too much in an ETL tool he will not use anymore once the data is migrated)

Those two assumptions allow us to build tables 7.3 and 7.4 that assign weights to the different figures of merit and criteria, thus providing a mean to compare tools and narrow the possible ETL tools to be employed for such a migration.

Yet, though this weights assignment is fitting for the particular context we described, it remains very specific and the slightest change in the migration environment can lead to a different analysis of priorities. We do not pretend these tables to be universally fitting whatever the context but we consider them relevant in the particular case of this thesis and regarding the assumptions we made.

Figure of merit	**Weight**	**Justification**
Cost	0.20	As assumed, cost is an important metric and even after filtering paying solutions, usage costs still matter
Ease of use	0.20	As assumed, the user is not an expert in data migration and thus needs an understandable interface
Flexibility	0.20	The relational to NoSQL migration is not yet very common in the industry, thus the tool needs to be flexible enough to provide a way to migrate towards an uncommon technology, specifically Datomic
Robustness	0.10	Since the migration is a one-time process and we aim in a first time at maintaining both database working, the data conversion process can possibly be restarted in case something goes wrong
Scalability	0.20	As assumed, the volume of data to be migrated has a consequent size that the tool has to cope with
Speed	0.10	Since the migration is a one-time process, the performance of the data conversion process is not the most important metric evaluated

Table 7.3: Weights for figures of merit

	Weight
Ease of use	
Product Architecture	0.20
Data Extraction	0.15
Data Transformation	0.15
Data Loading	0.15
Matching	0.15
Metadata Management	0.05
Development Environment	0.20
Flexibility	
Product Architecture	0.10
Data Support hline Data Extraction	0.25 0.15
Data Transformation	0.15
Data Loading	0.15
Matching	0.05
Metadata Management	0.05
Development Environment	0.10
Robustness	
Product Architecture	0.20
Data Support	0.05
Data Extraction	0.15
Data Transformation	0.15
Data Loading	0.15
Matching	0.10
Metadata Management	0.10
Development Environment	0.10

	Weight
Scalability	
Product Architecture	0.10
Data Extraction	0.20
Data Transformation	0.20
Data Loading	0.20
Matching	0.05
Metadata Management	0.20
Development Environment	0.05
Speed	
Product Architecture	0.05
Data Extraction	0.30
Data Transformation	0.30
Data Loading	0.30
Matching	0.05

Table 7.4: Weights for criteria

It is now possible to fill evaluation matrices by testing the operation of ETL tools on one or more similar testing scenario(s). Since there exist many different tools and testing all of them would go beyond the scope of this thesis, we will not detail their evaluation in this section. We selected five tools we consider fitting for a database migration towards NoSQL and Datomic that are described in the Technological Background chapter. Similarly, the operation of the extract-transform-load process differs from one tool to another and we will not detail it for all of them. We will illustrate Pentaho Data Integration's operation on our case study in the next section.

7.3.2 ETL tools and Datomic

The case of a migration towards Datomic rather than any other NoSQL technology is particular since Datomic is recent and thus few to no documentation of successful migrations can be found. The difficulty is that at the moment this thesis is written, no established ETL tool allows direct loading functionality towards Datomic.

Yet, that barrier does not mean that such a data conversion is not conceivable. Indeed, as we have discussed in chapter 2 Datomic is directly compatible with multiple NoSQL technologies such as Riak, Cassandra, Couchbase, DynamoDB or Infinispan. Moreover, ETL tools described in the same chapter provide data loading functionalities towards some of those technologies (depending on the tool employed). Being directly compatible means that Datomic can use those technologies as storage system and can be installed on a partition reserved for it and respecting its data model. Consequently, it is conceptually possible to set a Datomic database up on the top of one of those storage technologies and once that binding is realized to use an ETL tool to load data in that system. Though, this remains a conceptual assumption since we did not test it in this thesis, and studying the gap between Datomic and data storage technologies would be an interesting extended work.

7.4 Case study

The purpose of this section is to illustrate the data conversion method we described and thus the concrete operation of an ETL tool on our case study : the OSCAR system. More specifically, the case study we have been working on is a subset of OSCAR's database encompassing tables related to demographic and prescription information. Figures 6.16 and 6.19 depict the logical schemas of our case study before and after being converted to Datomic's data model.

We chose to illustrate the data conversion process for the content of the *demographic* table. Figure 7.5 details the source and logical schemas of this table's conversion.

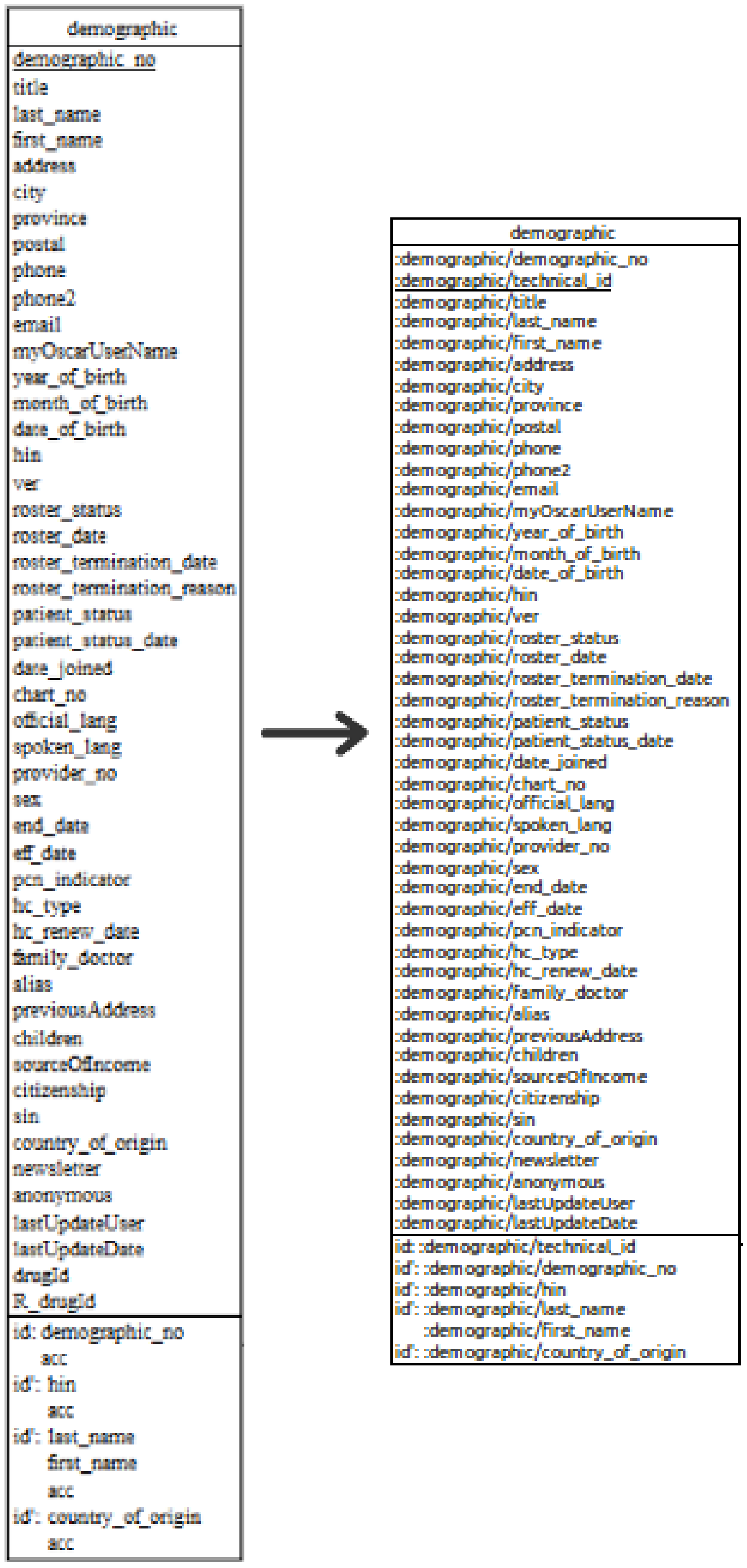


Figure 7.5: Source and target logical schemas of the *demographic* table from our case study

For illustration purpose, we chose arbitrarily the tool Pentaho Data Integration to perform the process given it is free, commonly used, well documented and supporting NoSQL technologies. As discussed in section 7.3.2, no established ETL tool provides a direct loading functionality towards Datomic, but such a process can be realized towards a directly compatible with Datomic data storage technology. Among available technologies, we chose to load the data within a Cassandra database, Cassandra being directly compatible with Datomic. Consequently, the first step of the conversion was to install Cassandra on an Ubuntu virtual machine, a process that we will not detail here. We also installed Pentaho Data Integration (PDI) on the same machine.

We then exported the source schema in a flat CSV file and manually populated it for 50 rows. This file is used as the source of the ETL process. Thus, that data (showed in the following figure) is no real data from OSCAR but artificial data used only for the example. Figure 7.6 illustrates the first 11 fields of the file and their related 40 first rows.

In Pentaho Data Integration, we created a new *transformation* and defined four steps for it as pictured in figure 7.7.

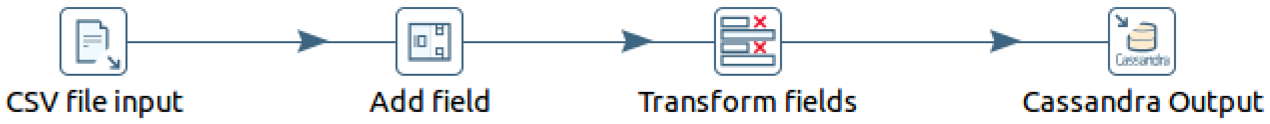
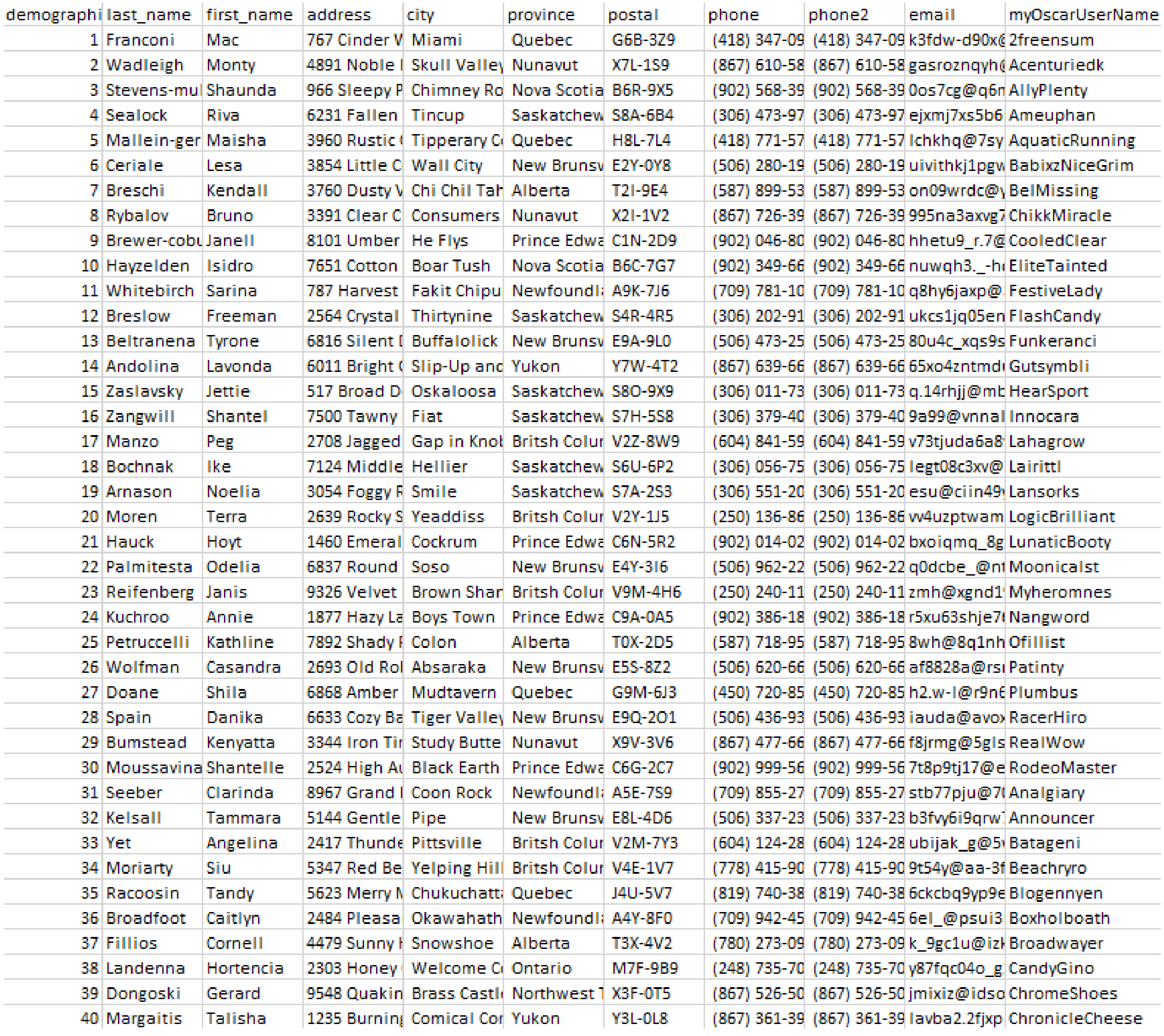


Figure 7.7: Extract-Transform-Load steps for our example

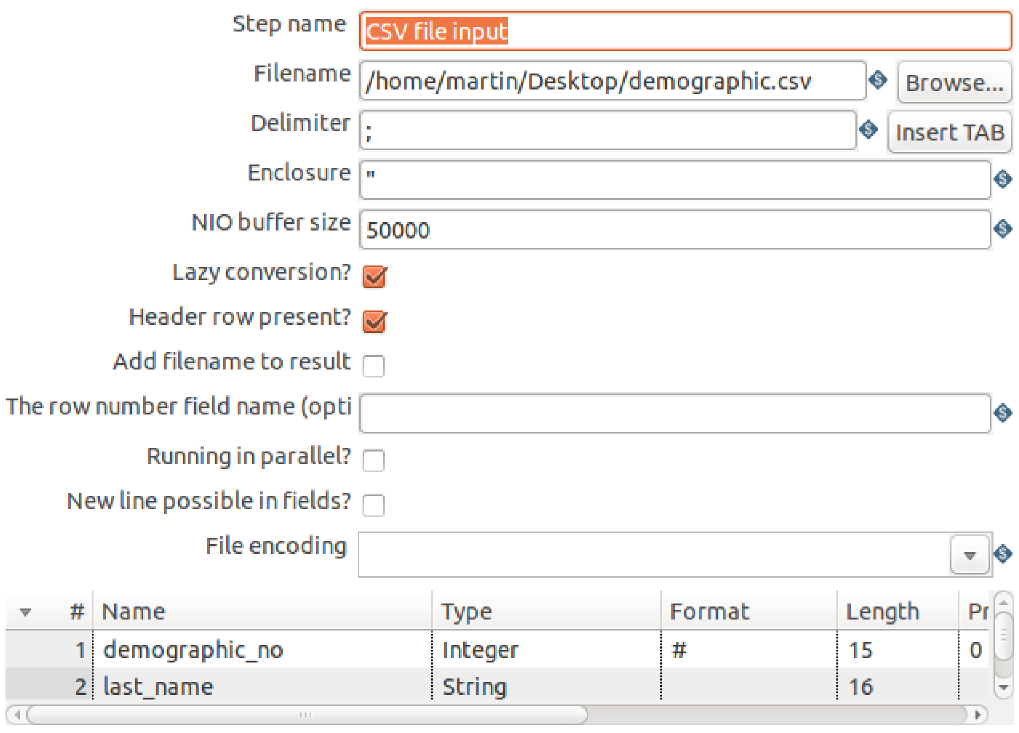


demographi	last_name	first_name	address	city	province	postal	phone	phone2	email	myOscarUserName
1	Franconi	Mac	767 Cinder V	Miami	Quebec	G6B-3Z9	(418) 347-09	(418) 347-09	k3fdw-d90x(	2freensum
2	Wadleigh	Monty	4891 Noble I	Skull Valley	Nunavut	X7L-1S9	(867) 610-58	(867) 610-58	gasroznqyh(	Acenturiedk
3	Stevens-mul	Shaunda	966 Sleepy P	Chimney Ro	Nova Scotia	B6R-9X5	(902) 568-39	(902) 568-39	0os7cg@q6n	AllyPlenty
4	Sealock	Riva	6231 Fallen	Tincup	Saskatchew	S8A-6B4	(306) 473-97	(306) 473-97	ejxmj7xs5b6	Ameuphan
5	Mallein-ger	Maisha	3960 Rustic (	Tipperary C	Quebec	H8L-7L4	(418) 771-57	(418) 771-57	lchkhq@7sy	AquaticRunning
6	Ceriale	Lesa	3854 Little C	Wall City	New Brunsv	E2Y-0Y8	(506) 280-19	(506) 280-19	uivithkj1pgw	BabixzNiceGrim
7	Breschi	Kendall	3760 Dusty V	Chi Chil Tah	Alberta	T2I-9E4	(587) 899-53	(587) 899-53	on09wrdc@y	BelMissing
8	Rybalov	Bruno	3391 Clear C	Consumers	Nunavut	X2I-1V2	(867) 726-39	(867) 726-39	995na3axvg7	ChikkMiracle
9	Brewer-cobu	Janell	8101 Umber	He Flys	Prince Edwa	C1N-2D9	(902) 046-80	(902) 046-80	hhetu9_r.7@	CooledClear
10	Hayzelden	Isidro	7651 Cotton	Boar Tush	Nova Scotia	B6C-7G7	(902) 349-66	(902) 349-66	nuwqh3._-h	EliteTainted
11	Whitebirch	Sarina	787 Harvest	Fakit Chipu	Newfoundl:	A9K-7J6	(709) 781-10	(709) 781-10	q8hy6jaxp@	FestiveLady
12	Breslow	Freeman	2564 Crystal	Thirtynine	Saskatchew	S4R-4R5	(306) 202-91	(306) 202-91	ukcs1jq05en	FlashCandy
13	Beltranena	Tyrone	6816 Silent [	Buffalolick	New Brunsv	E9A-9L0	(506) 473-25	(506) 473-25	80u4c_xqs9s	Funkeranci
14	Andolina	Lavonda	6011 Bright (	Slip-Up anc	Yukon	Y7W-4T2	(867) 639-66	(867) 639-66	65xo4zntmd	Gutsymbli
15	Zaslavsky	Jettie	517 Broad D	Oskaloosa	Saskatchew	S8O-9X9	(306) 011-73	(306) 011-73	q.14rhjj@mb	HearSport
16	Zangwill	Shantel	7500 Tawny	Fiat	Saskatchew	S7H-5S8	(306) 379-40	(306) 379-40	9a99@vnnal	Innocara
17	Manzo	Peg	2708 Jagged	Gap in Knot	Britsh Colur	V2Z-8W9	(604) 841-59	(604) 841-59	v73tjuda6a8	Lahagrow
18	Bochnak	Ike	7124 Middle	Hellier	Saskatchew	S6U-6P2	(306) 056-75	(306) 056-75	legt08c3xv@	Lairittl
19	Arnason	Noelia	3054 Foggy R	Smile	Saskatchew	S7A-2S3	(306) 551-20	(306) 551-20	esu@ciin49	Lansorks
20	Moren	Terra	2639 Rocky S	Yeaddiss	Britsh Colur	V2Y-1J5	(250) 136-86	(250) 136-86	vv4uzptwam	LogicBrilliant
21	Hauck	Hoyt	1460 Emeral	Cockrum	Prince Edwa	C6N-5R2	(902) 014-02	(902) 014-02	bxoiqmq_8g	LunaticBooty
22	Palmitesta	Odelia	6837 Round	Soso	New Brunsv	E4Y-3I6	(506) 962-22	(506) 962-22	q0dcbe_@nt	Moonicalst
23	Reifenberg	Janis	9326 Velvet	Brown Shar	Britsh Colur	V9M-4H6	(250) 240-11	(250) 240-11	zmh@xgnd1	Myheromnes
24	Kuchroo	Annie	1877 Hazy La	Boys Town	Prince Edwa	C9A-0A5	(902) 386-18	(902) 386-18	r5xu63shje7	Nangword
25	Petruccelli	Kathline	7892 Shady F	Colon	Alberta	T0X-2D5	(587) 718-95	(587) 718-95	8wh@8q1nh	Ofillist
26	Wolfman	Casandra	2693 Old Rol	Absaraka	New Brunsv	E5S-8Z2	(506) 620-66	(506) 620-66	af8828a@rsı	Patinty
27	Doane	Shila	6868 Amber	Mudtavern	Quebec	G9M-6J3	(450) 720-85	(450) 720-85	h2.w-l@r9n6	Plumbus
28	Spain	Danika	6633 Cozy Ba	Tiger Valley	New Brunsv	E9Q-2O1	(506) 436-93	(506) 436-93	iauda@avox	RacerHiro
29	Bumstead	Kenyatta	3344 Iron Tir	Study Butte	Nunavut	X9V-3V6	(867) 477-66	(867) 477-66	f8jrmg@5gls	RealWow
30	Moussavina	Shantelle	2524 High Au	Black Earth	Prince Edwa	C6G-2C7	(902) 999-56	(902) 999-56	7t8p9tj17@e	RodeoMaster
31	Seeber	Clarinda	8967 Grand I	Coon Rock	Newfoundl:	A5E-7S9	(709) 855-27	(709) 855-27	stb77pju@7	Analgiary
32	Kelsall	Tammara	5144 Gentle	Pipe	New Brunsv	E8L-4D6	(506) 337-23	(506) 337-23	b3fvy6i9qrw	Announcer
33	Yet	Angelina	2417 Thunde	Pittsville	Britsh Colur	V2M-7Y3	(604) 124-28	(604) 124-28	ubijak_g@5	Batageni
34	Moriarty	Siu	5347 Red Be	Yelping Hill	Britsh Colur	V4E-1V7	(778) 415-90	(778) 415-90	9t54y@aa-3f	Beachryro
35	Racoosin	Tandy	5623 Merry M	Chukuchatt:	Quebec	J4U-5V7	(819) 740-38	(819) 740-38	6ckcbq9yp9e	Blogennyen
36	Broadfoot	Caitlyn	2484 Pleasa	Okawahath	Newfoundl:	A4Y-8F0	(709) 942-45	(709) 942-45	6el_@psui3	Boxholboath
37	Fillios	Cornell	4479 Sunny I	Snowshoe	Alberta	T3X-4V2	(780) 273-09	(780) 273-09	k_9gc1u@izk	Broadwayer
38	Landenna	Hortencia	2303 Honey	Welcome C	Ontario	M7F-9B9	(248) 735-70	(248) 735-70	y87fqc04o_g	CandyGino
39	Dongoski	Gerard	9548 Quakin	Brass Castl	Northwest T	X3F-0T5	(867) 526-50	(867) 526-50	jmixiz@idso	ChromeShoes
40	Margaitis	Talisha	1235 Burnin	Comical Cor	Yukon	Y3L-0L8	(867) 361-39	(867) 361-39	lavba2.2fjxp	ChronicleCheese

Figure 7.6: Excerpt of the input flat CSV file

7.4.1 CSV file input

This step consists of selecting the CSV file that will be the input of the *transformation*, along with determining which symbols are delimiters of fields. It is then possible to scan the file to get the fields that will compose the table and precise their characteristics. The interface used for this step is depicted in figure 7.8 and its output can be previewed in figure 7.9.



Step name	CSV file input
Filename	/home/martin/Desktop/demographic.csv
Delimiter	;
Enclosure	"
NIO buffer size	50000
Lazy conversion?	✓
Header row present?	✓
Add filename to result	
The row number field name (opti	
Running in parallel?	
New line possible in fields?	
File encoding	

#	Name	Type	Format	Length	Pr
1	demographic_no	Integer	#	15	0
2	last_name	String		16	

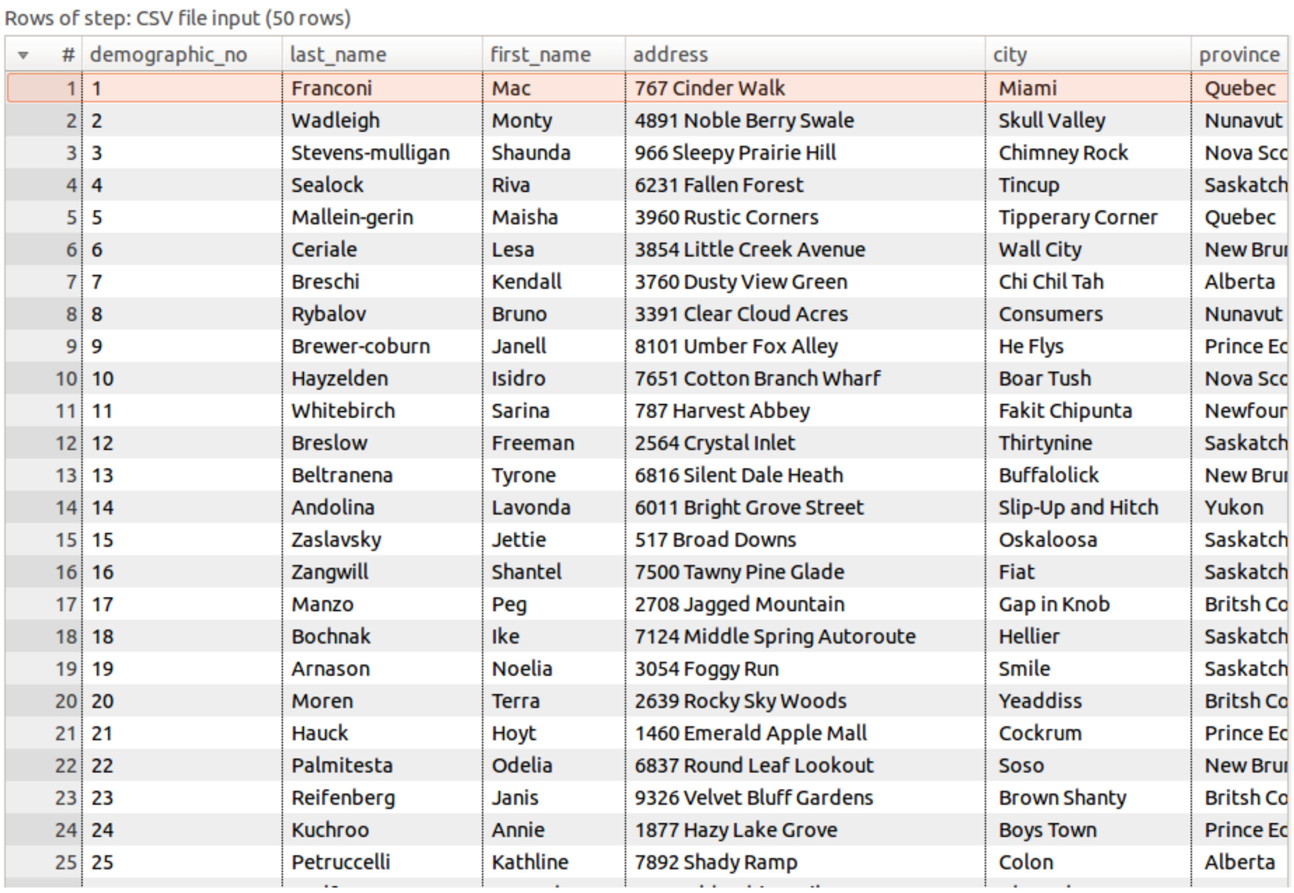
Figure 7.8: *CSV file input* transformation interface in PDI

7.4.2 Add field

This step consists of inserting the field *:demographic/technical_id* using the *Add sequence* transformation of PDI. This transformation allows to add an auto-incremented field to the source data. Figure 7.10 shows the interface used to add such a field.

7.4.3 Transform fields

This step consists of modifying the fields' names in order to correspond to the target schema. The fields' types cannot yet be redefined to match Datomic's types since those types are not directly compatible with standard Cassandra types. This is realized using the *Select values* transformation in PDI whose interface is pictured in figure 7.11.


Rows of step: CSV file input (50 rows)

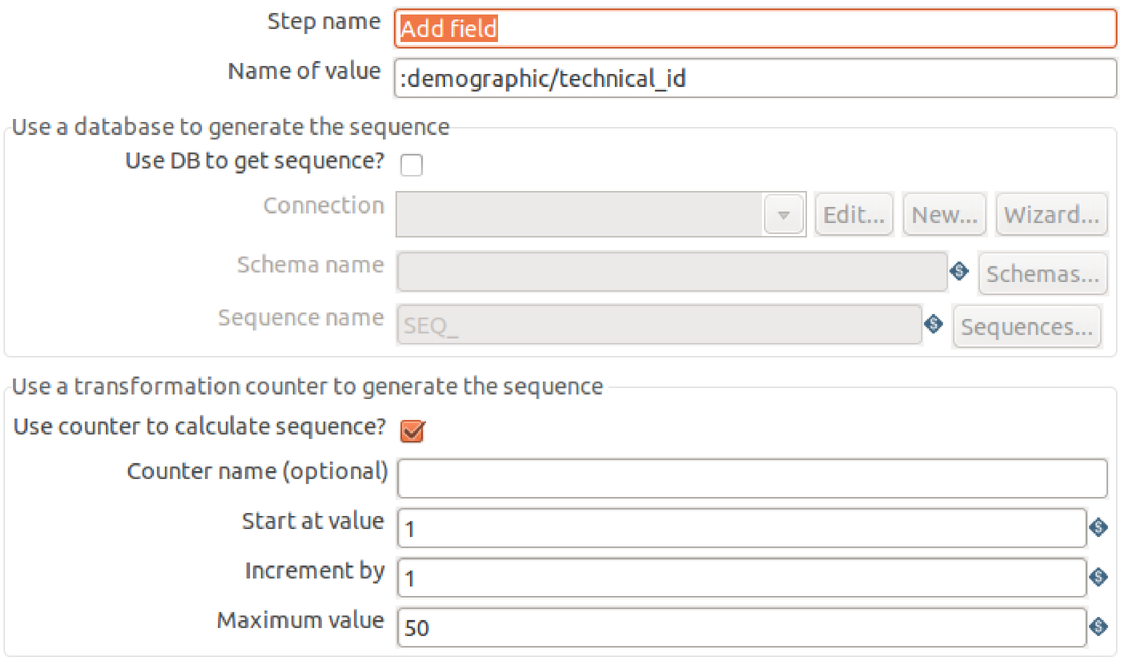
#	demographic_no	last_name	first_name	address	city	province
1	1	Franconi	Mac	767 Cinder Walk	Miami	Quebec
2	2	Wadleigh	Monty	4891 Noble Berry Swale	Skull Valley	Nunavut
3	3	Stevens-mulligan	Shaunda	966 Sleepy Prairie Hill	Chimney Rock	Nova Scc
4	4	Sealock	Riva	6231 Fallen Forest	Tincup	Saskatch
5	5	Mallein-gerin	Maisha	3960 Rustic Corners	Tipperary Corner	Quebec
6	6	Ceriale	Lesa	3854 Little Creek Avenue	Wall City	New Bru
7	7	Breschi	Kendall	3760 Dusty View Green	Chi Chil Tah	Alberta
8	8	Rybalov	Bruno	3391 Clear Cloud Acres	Consumers	Nunavut
9	9	Brewer-coburn	Janell	8101 Umber Fox Alley	He Flys	Prince Ec
10	10	Hayzelden	Isidro	7651 Cotton Branch Wharf	Boar Tush	Nova Scc
11	11	Whitebirch	Sarina	787 Harvest Abbey	Fakit Chipunta	Newfour
12	12	Breslow	Freeman	2564 Crystal Inlet	Thirtynine	Saskatch
13	13	Beltranena	Tyrone	6816 Silent Dale Heath	Buffalolick	New Bru
14	14	Andolina	Lavonda	6011 Bright Grove Street	Slip-Up and Hitch	Yukon
15	15	Zaslavsky	Jettie	517 Broad Downs	Oskaloosa	Saskatch
16	16	Zangwill	Shantel	7500 Tawny Pine Glade	Fiat	Saskatch
17	17	Manzo	Peg	2708 Jagged Mountain	Gap in Knob	Britsh Co
18	18	Bochnak	Ike	7124 Middle Spring Autoroute	Hellier	Saskatch
19	19	Arnason	Noelia	3054 Foggy Run	Smile	Saskatch
20	20	Moren	Terra	2639 Rocky Sky Woods	Yeaddiss	Britsh Co
21	21	Hauck	Hoyt	1460 Emerald Apple Mall	Cockrum	Prince Ec
22	22	Palmitesta	Odelia	6837 Round Leaf Lookout	Soso	New Bru
23	23	Reifenberg	Janis	9326 Velvet Bluff Gardens	Brown Shanty	Britsh Co
24	24	Kuchroo	Annie	1877 Hazy Lake Grove	Boys Town	Prince Ec
25	25	Petruccelli	Kathline	7892 Shady Ramp	Colon	Alberta

Figure 7.9: Excerpt of the preview of the data extracted from the CSV file

7.4.4 Cassandra Output

This last step consists of creating the bond between the transformation and an existing Cassandra database. Thus, the Cassandra *host*, *port*, *keyspace*, *username* and *password* are specified along with the *column family* (or *table*) that will be the target of the transformation and the *field used as a key*. Figure 7.12 illustrates the interface used for this step.

We have thereby pictured the concrete operation of an ETL tool to convert relational data to a NoSQL database. More specifically, we migrated data corresponding to the demographic table of our case study towards Cassandra in respect to Datomic's data model.



Step name	Add field
Name of value	:demographic/technical_id

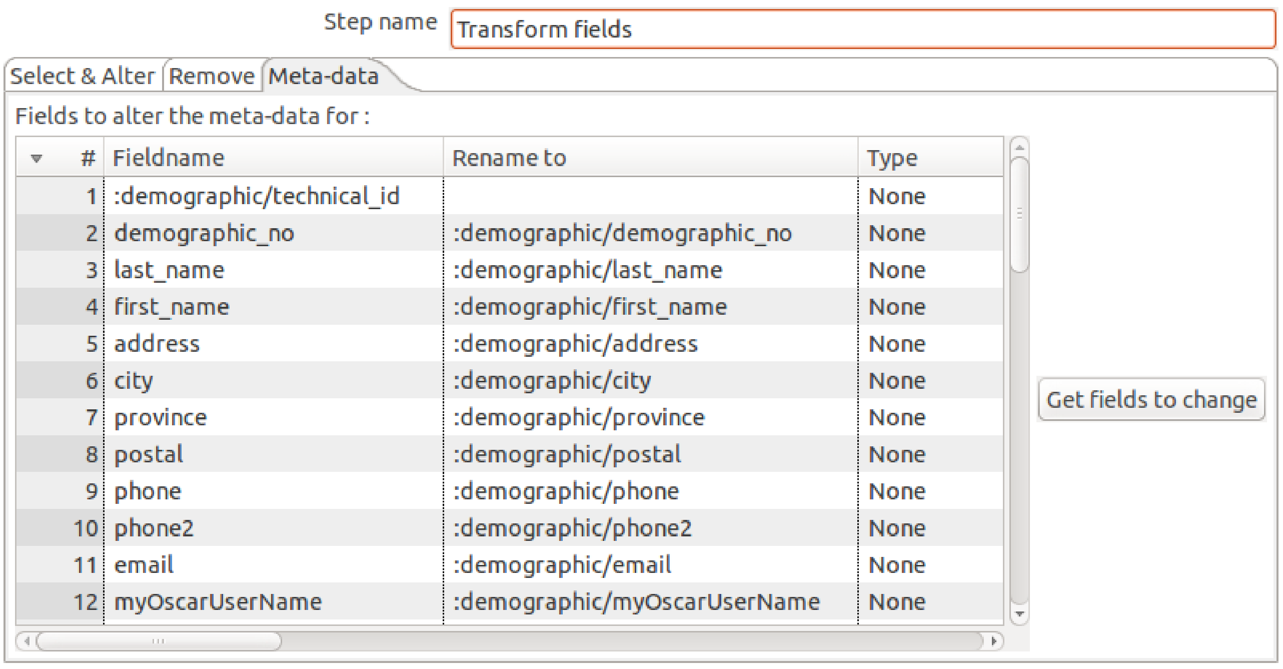
Use a database to generate the sequence

Use DB to get sequence?	☐
Connection	
Schema name	
Sequence name	SEQ_

Use a transformation counter to generate the sequence

Use counter to calculate sequence?	✓
Counter name (optional)	
Start at value	1
Increment by	1
Maximum value	50

Figure 7.10: *Add sequence* transformation interface in PDI



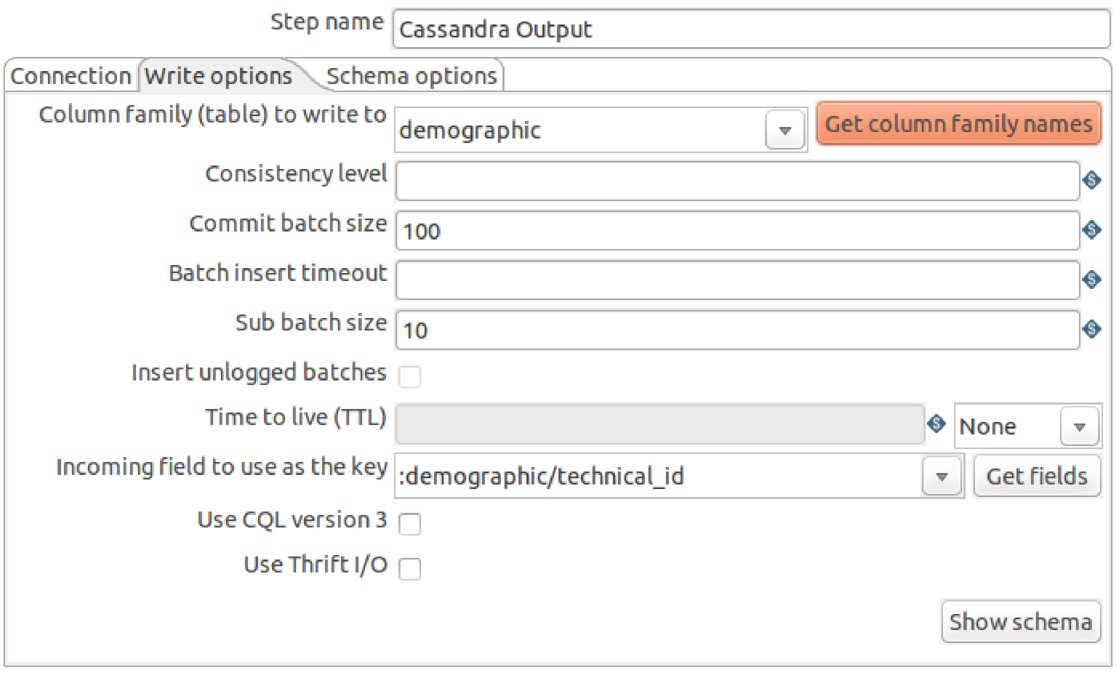
Step name: Transform fields

Select & Alter | Remove | Meta-data

Fields to alter the meta-data for :

#	Fieldname	Rename to	Type
1	:demographic/technical_id		None
2	demographic_no	:demographic/demographic_no	None
3	last_name	:demographic/last_name	None
4	first_name	:demographic/first_name	None
5	address	:demographic/address	None
6	city	:demographic/city	None
7	province	:demographic/province	None
8	postal	:demographic/postal	None
9	phone	:demographic/phone	None
10	phone2	:demographic/phone2	None
11	email	:demographic/email	None
12	myOscarUserName	:demographic/myOscarUserName	None

Figure 7.11: *Select values* transformation interface in PDI



Step name	Cassandra Output

Column family (table) to write to	demographic
Consistency level	
Commit batch size	100
Batch insert timeout	
Sub batch size	10
Insert unlogged batches	☐
Time to live (TTL)	None
Incoming field to use as the key	:demographic/technical_id
Use CQL version 3	☐
Use Thrift I/O	☐

Figure 7.12: *Cassandra Output* transformation interface in PDI

Chapter 8

Program conversion

8.1 Overview

At this point, we dispose of a target database that is filled with the transformed data extracted from the source database. Still, it is not yet possible for the application program(s) to execute queries and updates on this database since statements are written in the source DMS' logic and corresponding language. Figure 8.1 illustrates the situation of the migration before performing the program conversion process:

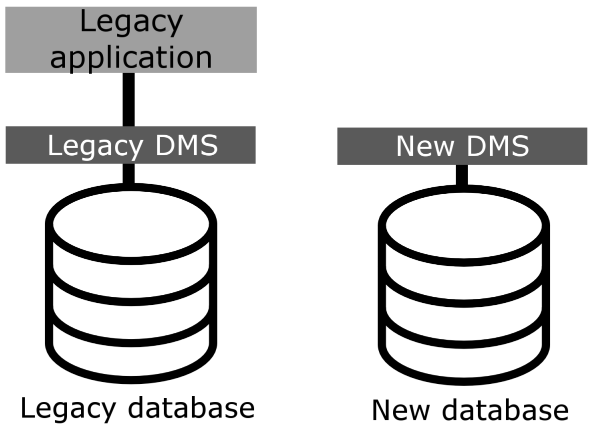


Figure 8.1: Situation pre-program conversion

Thus, we need to make the applicative layer evolve to create the bond with the new database, thus completing the whole database migration process. As expressed in the state of the art, three strategies exist when it comes to converting the application program [Hainaut et al., 2008]:

1. **Wrapper strategy**
2. **Statements rewriting strategy**
3. **Logic rewriting strategy**

However, selecting a particular strategy and accordingly transform the application program(s) in order to create a communication link with the target database might not be the only aspect to take into account when planning and performing a database migration. Thus, it would be interesting to go further and to discuss the relationship between legacy and target databases and application programs. In this chapter, we will establish a global program conversion method for a database migration towards NoSQL/Datomic and relying on the most suitable strategy(ies). The following sections will detail different treatment approaches of both source and target database and develop their strengths and weaknesses, in order to suggest a general method combining their benefits. During this analysis, we will assume there is only one application program by abuse of language. Yet, the concepts remain the same if there are multiple programs.

8.2 Maintain both databases

A first approach of program conversion is to avoid getting rid of the legacy database and make a use of it instead. Indeed, a database migration process might not aim at replacing the legacy database by the new one, but to maintain both databases. For instance, this possibility could be interesting in order to proceed to a comparison of attributes such as performance or security. To achieve maintaining two databases while manipulating a single applicative program, the three strategies have to be adapted.

8.2.1 Wrapper strategy

Using the wrapper strategy would allow performing changes while having very little impact on the application program. Indeed, the DMS calls within the application program along with related elements such as variables have to be replaced by wrapper calls. A wrapper is implemented for each data record type manipulated. Within a wrapper, legacy statements to the legacy DMS are not only re-stated, but are also followed by their translation in the target DMS language. This way, each call from the application program queries or updates both databases. In the specific case of queries, it is also the role of the wrapper to cross the results and select which one of them will be sent to the program since it expects one result only. Furthermore, employing an external element as an intermediate within a system would necessarily increase its complexity and thus make future changes more difficult to perform. Figure 8.2 illustrates the operation of the wrapper program conversion strategy when maintaining both legacy and target databases:

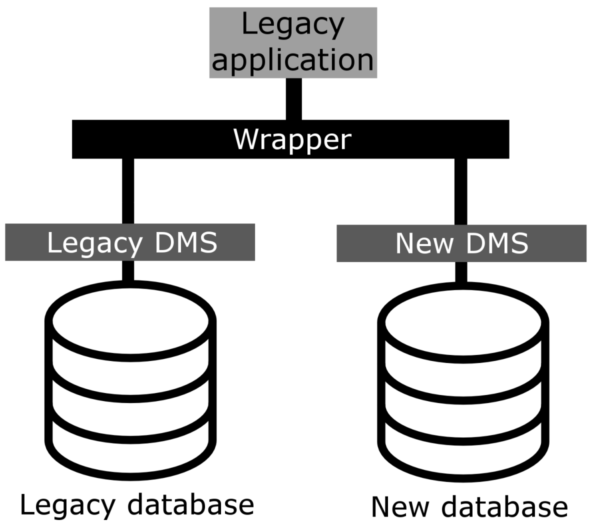


Figure 8.2: Wrapper strategy applied when maintaining both source and target databases

8.2.2 Statements rewriting strategy

Using the statements rewriting strategy would have a slightly stronger impact on the application program than using the wrapper strategy even though it would be moderate. Indeed, if the application program is more directly modified than by using wrappers, the changes remain concentrated to certain key areas within the source code that are the DMS calls. Those calls are kept as they are in order to keep accessing the legacy database, but are also augmented by target DMS statements. This way, each call from the application program queries or updates both databases. In the specific case of queries, the application program has to be enhanced to cross the results and select which one of them will be taken into account. Figure 8.3 illustrates the operation of the statements rewriting program conversion strategy when maintaining both legacy and target databases:

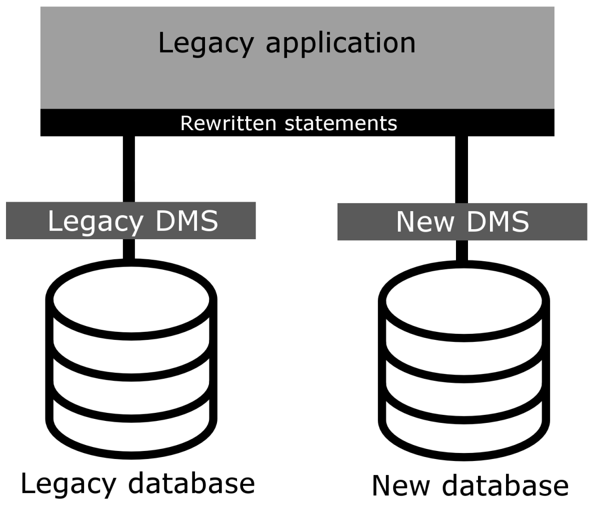


Figure 8.3: Statements rewriting strategy applied when maintaining both source and target databases

8.2.3 Logic rewriting strategy

Using the logic rewriting strategy would have a strong impact and be more complex in the optic of maintaining both source and target databases. The logic rewriting strategy aims at deeply modifying the source code in order to make the best out of the target DMS' language and data model. However, deep modifications to the program's logic would prevent it to communicate with the legacy database, making it impossible to query and update both source and target databases without adding one of the two other strategies on the top on this one. Indeed, using wrappers or add legacy DMS statements within the database calls in the new program logic would be necessary to maintain both databases. Figure 8.4 illustrates the operation of the statements rewriting program conversion strategy when maintaining both legacy and target databases:

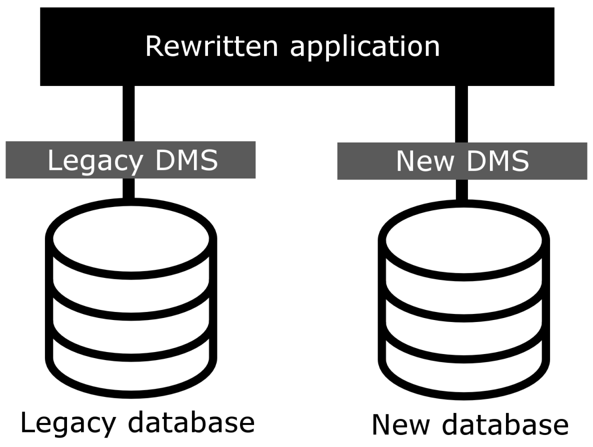


Figure 8.4: Logic rewriting strategy applied when maintaining both source and target databases

8.2.4 Global analysis

Globally, the concept of maintaining both source and target databases operating presents the advantages to ensure a high security of data, to offer the possibility to compare and test both databases at runtime for each query or update and to allow the use of whichever database is the most convenient in a given context. However, this approach is costly and strongly affects performance since each query or update commanded by the application program is performed twice. Yet, this concept might be viewed as a first program transformation phase used to test the new database' usage while securing the correct operation of the application program by maintaining the legacy database.

Concerning the strategy the employ to support this concept, we summarize the characteristics of the three strategies in the following way:

1. **Wrapper strategy:** Low impact on application program, emphasis on the source DMS' operation, conceived for short-term usage, easiest way to implement program conversion for two databases
2. **Statements rewriting strategy:** Moderate impact on application program, emphasis on the source DMS' operation, conceived for longer-term usage for similar types of DMS
3. **Logic rewriting:** Strong impact and cost, emphasis on the target DMS' operation, conceived for long-term usage for different types of DMS, difficult to implement while maintaining both databases

8.3 Maintain target database only

A second approach of program conversion is to use only the target database and backup or dispose of the existing one. Indeed, proceeding this way is a more traditional method to perform a database migration, aiming for example at benefiting only from the features of the target database. To achieve this type of program conversion, the three strategies can be considered.

8.3.1 Wrapper strategy

Using the wrapper strategy would allow performing changes while having very little impact to the application program. Indeed, the DMS calls within the application program along with related elements such as variables have to be replaced by wrapper calls. A wrapper is implemented for each data record type manipulated. Within a wrapper, legacy statements to the legacy DMS are translated in the target DMS language so that by calling the wrapper, the application program can query or update the target database. Furthermore, employing an external element as an intermediate within a system would necessarily increase its complexity and thus make future changes more difficult to perform. Figure 8.5 illustrates the operation of the wrapper program conversion strategy when maintaining the target database only:

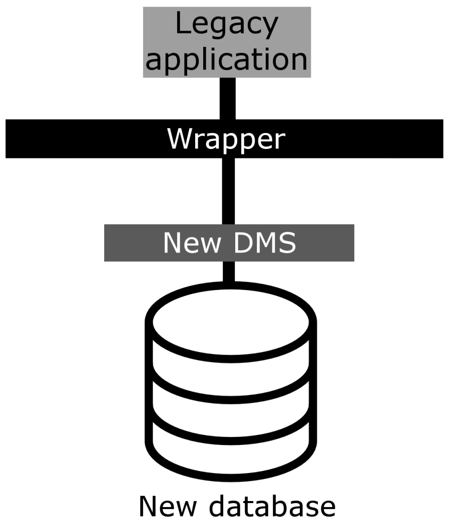


Figure 8.5: Wrapper strategy applied when maintaining the target database only

8.3.2 Statements rewriting strategy

Using the statements rewriting strategy would have a slightly stronger impact on the application program than using the wrapper strategy even though it would be moderate.

Indeed, if the application program is more directly modified than by using wrappers, the changes remain concentrated to certain key areas within the source code that are the DMS calls. Those calls are replaced by target DMS statements. This way, each call from the application program now directly queries or updates the target database. Figure 8.6 illustrates the operation of the statements rewriting program conversion strategy when maintaining the target database only:

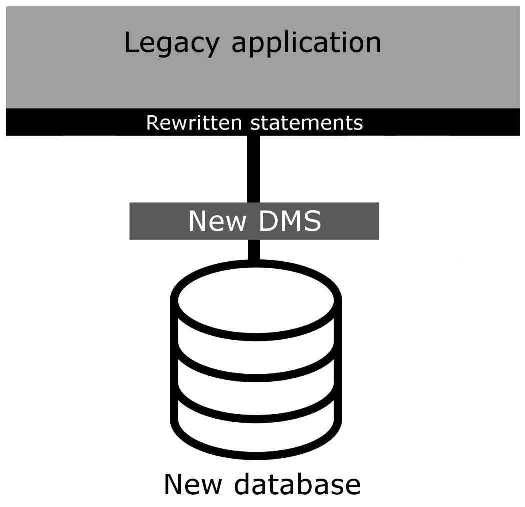


Figure 8.6: Statements rewriting strategy applied when maintaining the target database only

8.3.3 Logic rewriting strategy

Using the logic rewriting strategy would have a strong impact on the application program. Indeed, the logic rewriting strategy aims at deeply modifying the source code in order to make the best out of the target DMS' language and data model. If this strategy is the most expensive and the slowest to implement, it is also the only one that benefits of the full potential of the target DMS. Figure 8.7 illustrates the operation of the statements rewriting program conversion strategy when maintaining the target database only:

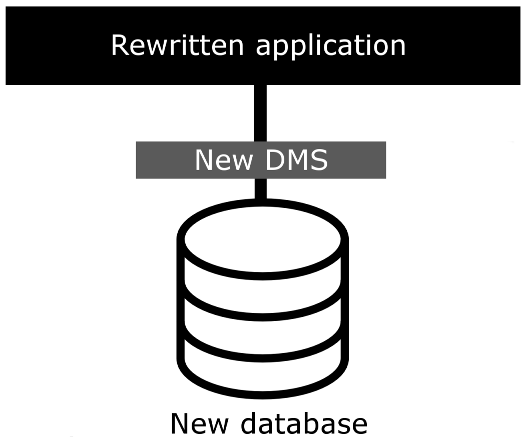


Figure 8.7: Logic rewriting strategy applied when maintaining the target database only

8.3.4 Global analysis

Globally, the concept of maintaining the target database operating only represents the traditional view of database migration that can be justified by all features the new system presents and that the legacy system lacked. It generally leads to a gain of performance. Yet, this approach simply cuts all links with the existing database that will not be updated nor queried anymore. Relying only on the target database means that the full operation of the migrated system must be ensured beforehand. As we will see in the next section, in a migration performed in ideal conditions, we prefer this approach as a second step following the maintenance of both databases for a certain period of time.

Concerning the strategy the employ to support this concept, the characteristics of the three strategies can be summarized in a way close to the one we detailed for maintaining both databases:

1. **Wrapper strategy:** Low impact on application program, emphasis on the source DMS' operation, conceived for short-term usage
2. **Statements rewriting strategy:** Moderate impact on application program, emphasis on the source DMS' operation, conceived for longer-term usage for similar types of DMS
3. **Logic rewriting:** Strong impact and cost, emphasis on the target DMS' operation, conceived for long-term usage for different types of DMS

8.4 Advocated method

In this section, we will rely on the conclusions of the preceding section to discuss of a program conversion method in the particular context of database migration towards NoSQL, specifically towards Datomic.

8.4.1 General method

We are aiming in this thesis at the migration of a relational database towards a NoSQL database (Datomic in our case study). NoSQL and NewSQL DMS in general rely on very different principles than traditional relational DMS (cf. state of the art) and Datomic is no exception. Thus, the logic on which underpins the legacy application program has to evolve during the migration process. This change, combined to the fact that Datomic databases are relatively recent on the market and so lack of a consequent corpus of experiences, tutorials and frequently asked questions, tends to lead to uncertainty about the operation of the target system. Consequently, if time and budget allow it, we suggest a two-phases approach of program transformation ensuring both a security of the data and a possibility to test the new system while keeping maintaining the legacy database.

The first phase of the approach is to maintain both databases operating for a short period of time. Since it is used for a trial time-lapse, the most fitting strategy we suggest employing is the *wrapper strategy*, which is the cheapest, quickest to implement, with the lowest impact on the application program and conceived for a short-time use. Indeed, a logic rewriting strategy would necessitate a lot of effort and consequently would not be interesting in the context of a short-term trial period. In the same way, a statements rewriting strategy could be conceivable but since the source and target DMS are not similar, the incurred costs and impact on the source code would not be worth this usage. De plus, puisque la conversion de schéma est lossless comme expliqué dans le chapitre 6, si la nouvelle base de données est read-only, il est alors pas possible de corrompre les contraintes. Ainsi, cette stratégie permet d'éviter l'implémentation des contraintes Datomic au niveau applicatif.

The second phase of the approach is, after a period of testing and habituation to the operation of the system with the target database, to cut the link with the legacy database and maintain only the target database. Here again, the statements rewriting strategy is compromised by the differences there are between both source and target DMS. Thus, we suggest the *logic rewriting strategy* which is the one that will benefit the most of the NoSQL DMS capabilities and aims at the longest-term usage. Since rewriting a program's logic is a costly and possibly long process, it can be engaged on a copy of the source code while queries and updates are reverberated on both still maintained databases during the first phase. Furthermore, cutting the link with the legacy database is facilitated by the use of the wrapper strategy in the previous phase.

We summarize the global suggested program conversion method in the following way:

1. **Maintain both databases** using the *wrapper strategy*
2. **Maintain target database only** using the *logic rewriting strategy*

We will now detail the concrete implementation of those two phases.

8.4.2 Building wrappers

Wrappers implementation and usage relies on the following concepts:

- *The wrapper schema* designates the schema used by the application program calling the wrapper (the relational schema in this case)
- *The database schema* designates the schema of the database the wrapper communicates with (both relational and Datomic schema in this case since both databases are maintained)
- *The schema mapping* designates the GER mapping as defined in the state of the art (the mapping between the relational and Datomic schemas in this case)

Wrappers typically treat queries and updates following three main steps [Thiran et al., 2005]:

1. **Language mappings:** Analyze the query/update and extract its semantics
2. **Inter-schema mappings:** Derive those semantics according to the schema transformation
3. **Language mappings and optimization:** Translate the result in the target DML language

Figure 8.8 illustrates the conversion process of a source query in a query understandable by the target database:

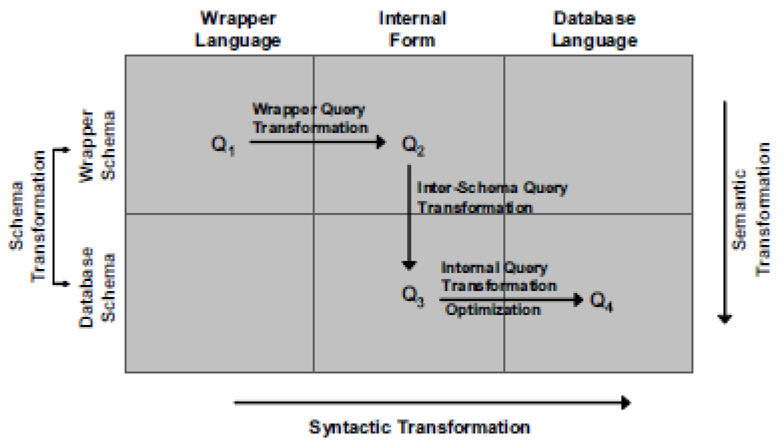


Figure 8.8: Language and schema mappings of a wrapper query Q1 into a DML query Q4 [Thiran et al., 2005]

In this thesis' case, since we aim at maintaining both databases, the legacy DML statement is systematically added to the translated query/update in order to reverberate all changes to both databases.

While schemas and schemas transformation are defined using the GER model, queries/updates are also expressed in an internal query language that represents each query/update's semantics and abstracts the DMS languages [Thiran et al., 2005]. Figure 8.9 illustrates a read/write (query/update) wrapper's generic architecture [Thiran et al., 2006]:

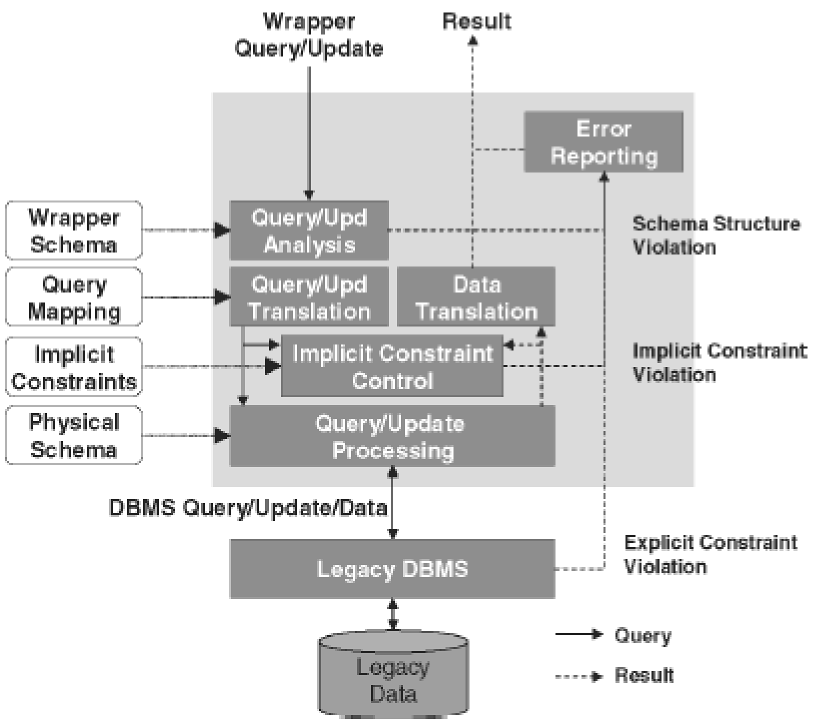


Figure 8.9: R/W wrapper architecture [Thiran et al., 2006]

In this case, this generic architecture has to be refined to take into account two databases instead of one. To do so, further than systematically coupling a translated query/update with its source DML statement, a *result selection* module also has to be implemented in the case of queries to select which result will be taken into account and sent as a result to the legacy application program.

Furthermore, in the case of a migration towards Datomic (rather than towards another NoSQL technology), a strategy to avoid implementing Datomic constraints seen in chapter 6 at the applicative level necessitates to adapt the method described above. Indeed, since the *schema conversion* is lossless, if the target database is read-only and updated via another mechanism (such as a *data conversion process*), it it not possible to corrupt constraints. Thus, the wrappers do not need to translate updates but only queries so they only keep the source database up to date and can query both databases. By doing so, the testing character of this first step of *program transformation* takes all its sense since the impact on the applicative layer is limited.

Thus, we covered the typical internal operation of a wrapper and how to decline it to fit this thesis' case. We will not go further in the details of the implementation of a wrapper since it goes beyond the scope of this work, but many references are available to illustrate it, notably [Thiran et al., 2005], [Thiran et al., 2006] and [Cleve et Hainaut, 2006].

8.4.3 Rewriting the program's logic

Rewriting the application program's logic consists of adapting an existing source code to the concepts and principles that lye behind a DMS. The effort necessary to review the logic depends on the gap between the DMS logics. The logic behind a Cobol database is different than the logic behind a relational database and thus requires lots of changes, and the logic behind two relational databases running on different DMS is relatively similar, thus minimizing the logic rewriting work.

Yet, unlike implementing wrappers or rewriting statements, rewriting the program's logic can modify and extend the system's functionalities and can be performed in many ways according to the expectations of the developers. The process strongly depends on the context, the purpose of the system and on the motivations behind the migration. Consequently, it is complex to educe a general approach or a set of good logic rewriting principles. The two following paragraphs cover examples of logic differences in the case of a migration from a relational database towards a Datomic database.

In the case of Datomic, most of the effort will consist of working on the constraints and globally the difference of rigidity between the relational database and the Datomic database. Indeed, if the databases schemas are comparable in some ways, the flexibility level is very different and will lead to divergent ways to access data. Attributes in the relational world belong to an entity type and thus are part of a relationship and incur constraints such as having a default value, being a primary key or else. In Datomic, attributes are just names that can be found in any entity and that incur no explicit constraints. This changes the way data is accessed. In the case of a migration towards Datomic, the constraints must be taken into account at the applicative level if needed.

Another logic difference between relational databases and Datomic databases is the immutability inherent to Datomic. Indeed, the update does not exist in Datomic and is replaced by a Datom creation that will take the place of the previous one, while maintaining it accessible. This is different than in the relational world where data can be updated, and once the old state is, it cannot be accessed anymore. Thus, this opens doors for an adapted application program to extend its functionalities and benefit from those new possibilities.

The logic rewriting process' impact on the application program also depends on the localisation of data accesses in the source code and the use of this data. The use of centralized data accesses in key areas such as Data Access Objects would ease the identification of portions of code to modify. Similarly, results of queries used in local methods with limited repercussions would be easier to treat than results split and interpreted throughout the whole program. So, it is important to have a deep comprehension of the application program in order to be able to apprehend the implications of database accesses on the whole system's operation.

8.5 Case study

We will not illustrate the program conversion process on our case study due to the complexity and width of the OSCAR system, making such an illustration worth a standalone work. Indeed, if the data itself is centralized within database tables and can be isolated, the call sites of a particular set of data is spread throughout the code. Yet, we will discuss of specificities and difficulties inherent to the context our this case study.

The principal aspect specific to the OSCAR system to consider when converting the application program would be its diversity. Indeed, OSCAR is developed and updated in parallel by a lot of developers, each with his own coding standards and objectives. This results in a heterogeneous system mixing multiple technologies and lacking a standard complete documentation. Consequently, the program conversion process, whether performed by wrapper or by logic rewriting has to take into account this constraint. In terms of wrappers, it will induce a multiplication of the number of wrappers to cope with the different technologies along with the different record types. In terms of logic rewriting, it will induce the necessity of adapting to multiple existing logics rather than one in the generic case.

The wide range of authors, technologies and users, coupled with the consistent content of the database, leads to a large system whose logic rewriting process will not be trivial. This tends to support the method described in this thesis consisting of combining both wrapper and logic rewriting approaches. Indeed, diversity and width of the system inherited from its open source nature both increase risks of errors and dysfunctions, making it safer to test the migration via wrappers while progressively rewriting the program's logic. They also provide the system with many potential testers.

Chapter 9

Performance

9.1 Overview

A general and precise definition of the performance concept is pretty complicated to define due to the fact that it covers several different areas. Thus, this concept can be applied to art, entertainment (such as music, film, theater, etc.), sport (from physical sport to cars), business, computer sciences and many more. Nevertheless, the notion of performance, and particularly how to measure it, is very important and recognized by many people from functional disciplines.

The etymology of the word “performance” shows that “performance” is composed of the verb “to perform” and the suffix “-ance” from the old French, meaning “Achievement”. For centuries, the meaning of this word has evolved. Thus, around 1590, the meaning of “performance” as “a thing performed” is appeared but around 1610, this meaning changed to become “the action of performing a play, etc.”. While the performance as “a public entertainment” appeared around 1709 and the art of the performance only emerged in 1971. The different meanings is explained by the fact that the concept of performance covers many areas [Dictionary, 2015].

Focus on the notion of performance in the computer world to provide a definition of the word “performance“ in database area. In general terms, the performance of a computer system can be defined in two ways [WhatIs, 2015b]:

- As being a set of amounted indicators for measuring the elapsed time for a computer system to complete given tasks taking into account time and given resources.
- As being the computer speed executing a certain number of millions of instructions per second (MIPS) throughout the benchmark test, which will explain in the next section.

The performance of a computer system uses several concepts such as response time, data volume, throughput, availability, resources available, etc.

These first two concepts are key factors of database performance. When the data volume increases, the response time increases too. Therefore, these concepts are linked. However, the performance of a database can be significantly weakened because of the data structure. So, a relational database will be mainly normalized (normal form) or non-normalized (non-nomal form). This last type is the cause of more than 70% performance issues [SQLpro, 2015]. This difference stems from the fact that a normalized structure is characterized by a large number of small tables, containing very few columns and using the method of "join" capable of providing a high speed to perform various operations. While a non-normalized structure contains bigger tables, providing lower performance for write operations (INSERT, UPDATE or DELETE) given that the DBMS (DataBase Management System) activates a lock on the modified table, preventing other programs to access it.

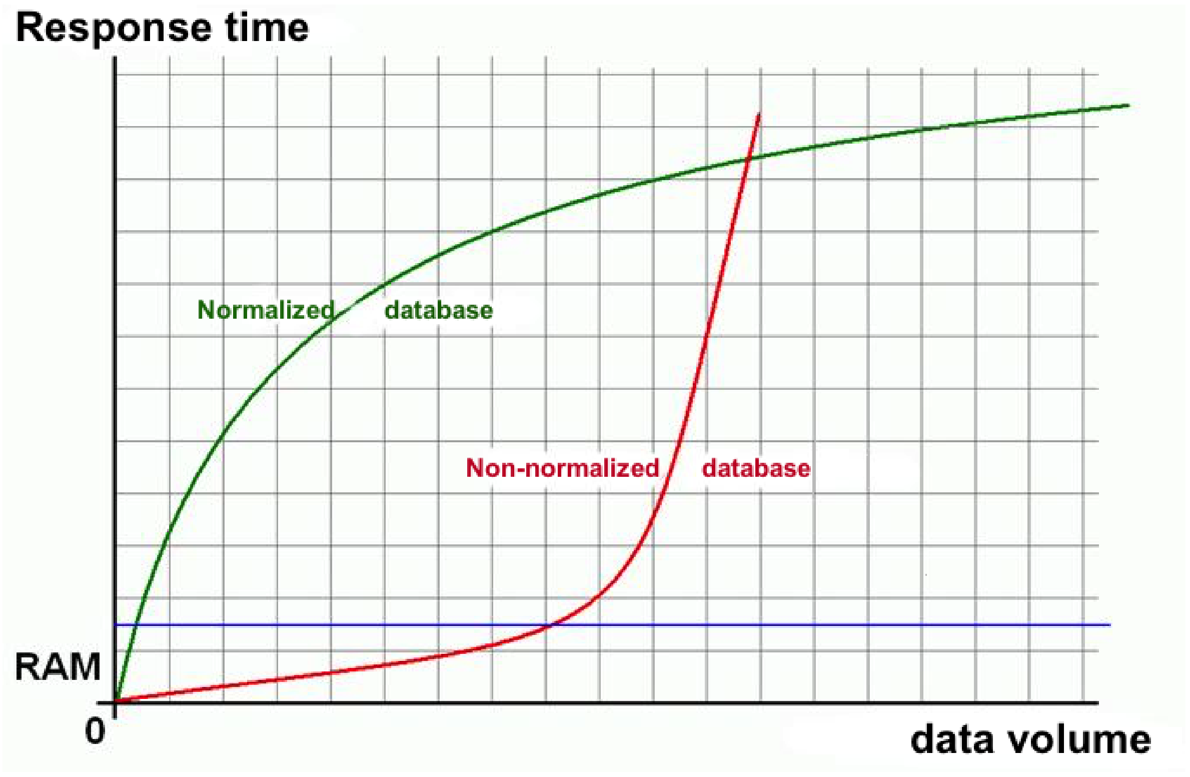


Figure 9.1: Normalized and Non-normalized database

The figure 9.1 shows the correlation between response time and data volume for the normalized- and non-normalized databases. Therefore, when the data volume increases, a normalized database will continue to provide a reasonably constant response time, unlike the other structure which, from a certain threshold, can not provide an answer within acceptable time.

At a time when population produces more and more data each day to such an extent that between 2010 and 2012, more than 90% of information ever created before was produced, equivalent to 2.5 quintillion bytes of data [IBM, 2015]. Data are collected from a plethora of electronic devices such as sensors, medical bracelet, smartphones, etc. This is also called “Big Data”. Therefore, technologies referenced by the “Big Data” involve a large number of data of various types which are changing rapidly [MongoDB, 2015]. The volume (nowadays, tons of data are processed, for example, Facebook processes more than 500 terabytes of data each day), the velocity (tons of data are transmited simultaneously and in a very short time, thousands of machines communicate with each other) and the variety of data (data are not only “string”, “date” or “number” but can be other types such as video, photo, audio, etc.) are key concepts of “Big Data”.

Coming back to the comments of the figure 9.1, the rise of “Big Data“ in society does not allow settling for a non-normalized database because of the large volume of data. Moreover, relational databases are not suitable for large data volume [Hainaut, 2015].

Moreover, the performance of databases does not only depend on response time and data volume. Naturally, the database performance also depends on throughput, defined by the capacity of a database to process data, resources available such as memory, storage disk, cache controllers, CPU, etc [Mullins, 2015]. Therefore, the concept of availability becomes significant in the definition of “performance”. A database, which is not available, is naturally not efficient. However, the reciprocal is not true because an available database does not involve that this database is efficient. So, the availability is metric for measuring the performance.

The availability is an important issue for some business [Subharthi, 2008]. When a company delivers a service, such as a rental system of online DVD (such as Netfix) which rents more than 2 million movies each day all around the world. The system must be continuously available, even during a large number of simultaneous connections. Ten minutes can be very expensive to the company.

Today, the performance of databases is an important issue, which requires a constant evaluation for its measurement. However, the evaluation of database performances may be complicated due to the fact that database systems may be different from each other with specific requirements. Performance analysts nevertheless tried to establish mechanisms allowing the performance evaluation from common key aspects of databases. This is also called “Benchmarks”.

9.2 Performance evaluation : Benchmarking

The goal of this section is to discuss about the evaluation of the performance of databases and more specifically about benchmarks, by studying a few.

Globally, a benchmark is a measure of the quality of policies, products, strategies, etc. of an organization in order to compare them to market leaders or to standard measures. In other words, benchmarks are comparative studies of things with the objectives to get the best.

To give a little history note about benchmarking, the latter has been invented in the 1980s by “Rank Xerox” company, specialized in photocopiers since the 1950s [Xerox, 2015]. While Xerox was the world leader in its field in the 1970s, some Japanese companies entered the market by offering better copiers at a lower cost. And it was when Xerox decided to compare itself to its direct competitor in order to learn more about possibility of reducing costs while increasing productivity. An analysis made it possible due to pinpoint the weak points of the company to make some changes. In the subsequent years, Xerox increased its productivity by 18%. From that moment other companies started to put in place such comparative studies and an interest appeared to become something indispensable nowadays [Blakeman, 2002].

This craze has spread in many areas, other then marketing such as computer sciences and more specifically the databases. The evaluation of performances of databases has two different goals [Subharthi, 2008]:

- To evaluate the best configuration and the operating environment of a single database management system.
- To study several database management systems and to provide a systematic comparison of these systems.

When the comparison between several database management systems is the first objective, the method of benchmarks (aka benchmarking) is the best. The drawback of this method is that it requires the complete installation of the different systems in order to exploit them. According to [Yao et Hevner, 1984], the evaluation of performance of DBMS by benchmark is established through 3 phases :

Benchmark Design: The objective of this phase is to put in place the system environment in order to prepare the benchmark. Moreover, benchmark design includes 4 steps, as shown in figure 9.2. These steps are the hardware and software configuration of the system to prepare the benchmark, test data of database to execute the benchmark, the benchmark workload that consists in considering different aspects of transaction loads (query types, number of users, etc.) and the experimental design based on the three previous steps consists in fixing variables and values which will be used in the benchmark.

Benchmark Execution: This step is realized in order to collect performance data.

Benchmark Analysis: Once the benchmark execution completed, an analysis can be realized in order to compare the results. This third phase allow analyzing of a single system to observe its performances according to different algorithms used, or several systems by comparing the performance of each system according to same criteria.

So, figure 9.2 is a schematic representation of the database system benchmark methodology thinking by the author. It is important to note that when a comparison of several systems is realized, it is essential that the benchmark does not vary from one system to another to have false results.

Furthermore, there are many different types of benchmarks, as pointed out by [Stonebraker et Hellerstein, 1999]. According to them, there are basically two types of benchmarks : those used for comparing price / performance that are also called *Generic benchmark*, while the *Application-Specific benchmarks* intend to restrict to certain classes of workloads. This last type of benchmark contains different “families” of benchmarks. The best known is the “TPC” family (Transaction Processing Performance Council). Here are the different families of benchmarks observed by Stonebraker :

SPEC: SPEC or “System Performance Evaluation Cooperative” is a family of benchmarks intended to maintain a certain standardization of a set of different benchmarks used to measure performance of CPU, graphic, workstations, virtualization, etc[1].

The Perfect Club: This family of benchmarks aims to represent scientific supercomputer workloads. The performance of these benchmarks is defined in term of MFLOPS (Million FLoating-point Operations Per Second) of each program designed for this family (there are more than 30 programs) on each machine evaluated [Saavedra et Smith,].

[1]A complete list of benchmarks of the SPEC family can be consulted on https://www.spec.org/benchmarks.html#tools

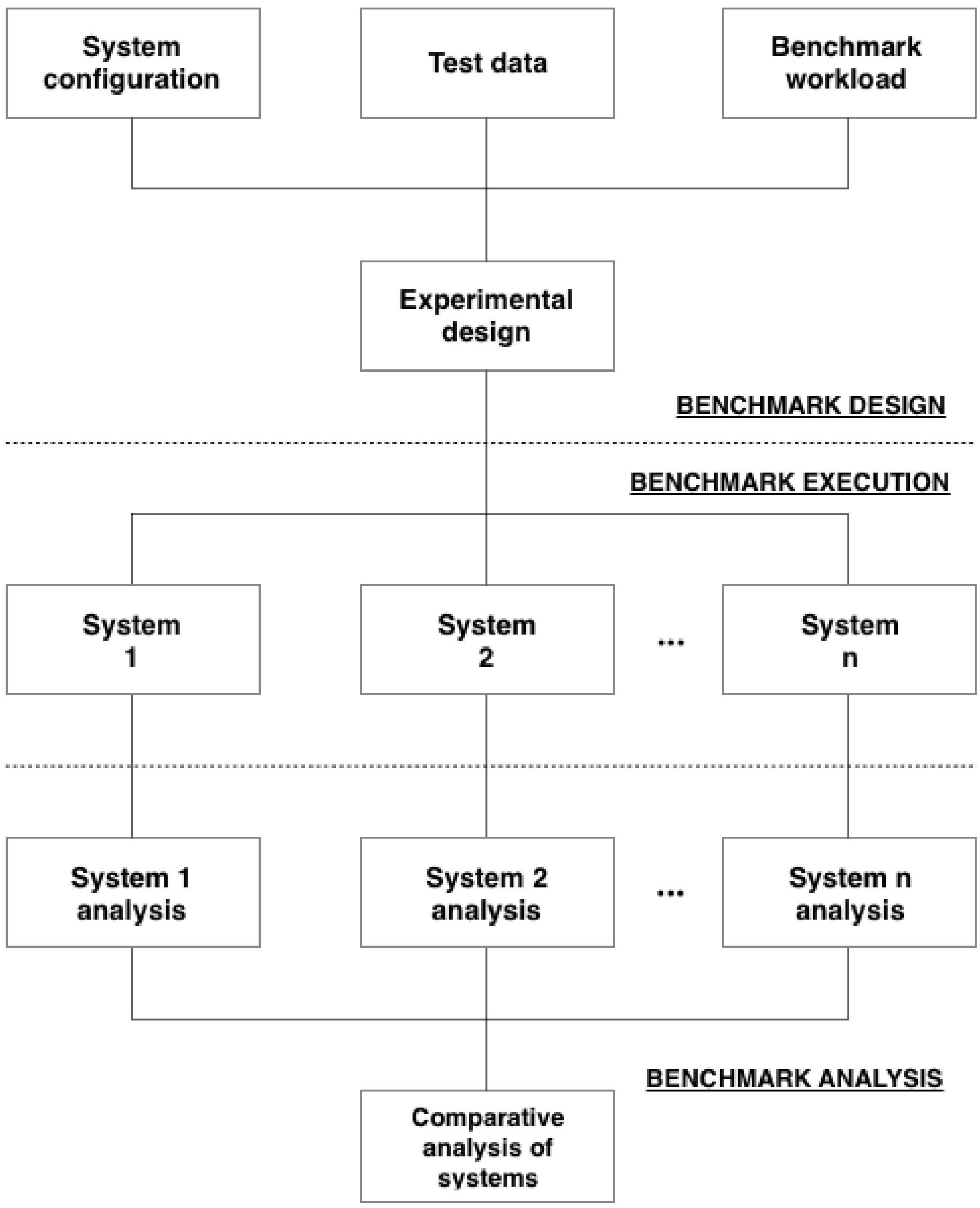


Figure 9.2: Database System Benchmark Methodology [Yao et Hevner, 1984]

TPC: TPC or "Transaction Processing Performance Council" is a family of benchmarks measuring performances of transaction processing and of the database in term of amount of transactions that can be executed in a given amount of time. The characteristic of TPC family is that the benchmark do not only evaluate the components of the database but rather the whole system including the database. This family allows to better understand the business world[2].

[2]A complete list of benchmarks of the TPC family can be consulted on http://www.tpc.org/information/benchmarks.asp

Always according to [Stonebraker et Hellerstein, 1999], a good benchmark must be portable (the benchmark should be capable to be implemented on many different systems), relevant (it must directly translate a useful work), scalable (the benchmark must be understandable). Therefore, what makes a good benchmark is how it is used. Naturally, objectives of each benchmark are all different. It is important to choose the good benchmark according to the system which will be evaluated. The use of a wrong benchmark can involve the collect of results that do not correspond to the desired evaluation. Some errors can be made due to the fact that results infer a false conclusion.

Naturally, there are other performance evaluation methods of databases, other than benchmarks. Without going into detail, certain system performance analysis methods can be extended to databases, such as cost models, queuing models or even simulating modeling (the diligent reader will inform him-/herself on his/her side to learn more about these models).

Furthermore, some mechanisms for measuring the performance of specific databases (such as real-time database systems, web-database systems, enterprise data mining system or even object oriented system) have been developed in order to evaluate such systems more easily. This thesis is not covering such system, thus these types of evaluation are not discussed here.

9.3 Yahoo! Cloud Serving Benchmark

In the following parts of this thesis, the benchmark called "Yahoo! Cloud Serving Benchmark - YCSB" [Cooper et al., 2010] will be used in order to measure the performance of some NoSQL databases. The particularities of the cloud serving system are that it can handle a large amount of dataset, it can set new instances to easily add capacity to a running system and it provides a high level of availability. Therefore, YCSB has been developed to meet those needs.

This benchmark is pretty easy to use because it consists of a data generator and a set of performance test to achieve but requires the complete installation of the different systems. Once installations completed, the YCSB works as a black-box system because users provide a properties file for the selected workload to the YCSB and the YCSB returns a results file. These performance tests consist of running read- and write- (update or insert) operations on the chosen database, this is called "Workload". In this document, four workloads are used:

Workload A: 50% of read operations - 50% of update operations

Workload B: 50% of read operations - 25% of update operations - 25% of insert operations

Workload C: 100% of read operations

Workload D: 50% of update operations - 50% of insert operations

Thus, in order to evaluate NoSQL databases that interest us, 1,000,000 records have been generated via the YCSB benchmark and a certain amount of read-/write operations (from 10,000 to 300,000 operations) have been executed on these records. With such a variation of amount of operations, it is possible to see if there are some differences between a small amount and a large amount of operations on same data.

All those tests have been executed on a virtual machine Ubuntu 12.04.5 LTS 64bit with 4GB of RAM available, hosted on a computer with OSX Yosemite and a total of 8GB of RAM. Furthermore, it is necessary to install the chosen database in order to evaluate its performances.

The next section will evaluate MySQL, MongoDB, Couchbase and Cassandra. Unfortunately, DynamoDB requires lots of money to get representative results and Riak is not supported by YCSB.

Chapter 10

Performance of a sample NoSQL technologies

10.1 Overview

As seen before in the dedicated section of the “Technical background” of this thesis, NoSQL is a storage technology class that has the characteristic of being non-relational. NoSQL is the acronym of “Not only SQL”. The NoSQL databases have been designed with a view to complement relational databases and not to replace them.

Therefore, NoSQL databases have the advantages to be more flexible than traditional relational database management systems, to offer horizontally scalability, high performances, etc. Here is the list of NoSQL families covered in this thesis:

- Key-Value Stores
- Document Stores
- Column Stores
- Graph Databases

This chapter aims to evaluate the performance of different types of NoSQL technologies that support Datomic and the relational database MySQL, used by the OSCAR system. The particularity is that Datomic is not yet implemented on NoSQL databases because the objective here is just to get a sense of performance of those databases. The implementation of Datomic is one of the future works of this thesis.

Assuming that the systems evaluated already contain data, read- and write operations will be performed on them. This fact may have an effect on the performance due to the different mecanisms implemented such as storing data in cached memory. In that way, the cache memory was not empty after each workload to have representative results. In addition, the YCSB benchmark does not implement “join” mecanism (to refer to MySQL).

10.2 Relational Database

10.2.1 YCSB Benchmarking of MySQL

This section presents the results collected through YCSB benchmark, which is presented above. As a reminder, the tests have been performed on a virtual machine Ubuntu 12.04.5 LTS 64bit with 4GB of RAM available, hosted on a computer with OSX Yosemite and a total of 8GB of RAM. Four batteries of tests (also called workload) have been performed by processing between 10,000 et 300,000 operations (read-, update- and insert operation depending on the purpose of workloads) over 1,000,000 records. The results of all Workload can be consulted on the website: "`http://crespeigneromain.wix.com/thesis`".

Moreover, MySQL may store data in cache memory. This memory was not empty after each test in order to simulate a real running system.

	Workload A		**Workload B**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	4343	2302.56	7872	1270.33
20000	6914	2892.68	15458	1293.83
30000	8779	3417.25	22177	1352.75
40000	11361	3520.82	28191	1418.89
50000	14043	3560.49	35909	1392.41
150000	29466	5090.61	146546	1023.57
300000	49002	6122.20	267014	1123.54

Table 10.1: MySQL — Runtime and throughput of Workload A & B

The tables 10.1 and 10.2 contain the time of execution in millisecond and the throughput in operations per second for each workload. The data of table 10.1 differ in the fact that the workload A does not process insert operations, unlike the Workload B which runs 25% of insert operations and write operations. An important observation can be made that the Workload B is slower than Workload A due to the adding of insert operations. The difference is pretty huge when the amount of operations exceed 50,000. At this time of analysis, insert operations draw attention to their negative impact on the performance of MySQL system.

Once more, the table 10.2 shows clear differences in terms of runtime. The runtime of Workload D is 10 times bigger than the runtime of Workload C. This finding is very interesting because it is possible to confirm that the throughput of insert operations tend to really have a negative impact on the total throughput for a large number of operations. Moreover, the runtimes of the Workload A and Workload C show that only read operations

	Workload C		**Workload D**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	3888	2572.02	14468	691.18
20000	6474	3089.28	26882	743.99
30000	8874	3380.66	42655	703.32
40000	10640	3759.39	55838	716.36
50000	12344	4050.55	69465	719.79
150000	28935	5184.03	254160	590.18
300000	52807	5681.07	558622	537.04

Table 10.2: MySQL — Runtime and throughput of Workload C & D

and a mix of read- and write operations are almost similar. This may be explained by the fact that MySQL stores data in cache memory when it is necessary.

To examine this, the tables 10.3, 10.4, 10.5 and 10.6 contain the average latency in microsecond (us) and the amount of operations processed for each operation types. In order to have a visual of these tables, the figures 10.1, 10.2, 10.3 and 10.4[1] are graphs associated to these tables. Note that the axe of abscissa is not to scale because it goes from 50,000 to 150,000 and from 150,000 to 300,000 operations. The software used to draw graphs did not allow that. However, graphs are not biased.

With regard to the figure 10.1, the average latency of write operations (red color) is constantly bigger than the average latency of read operations. The both read- and write operations tend to decrease when the amount of operations increases to become almost similar with a lot of operations. This may be a strange observation because the relational database are not designed to handle a large quantity of operations but the cache memory mechanism may be an explication.

The figure 10.2 is very interessant. There is a contrast between read-, update- and insert operations. The figure 10.2 clearly shows the negative impact of insert operations on the average latency. In addition, the average latency of insert operations increases when the amount of operations also increases. This is an important observation because the weakness of MySQL with insert operations appears not to be open to question. This finding is also observed in the figure 10.4.

The figures 10.1 and The figure 10.3 illustrate the same curve for the read operations. In that way, it is possible to observe that update operations do not influence the average latency of the other.

[1]The appendix A - section “MySQL” - contains these figures with a bigger size to get a better look.

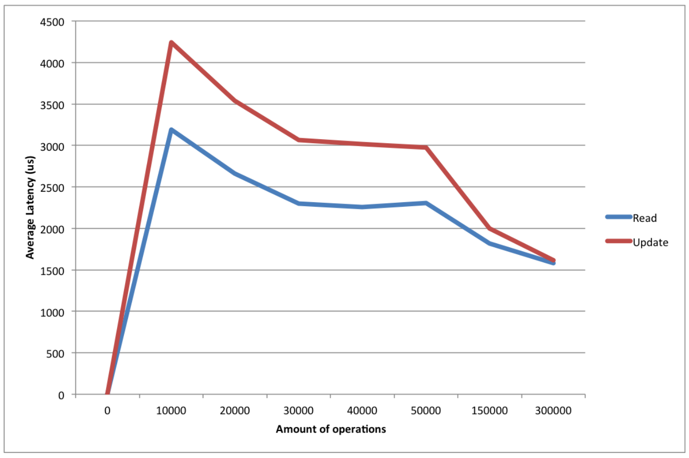


Figure 10.1: MySQL Workload A

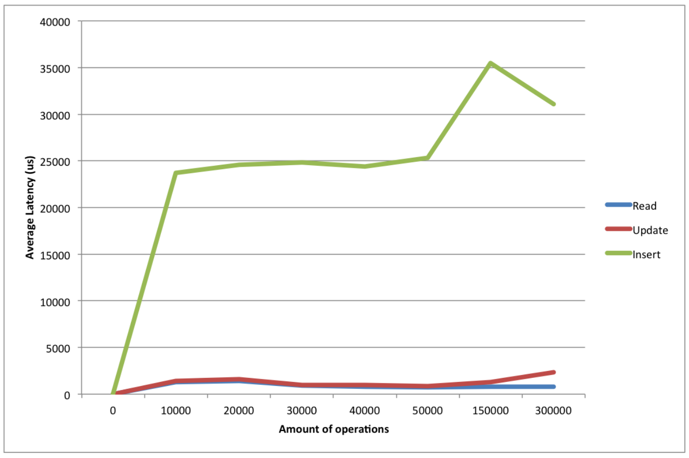


Figure 10.2: MySQL Workload B

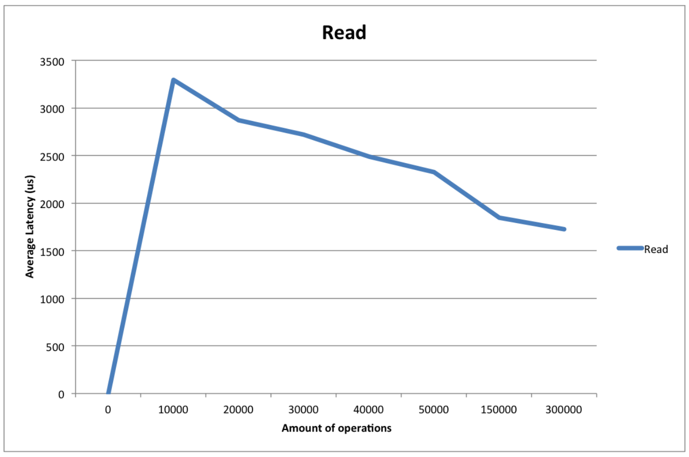


Figure 10.3: MySQL Workload C

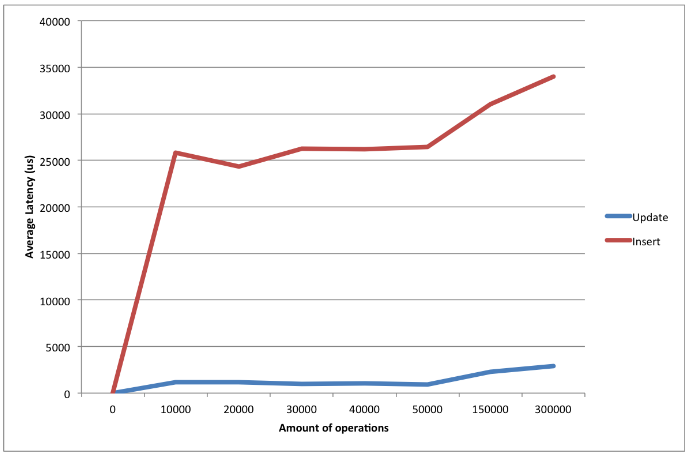


Figure 10.4: MySQL Workload D

Therefore, it is important to bear in mind that the machine (Linux 12.04) used to perform tests may have ran other processes. Here was present the best case for MySQL, that no “Join” mecasnism was tested. In conclusion, the performance evaluation of MySQL reveals that this database is not appropriated for insert operations. For a large amount of operations, the average latency of insert operations is too big to be efficient, especially in system where this type of operation is current. However, the mechanism of cache memory is getting interesting when there are large amounts of operations processed.

Workload A	Read		Update	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	3192.04	5002	4240.87	4998
20000	2663.16	10083	3539.27	9917
30000	2299.66	14944	3064.93	15056
40000	2257.39	19899	3015.24	20101
50000	2308.26	25186	2973.73	24814
150000	1817.90	75495	1996.43	74505
300000	1581.43	150116	1615.12	149884

Table 10.3: MySQL Workload A

Workload B	Read		Update		Insert	
Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations
10000	1301.30	5022	1443.06	2489	23685.31	2489
20000	1392.67	10007	1605.52	4981	24559.43	5012
30000	913.65	14930	975.43	7506	24856.59	7564
40000	781.39	19873	957.56	10093	24403.13	10034
50000	706.76	25066	845.80	12405	25344.16	12529
150000	787.79	74956	1307.14	37365	35472.04	37679
300000	790.18	149816	2332.14	74905	31061.02	75279

Table 10.4: MySQL Workload B

Workload C	Read	
Operations	AverageLatency (us)	Operations
10000	3293.61	10000
20000	2867.14	20000
30000	2717.28	30000
40000	2488.49	40000
50000	2323.85	50000
150000	1843.94	150000
300000	1728.23	300000

Table 10.5: MySQL Workload C

Workload D	Update		Insert	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	1145.52	5003	25794.39	4997
20000	1162.34	9946	24364.21	10054
30000	1013.80	14980	26264.22	15020
40000	1065.52	19969	26219.23	20031
50000	950.95	25219	26430.21	24781
150000	2284.17	74614	30998.34	75386
300000	2934.71	150209	33976.47	149791

Table 10.6: MySQL Workload D

10.3 Document store

10.3.1 YCSB Benchmarking of MongoDB

This section presents the results collected through YCSB benchmark, which is presented above. As a reminder, the tests have been performed on a virtual machine Ubuntu 12.04.5 LTS 64bit with 4GB of RAM available, hosted on a computer with OSX Yosemite and a total of 8GB of RAM. Four batteries of tests (also called workload) have been performed by processing between 10,000 et 300,000 operations (read-, update- and insert operation depending on the purpose of workloads) over 1,000,000 records. The results of all Workload can be consulted on the website: "`http://crespeigneromain.wix.com/thesis`".

	Workload A		**Workload B**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	4713	2121.79	4479	2232.64
20000	7412	2698.33	5921	3377.81
30000	8721	3439.97	8042	3730.42
40000	10058	3976.93	11306	3537.94
50000	11060	4520.8	13299	3759.68
150000	24341	6162.44	37082	4045.09
300000	63100	4754.36	81964	3660.14

Table 10.7: MongoDB — Runtime and throughput of Workload A & B

The tables 10.7 and 10.8 contain the time of execution in millisecond and the throughput in operations per second for each workload. The data of table 10.7 differ in the fact that the workload A does not process insert operations, unlike the Workload B which runs 25% of insert operations and write operations. Under 40,000 operations, the Workload B is a bit faster than the Workload A. When the number of operations exceeds 40,000, it seems that the Workload B becomes slower than the Workload A. This difference may be explained by the adding insert operations, which would tend to show that the insert operations have an impact on the runtime during the execution of large amounts of operations.

For its part, the table 10.8 shows clear differences in terms of runtime. The runtime of Workload C is bigger than the runtime of Workload D. Once there are large amount of operation processed (300,000 operations), the execution time is doubled when there are only read operations. This finding is very interesting because it is possible to observe that the throughput of read operations tend to have a negative impact on the total throughput for a large number of operations. Moreover, it is possible to observe that update operations tend to have a positive impact on the total throughput. The tables 10.7 and 10.7 show that

	Workload C		**Workload D**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	2999	3334.44	4420	2262.44
20000	5498	3637.69	8128	2460.63
30000	6334	4736.34	10528	2849.54
40000	8025	4984.42	11683	3423.78
50000	9171	5451.97	10488	4767.35
150000	26791	5598.9	39561	3791.61
300000	97929	3063.44	54298	5525.07

Table 10.8: MongoDB — Runtime and throughput of Workload C & D

write operations compensate the poor runtime of read operations when 300,000 operations are processed.

To examine this, the tables 10.9, 10.10, 10.11 and 10.12 contain the average latency in microsecond (us) and the amount of operations processed for each operation types. In order to have a visual of these tables, the figures 10.5, 10.6, 10.7 and 10.8[2] are graphs associated to these tables. Note that the axe of abscissa is not to scale because it goes from 50,000 to 150,000 and from 150,000 to 300,000 operations. The software used to draw graphs did not allow that. However, graphs are not biased.

With regard to the figure 10.5, it is interesting to see that the average latency of write operations (red color) is constantly bigger than the average latency of read operations. The both read- and write operations tend to decrease when the amount of operations increases. Moreover, a peak of average latency is observed when there are a few operations.

The figure 10.6 reveals the same finding. An observation can be made that the average latency of insert operations tends to decrease and to become smaller than the average of update operations. Once again, the average latency decrease when the number of operations increases. This finding is also observed in the figure 10.8.

In contradiction with the table 10.9 where the runtime was the biggest, the figure 10.7 shows that the average latency is pretty small and decreases when the amount of operations increases. At 300,000 operations, the average latency is the smaller than other average latencies observed.

Therefore, how to explain the finding obtained from the table 10.8? This can be explained by the machine used to perform tests. The system (Linux 12.04) may have ran

[2]The appendix A - section "MongoDB" - contains these figures with a bigger size to get a better look.

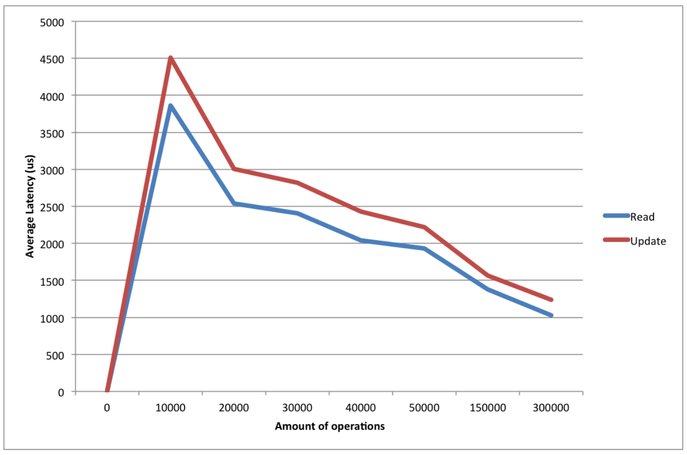


Figure 10.5: MongoDB Workload A

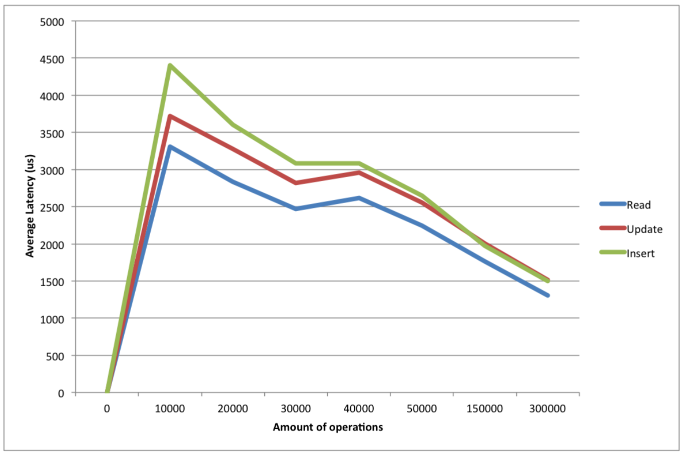


Figure 10.6: MongoDB Workload B

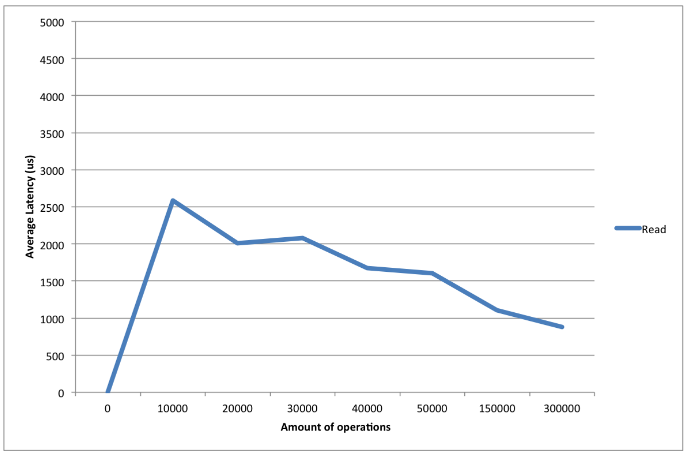


Figure 10.7: MongoDB Workload C

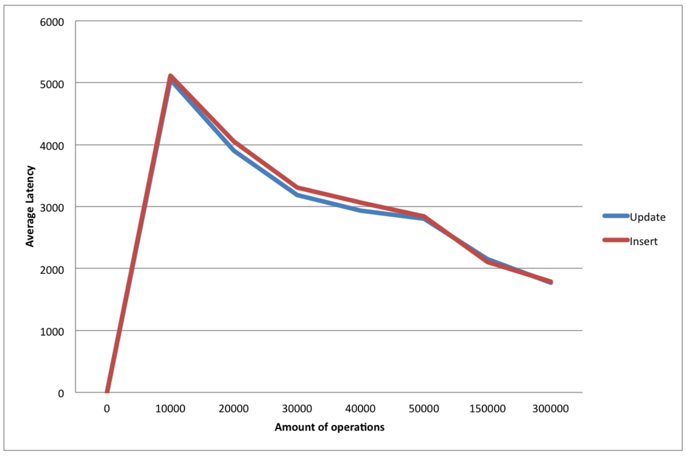


Figure 10.8: MongoDB Workload D

other processes that impacted the read operations. In conclusion, the performance evaluation of MongoDB reveals that this database is appropriated for read operations. The write operations are getting interesting when there are large amounts of operations processed. This operation type has a negative impact on the performance with a few operations.

Workload A	Read		Update	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	3866.55	4972	4510.36	5028
20000	2539.13	10005	3009.02	9995
30000	2404.73	15010	2823.12	14990
40000	2043.71	19828	2432.99	20172
50000	1932.12	24868	2223.14	25132
150000	1380.06	75125	1569.19	74875
300000	1027.47	150121	1238.08	149879

Table 10.9: MongoDB Workload A

Workload B	Read		Update		Insert	
Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations
10000	3308.58	5004	3717.87	2517	4399.54	2479
20000	2832.50	9969	3276.67	5007	3606.06	5024
30000	2471.58	14999	2818.84	7492	3085.26	7509
40000	2615.46	20011	2961.69	9940	3084.09	10049
50000	2241.27	24929	2558.83	12490	2646.85	12581
150000	1762.54	74995	2005.09	37578	1969.42	37427
300000	1303.77	149821	1517.16	75071	1501.66	75108

Table 10.10: MongoDB Workload B

Workload C	Read	
Operations	AverageLatency (us)	Operations
10000	2587.46	10000
20000	2009.45	20000
30000	2080.01	30000
40000	1678.13	40000
50000	1605.25	50000
150000	1106.85	150000
300000	880.37	300000

Table 10.11: MongoDB Workload C in term of operations per second

Workload D	Update		Insert	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	5054.36	4924	5113.89	5076
20000	3901.33	10051	4056.98	9949
30000	3191.77	15068	3307.57	14932
40000	2938.13	19890	3065.44	20110
50000	2809.86	24888	2841.28	25112
150000	2151.32	74817	2108.09	75183
300000	1768.51	150202	1794.70	149798

Table 10.12: MongoDB Workload D in term of operations per second

10.3.2 YCSB Benchmarking of Couchbase

This section presents the results collected through YCSB benchmark, which is presented above. As a reminder, the tests have been performed on a virtual machine Ubuntu 12.04.5 LTS 64bit with 4GB of RAM available, hosted on a computer with OSX Yosemite and a total of 8GB of RAM. Unlike the others, the YCSB Benchmarking of Couchbase did not benefit of the entire RAM capacity of the virtual machine because Couchbase needs a part of the RAM capacity to simulate its own server. In that way, 2GB of RAM was dedicated to Couchbase and 2GB of RAM was dedicated to the virtual machine. This has a significant impact on the results.

Four batteries of tests (also called workload) have been performed by processing between 10,000 et 300,000 operations (read-, update- and insert operation depending on the purpose of workloads) over 1,000,000 records. The results of all Workload can be consulted on the website: "`http://crespeigneromain.wix.com/thesis`".

	Workload A		**Workload B**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	5990	1669.45	5859	1706.77
20000	10024	1995.21	9594	2084.64
30000	10485	2861.23	11874	2526.53
40000	12631	3166.81	14000	2857.14
50000	15047	3322.92	15699	3184.92
150000	32489	4616.95	35625	4210.53
300000	54109	5544.36	71002	4225.23

Table 10.13: Couchbase — Runtime and throughput of Workload A & B

The tables 10.13 and 10.14 contain the time of execution in millisecond and the throughput in operations per second for each workload. The difference of data in the table 10.13 is explained by the fact that the workload A does not process insert operations, unlike the workload B which runs 25% of insert operations and write operations. Under 50,000 operations, there is no significant difference between runtime of each workload. Once the number of operations reaches 150,000, a significant difference appears and shows that the Workload A is faster than Workload B. This may be explained by the fact that adding insert operations has a negative impact on the runtime execution of large amounts of operations.

While there is a minor difference in the table 10.13, the table 10.14 directly shows major differences in terms of runtime between the Workload C, which runs 100% of read operations, and the Workload D, which runs 50% of update operations and 50% of

	Workload C		**Workload D**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	5026	1989.65	10661	938
20000	7606	2629.50	15497	1290.57
30000	9084	3302.51	19241	1559.17
40000	10957	3650.63	20607	1941.09
50000	12624	3960.71	25158	1987.44
150000	30374	4938.43	51079	2936.63
300000	44743	6704.96	89763	3342.13

Table 10.14: Couchbase — Runtime and throughput of Workload C & D

insert operations. The execution time is doubled when there only are write operations. A first observation could be made that the throughput of insert operations tend to have a negative impact on the total throughput in the Workload A and B in the table 10.13. Therefore, the runtime of the Workload D shows that only write operations tend to have a negative impact on the total throughput. A reason could be that the insert operations have a significant impact.

To deepen these results, the tables 10.15, 10.16, 10.17 and 10.18 contain the average latency in microsecond (us) and the amount of operations processed for each operation types. In order to have a visual of these tables, the figures 10.9, 10.10, 10.11 and 10.12[3] are graphs associated to these tables. Note that the axe of abscissa is not to scale because it goes from 50,000 to 150,000 and from 150,000 to 300,000 operations. The software used to draw graphs did not allow that. However, graphs are not biased.

With regard to the figure 10.9, it is interesting to see that the average latency of write operations (red color) is constantly bigger than the average latency of read operations but tend to significantly decreases when the amount of total operations increase. Moreover, a peak of average latency is observed when there are a small number of operations.

The figure 10.10 reveals different findings. Firstly, the average latency of read operations seems to be consistent while the average latency of write operations tend to decrease. Secondly, the insert operations have a huge average latency when the amount of operations is small. Thirdly, a large number of operations significantly decreases the average latency of write operations and seems to be consistent between 150,000 and 300,000 operations. This finding is also observed in the figure 10.12. However, the average latency of 25% insert operations from 8,000 us to 11,000 us when there are 50% of insert operations.

[3]The appendix A - section "Couchbase" - contains these figures with a bigger size to get a better look.

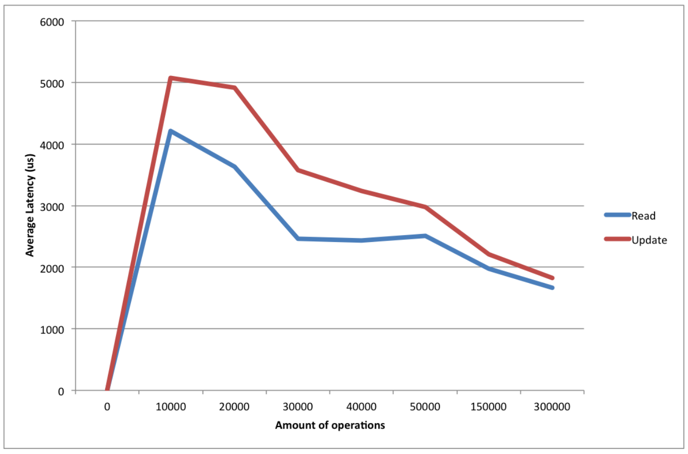


Figure 10.9: Couchbase Workload A

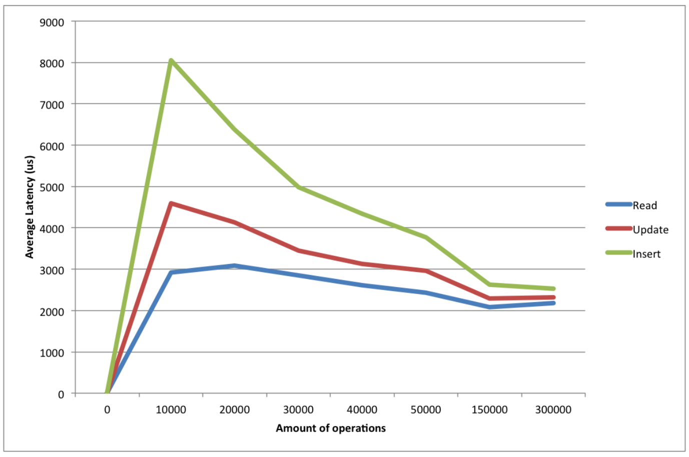


Figure 10.10: Couchbase Workload B

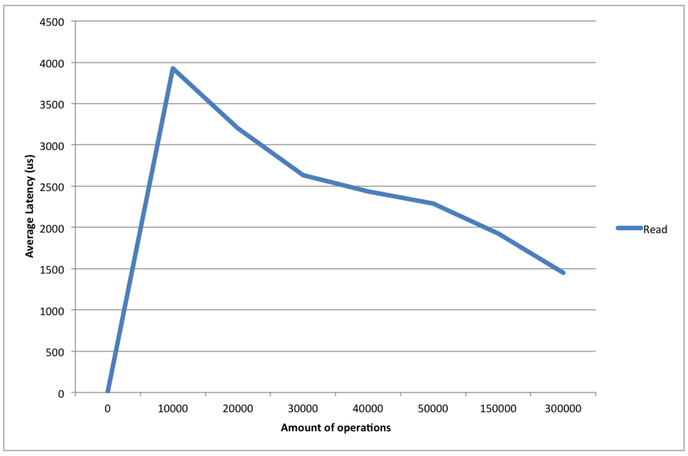


Figure 10.11: Couchbase Workload C

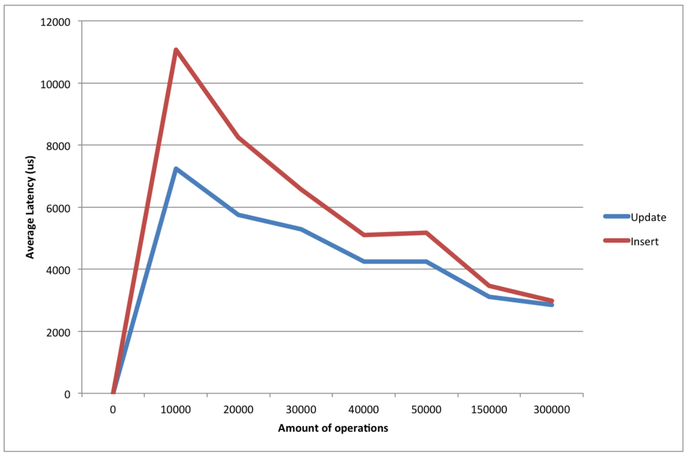


Figure 10.12: Couchbase Workload D

According to the table 10.15 where the runtime was the smallest, the figure 10.11 shows that the average latency is pretty small and decreases when the amount of operations increases to finally be below 1,500 us at 300,000 operations. This is the smaller average latencies observed.

Therefore, these results show that insert operations have negative impact on the system when a few operations are executed. However, it seems that the negative impact of insert operations (and even update operations) with numerous operations tend to disappear in term of average latency. Like MongoDB, Couchbase is beneficial for read operations, while write operations are not appropriated when a small of operations is processed but is suitable for a large number of operations.

Workload A	Read		Update	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	4209.03	4944	5070.87	5056
20000	3635.31	10036	4918.51	9964
30000	2464.56	14987	3574.92	15013
40000	2436.51	20018	3239.31	19982
50000	2510.16	24860	2978.75	25140
150000	1973.84	75007	2208.15	74993
300000	1665.86	149833	1830.1	150167

Table 10.15: Couchbase Workload A

Workload B	Read		Update		Insert	
Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations
10000	2920.33	5039	4587.77	2448	8053.08	2513
20000	3082.62	10000	4137.40	5011	6381.08	4989
30000	2853.11	15098	3446.84	7460	4983.48	7442
40000	2617.27	19979	3127.28	9920	4338.83	10101
50000	2429.66	25038	2960.24	12553	3765.78	12409
150000	2076.82	74988	2295.19	37592	2624.19	37420
300000	2175.11	150309	2314.55	74760	2529.62	74931

Table 10.16: Couchbase Workload B

Workload C	Read	
Operations	AverageLatency (us)	Operations
10000	3929.82	10000
20000	3195.64	20000
30000	2634.59	30000
40000	2433.94	40000
50000	2287.6	50000
150000	1927.77	150000
300000	1452.30	300000

Table 10.17: Couchbase Workload C

Workload D	Update		Insert	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	7232.93	4940	11064.15	5060
20000	5746.79	9872	8241.40	10128
30000	5292.69	15020	6570.02	14980
40000	4240.90	19982	5105.08	20018
50000	4236.05	24985	5175.19	25015
150000	3101.88	74819	3461.76	75181
300000	2842.79	150073	2983.23	149927

Table 10.18: Couchbase Workload D

10.4 Column-Oriented Databases

10.4.1 YCSB Benchmarking of Cassandra

This section presents the results collected through YCSB benchmark, which is presented above. As a reminder, the tests have been performed on a virtual machine Ubuntu 12.04.5 LTS 64bit with 4GB of RAM available, hosted on a computer with OSX Yosemite and a total of 8GB of RAM. Four batteries of tests (also called workload) have been performed by processing between 10,000 et 300,000 operations (read-, update- and insert operation depending on the purpose of workloads) over 1,000,000 records. The results of all Workload can be consulted on the website: "`http://crespeigneromain.wix.com/thesis`".

	Workload A		**Workload B**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	5990	1669.45	5859	1706.78
20000	10024	1995.21	9594	2084.64
30000	10485	2861.23	11874	2526.53
40000	12631	3166.81	14000	2857.14
50000	15047	3322.92	15699	3184.92
150000	32489	4616.95	35625	4210.53
300000	54109	5544.36	71002	4225.23

Table 10.19: Cassandra — Runtime and throughput of Workload A & B

The tables 10.19 and 10.20 contain the time of execution in millisecond and the throughput in operations per second for each workload. Let observe the data from the table 10.19, where the difference between workload A and B lies in the proportion of update- and insert operations. Between 10,000 and 150,000 operations, the gaps between runtimes are not significant but once in 300,000 operations, a difference of more than 15,000 ms (15 sec) is felt. This difference may be explained by the adding insert operations, which would tend to show that the insert operations have an impact on the runtime during the execution of a large amount of operations.

For its part, the table 10.20 shows clear differences in terms of runtime. The execution time is doubled when there only are write operations. The correlation of these four tests clearly shows that the execution time is negatively impacted by write operations. However, it is not possible to say at the sight of these simple tables that the speed of read operations are better than the speed of write operations, which would contradict what was said at the section dedicated to "Cassandra" in chapter "Fundamentals".

	Workload C		**Workload D**	
Operations	RunTime(ms)	Ops/sec	RunTime(ms)	Ops/sec
10000	5026	1989.65	10661	937.99
20000	7606	2629.50	15497	1290.57
30000	9084	3302.51	19241	1559.17
40000	10957	3650.63	20607	1941.09
50000	12624	3960.71	25158	1987.44
150000	30374	4938.43	51079	2936.63
300000	44743	6704.96	89763	3342.13

Table 10.20: Cassadra — Runtime and throughput of Workload C & D

To examine this, the tables 10.21, 10.22, 10.23 and 10.24 contain the average latency in microsecond (us) and the amount of operations processed for each operation types. In order to have a visual of these tables, the figures 10.13, 10.14, 10.15 and 10.16[4] are graphs associated to these tables. Note that the axis of abscissa is not to scale because it goes from 50,000 to 150,000 and from 150,000 to 300,000 operations. The software used to draw graphs did not allow that. However, graphs are not biased.

With regard to the figure 10.13, it is interesting to see that the average latency of write operations (red color) is constantly smaller than the average latency of read operations. The read operations tend to greatly increase when the amount of operations increases.

The figure 10.14 reveals the same finding. The difference is that decreasing the proportion of update operations has an impact on the average latency of read operations because the average latency of read- and update operations are substantially similar, until a certain amount of operations performed. For a small quantity of operations, the insert operations have an average latency bigger than the two others but the average latency tend to decrease when the amount of performed operations increase. Thus, for numerous performed operations, the speed of processing of write operations are better than the processing of read operations.

The figures 10.15 and 10.16 support previous observations, that when a large amount of operations are performed, the average latency of read operation also tends to increase. On the other side, the average latency of write operations tend to decrease when the number of operations increase. So, Cassandra is more appropriate for systems that require a lot of write operations on a large quantity of operations. It is not designed to be used for a small amount of operations.

[4]The appendix A - section “Cassandra” - contains these figures with a bigger size to get a better look.

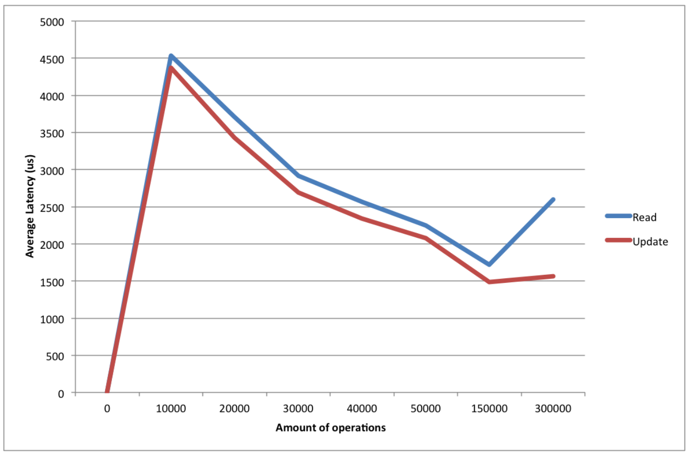


Figure 10.13: Cassandra Workload A

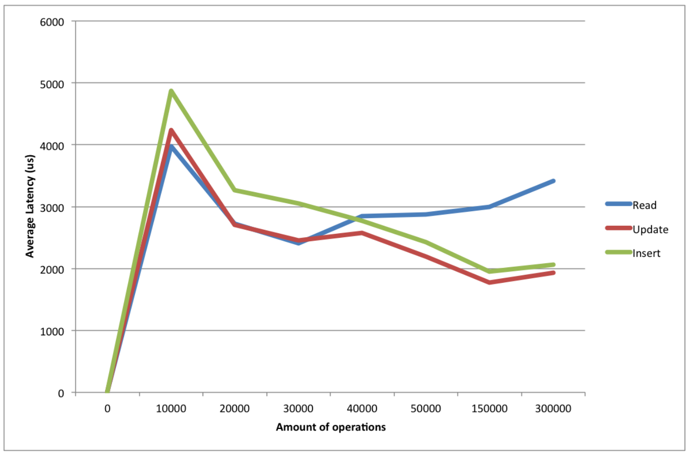


Figure 10.14: Cassandra Workload B

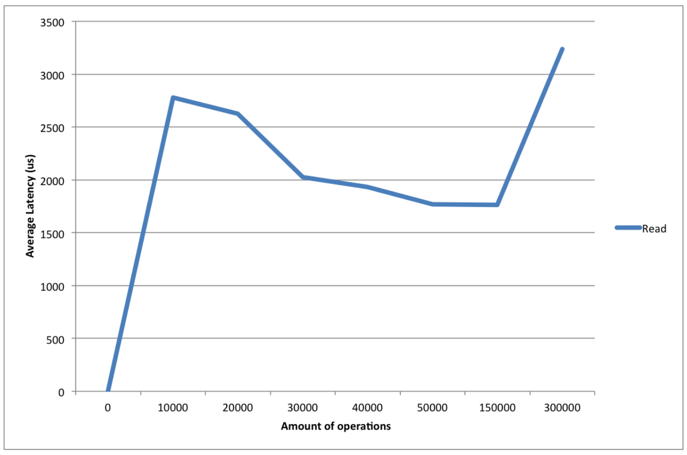


Figure 10.15: Cassandra Workload C

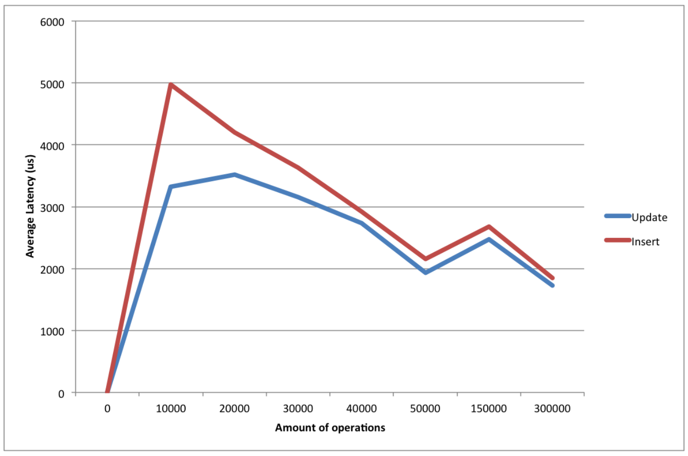


Figure 10.16: Cassandra Workload D

Workload A	Read		Update	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	4530.35	5093	4373.62	4907
20000	3708.47	10004	3431.08	9996
30000	2916.27	14949	2689.05	15051
40000	2570.52	19980	2339.89	20020
50000	2244.24	24838	2075.2	25162
150000	1716.09	74795	1483.77	75205
300000	2598.31	150533	1562.95	149467

Table 10.21: Cassandra Workload A

Workload B	Read		Update		Insert	
Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations	Average-Latency (us)	Operations
10000	3979.94	5022	4233.16	2497	4870.29	2481
20000	2725.68	9956	2707.37	5061	3264.30	4983
30000	2404.16	15004	2457.14	7485	3051.40	7511
40000	2849.74	20049	2575.30	9939	2769.91	10012
50000	2871.36	24954	2194.22	12571	2423.85	12475
150000	3000.13	75173	1773.54	37230	1953.24	37597
300000	3411.58	149956	1929.84	74823	2063.59	75221

Table 10.22: Cassandra Workload B

Workload C	Read	
Operations	AverageLatency (us)	Operations
10000	2781.81	10000
20000	2628.22	20000
30000	2028.28	30000
40000	1934.85	40000
50000	1771.24	50000
150000	1761.46	150000
300000	3237.78	300000

Table 10.23: Cassandra Workload C

Workload D	Update		Insert	
Operations	AverageLatency (us)	Operations	AverageLatency (us)	Operations
10000	3322.23	4967	4977.48	5033
20000	3514.95	10046	4203.88	9954
30000	3153.31	15091	3631.13	14909
40000	2734.84	19923	2921.28	20077
50000	1937.18	24923	2157.55	25077
150000	2473.46	74994	2683.55	75006
300000	1726.11	150235	1847.67	149765

Table 10.24: Cassandra Workload D

10.5 Short comparison

In conclusion of this chapter, a short comparison of the performance of MySQL, MongoDB, Cassandra and Couchbase will be made. It is important to bear in mind that Couchbase had 2GB of RAM allocated to its server and that MongoDB does not support Datomic. Therefore, the results of Couchbase can not be compared to other database. However, the table 10.25 contains all average latencies for each workload of each database for 300,000 operations, including Couchbase.

In that way, MySQL is not appropriated for insert operations, unlike the others. The workloads B and D show that the average latencies of insert operations for a large amount of operations are respectively more than 30,000 us and more than 34,000 us. The strength of MySQL lies in the cach memory mechanism which provides low latencies for read- and update operations. MongoDB had the best results for each workload. So, MongoDB shows that it is appropriated for each type of operations.

While Couchbase was performed with just 2GB of RAM, its results follow the same general curve than MongoDB. Moreover, this database has best average latency for read operations than Cassandra. So, Cassandra is really not appropriated for read operations. Therefore, its results for write operations seems to be satisfactory because it is why Cassandra is designed.

Workloads		MySQL	MongoDB	Cassandra	Couchbase
A	Read	1581.43	1027.47	2598.31	1665.86
	Update	1615.12	1238.08	1562.95	1830.1
B	Read	790.18	1303.77	3411.58	2175.11
	Update	2332.14	1517.16	1929.84	2314.55
	Insert	31061.02	1501.66	2063.59	2529.62
C	Read	1728.23	880.37	3237.78	1452.30
D	Update	2934.71	1768.51	1726.11	2842.79
	Insert	33976.47	1794.70	1847.67	2983.23

Table 10.25: Average latency for each workload of each database

Chapter 11

Conclusion

At the end of this thesis, we can finally address our initial research questions. We conclude that a migration from an existing relational database towards a non-relational database is possible. We described in this thesis a method to perform that migration which is decomposed in multiple steps.

In the context of that migration method, the first step is to migrate the structure. This step aims at establishing a mapping between a schema which fits with the source data model and a schema which fits with the target data model (cf. figure 4.2, p. 88).

On the one hand, the database reverse engineering has for a goal to retrieve the conceptual schema representing the domain supported by the database studied from both the source DDL code and the program using that database. We did not retrieve the smallest detail about this process but showed how it is possible to apply it. That being said, the bottom line is the fact that this process can be applied on any database from the moment it respects the relational model. Since this process is the first part of our chosen approach (cf. figure 4.4, p. 89), it goes without saying that the starting point of our migration method covers all relational database management systems. Also, through all steps of the database reverse engineering process (i.e. physical extraction, logical extraction and conceptualization), we can state that the final result is a conceptual schema devoid of any technological constraint.

On the other hand, there is the database forward engineering process. This process has for a goal to go from the conceptual schema fully abstract (produced by the database reverse engineering process) towards an understandable code for a NoSQL technology. In other words, this process is the descending part of our chosen approach about the schema conversion into our method of migration pictured in the figure 4.4, page 89. However, we were unable to generalize the target of our migration to the whole of non-relational databases by their lack of standardization. Nevertheless, we proved thanks to a transformation plan and the composition of several designs (i.e. conceptual analysis, logical design, physical design and coding) that it is still possible to ensure lossless transformations between the source conceptual schema and the target code only understandable for Datomic according to many assumptions. Indeed, this process shows how difficult it is to go from the relational to the non-relational world, from rigidity to flexibility. By definition, many constraints from relational databases are contrary to NoSQL and need to be implemented at another layer. That being said, the transformation plan (cf. figure figure 6.12, page 6.12) is implemented under DB-Main where relationship transformations are proved by the Generic Entity Relationship (GER). Also, we have chosen Datomic as target rather than a precise non-relational database technology because Datomic allows covering more than one NoSQL database (cf. Chapter 2, section 2.2.2.1, subsection 2.2.2.1.b, p. 37).

Therefore, when we combine these two processes, we ensure that the schema conversion of our migration method is lossless between the DDL source code in SQL and the target code understandable for Datomic. Also, this first step of our migration method is the first filter of what can be migrated and what cannot be without making any assumption.

Once the structure is converted, the next step is to aim at synchronizing the data, which means both inserting the existing data in the target database and providing ways to access it. We described for this step a method taking as an input a relational database and its schema transformation towards NoSQL and covering the conversion of both the data itself and the application program(s) that communicate with the database.

The purpose of this step is to complete the database migration process and obtain a new system where the new database, its Data Management System and the modified application program, cohabit in harmony. This new system must at least offer the same functionalities than the legacy one and can possibly extend them depending on the motivations behind the migration, along with preserving the original data's semantics. The proposed method is the following:

1. **Phase 1: data conversion**
   (a) **Select an ETL tool**
   (b) **Perform data migration using this tool**
   (c) (If some data types are too complex to be directly migrated using the tool) **Employ the manual (purely manual or semi-automatic) approach for those types**
2. **Phase 2: program conversion**
   (a) **Maintain both databases by using the *wrapper strategy***
   (b) **Maintain the target database by using the *logic rewriting strategy***

First, we detail the different data conversion approaches and elicit the most suitable one for a database migration from relational to NoSQL and specifically Datomic respecting multiple assumptions, and then describe a method to compare Extract-Transform-Load tools. We then apply the tool Pentaho Data Integration on our study case to illustrate the operation of an ETL tool in the concrete context of the data conversion of the demographic table of Oscar. We describe the process we think is the most natural possible for a data conversion respecting the assumptions we made. The result of those steps is a target database filled with data semantically matching the source database's content.

Then, we report the different program conversion approaches and strategies and suggest a general method combining data safety and soft transition from one system to another but at the cost of performance and time which we assumed the migrating team to dispose. Here again, we describe the process we think ideal for a program conversion respecting the assumptions we made. The result of those steps is a modified application program capable of query and update the new database through the new DMS in an efficient way.

To complete the subject of the migration, another topic has been discussed, namely the performance. Therefore, in the context of computer sciences and more specifically of databases, the performance is an important concept due to the emergence of "Big Data" and can be defined in two ways. The first one defines the performance as being a set of amounted indicators for measuring the elapsed time for a computer system to complete given tasks taking into account time and given resources. And the second one defines the performance as being the computer speed executing a certain number of millions of instructions per second. The performance is mainly impacted by the data structure of the database but also by resources available, the throughput, etc. So, a lot of concepts are included in the definition of "performance".

Therefore, the performance of databases has different goals, such as the evaluation of the best configuration and the operating environment of a single database management system or to study several database management systems and to provide a systematic comparison of these systems. This performance requires a constant evaluation for its measurement but this constant evaluation of database performances can be complicated due to the fact that database systems may be different from each other with specific requirements. However, performance analysts tried to establish mechanisms allowing the performance evaluation from common key aspects of databases. This is also called "Benchmarks". There also are other methods such as cost models, queuing models, etc.

In that way, a benchmark can be defined as a comparative study of things with the objectives to get the best. Benchmarks were designed to be used in the marketing area but had spread in many areas such as computer sciences. Benchmarking is the best way to compare several database management systems. There are different families of benchmarks but the "Yahoo! Cloud Serving Benchmark" was chosen to compare a sample of NoSQL technologies.

This benchmark requires the complete installation of the different systems and works as a black-box system. Four workloads are used in order to evaluate the performance of the NoSQL technologies. In that way, MySQL, MongoDB, Cassandra and Couchbase have been evaluated with this benchmark. Unfortunately, DynamoDB requires lots of money to get representative results and Riak is not supported by YCSB.

The results of those technologies show differences in terms of performance. In that way, Cassandra is designed to perform a large amount of write operations and shows poor performances when there are only read operations. Unlike Cassandra, MongoDB and Couchbase are faster when there are only read operations and theirs performances are impacted by write operations. This impact is attenuated with a large amount of operations. So, in terms of performance, it is important to know the objectives of the future system in order to choose the best technology.

Chapter 12

Future Work

12.1 Schema conversion

The part about schema conversion is not complete and highlights two major future works.
On the one hand, this part is the first filter between what is possible to migrate and what is not because it is the first phase of our migration method. And we have had to take a lot of assumptions in order to ensure semantics-preserving or at least lossless transformations. These assumptions can nevertheless materialize by an implementation at the application layer or another than the data one. Either required attributes or tables could be implemented by an API for example. Or another artifact which would gather the whole of constraints and features which are not implemented in Datomic. This artifact could be subject to a future work. This would involve an impact analysis of this artifact on the working of Datomic. And then, it would be necessary to challenge the relevance of Datomic as being our target technology of our migration method.

On the other hand, if Datomic is still relevant, the second work which could be interesting would be to enrich this migration method by extending the transformation plan at logical level. Indeed structures like is-a relationship are not considered yet. Even though our transformation plan fits well with our case study, this is not enough to cover all possible schemas. And the final goal would be to have the most general method covering as many areas as possible.

12.2 Data synchronization

The data synchronization process described in this thesis could not be treated exhaustively and raises possibilities of future works. Indeed, some steps or sub-steps of this database migration method represent a content substantial enough to be addressed in a separate work. We identified four tracks of potential further research:

- **Complete ETL tools comparison:** We depicted in this work a method to compare ETL tools using selected criteria and figures of merit. We proposed a weights assignment fitting for a migration corresponding to the context of our case study and respecting assumptions we made. Yet, we did not test tools one by one to fill an evaluation matrix in order to determine the most suitable tool for this particular kind of migration. Performing a comparative analysis of most or all available ETL tools would be a possibility to extend this thesis.

- **Data reconciliation step:** The data reconciliation step ensuring the integrity of data and following the Extract-Transform-Load process has not been explored in this thesis. We described it conceptually in the state of the art but did not go further in its operation nor pictured it on our case study. Detailing the possible approaches relative to this step and illustrating it on a concrete case study would be an interesting extended work.
- **Bridge between Datomic and the target storage service technology:** Established ETL tools do not allow direct loading data in a Datomic database but rather in compatible data storage services. Thus, artifacts specific to Datomic's architecture and operation are not yet taken into account when performing the conversion. Thereby, another possibility of future work would be to concretely set Datomic up in a partition of a compatible data storage technology previously installed and perform a data loading in that target database.
- **Program conversion case study:** The complexity of the Oscar system, due to the diversity of technologies used, the width of the program itself and the number of developers, results in the difficulty of rewriting the program's logic, making the illustration of a concrete program transformation worth a standalone work that would be interesting to link to this thesis.

12.3 NoSQL performance

For this section, there are two major future works. The first one is to evaluate Amazon DynamoDB and Riak. Amazon DynamoDB did not evaluate because it is not open source and is expensive to have a representative evaluation of its performance. If an agreement with Amazon could be reached to use DynamoDB for free for the duration of the performance evaluation would allow to evaluate it without any cost. With a regard to Riak, YCSB does not support this database. It would be interesting to use another benchmark in order to have an idea of its performance. Naturally, it would be inappropriate to compare this benchmark with another benchmark because theirs objectives are different.

The evaluation of the performance of the NoSQL technologies is not made from Datomic. Its objective is to get a sense of the performance of those technologies and to be able to select the most suitable database management system for OSCAR. The second future work is to implement Datomic on the chosen database and to evaluate it. This work can be made on all databases but it will take some time to implement it.

Chapter 13

Appendix A

13.1 MySQL

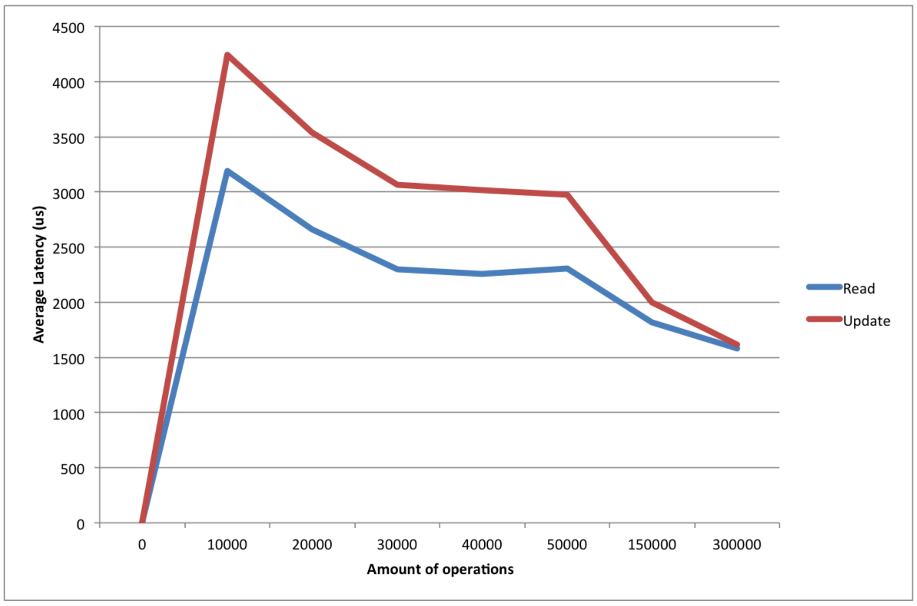


Figure 13.1: MySQL Workload A

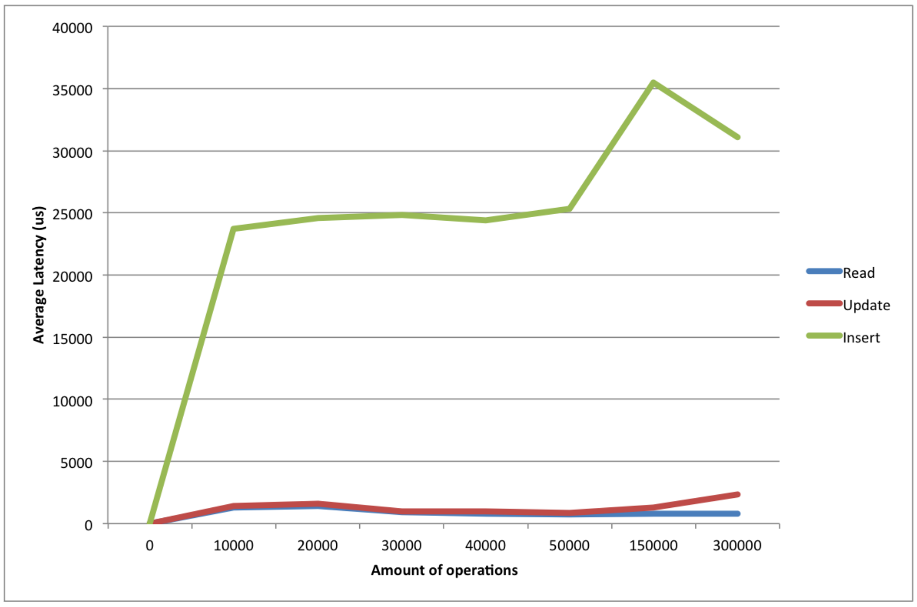


Figure 13.2: MySQL Workload B

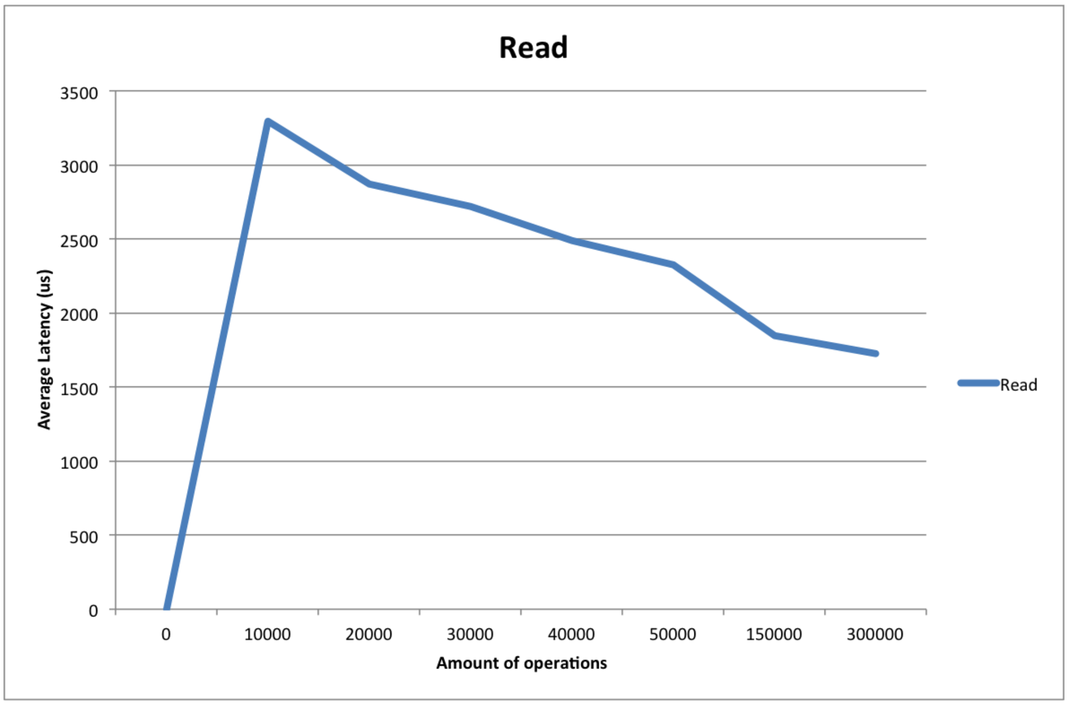


Figure 13.3: MySQL Workload C

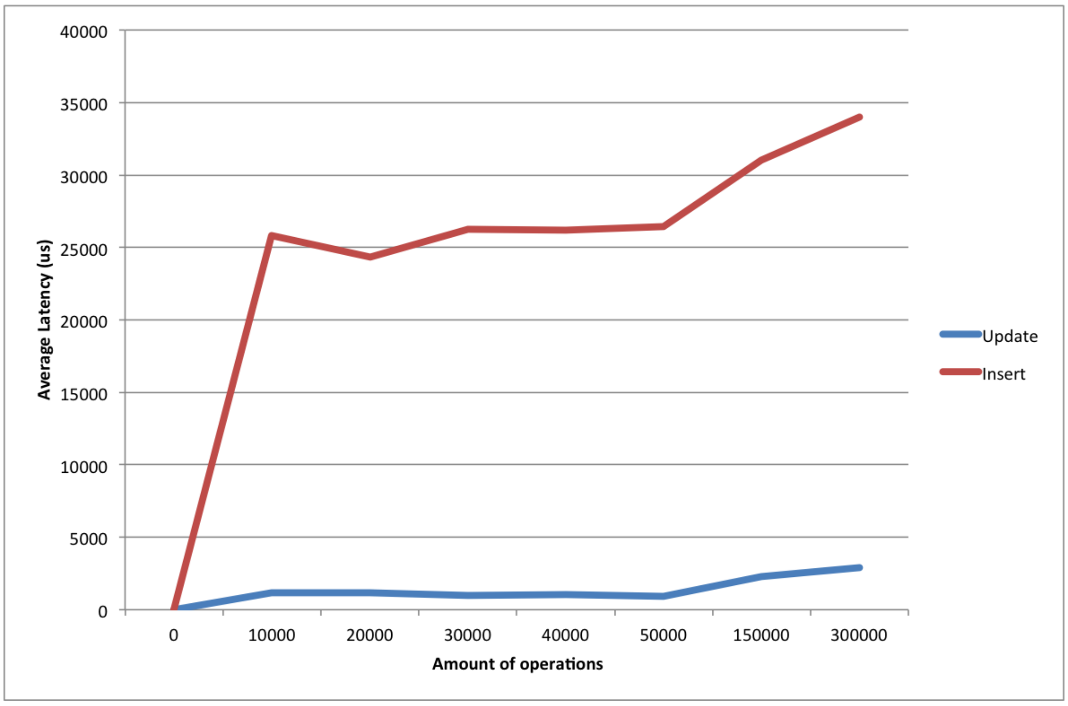


Figure 13.4: MySQL Workload D

13.11 MongoDB

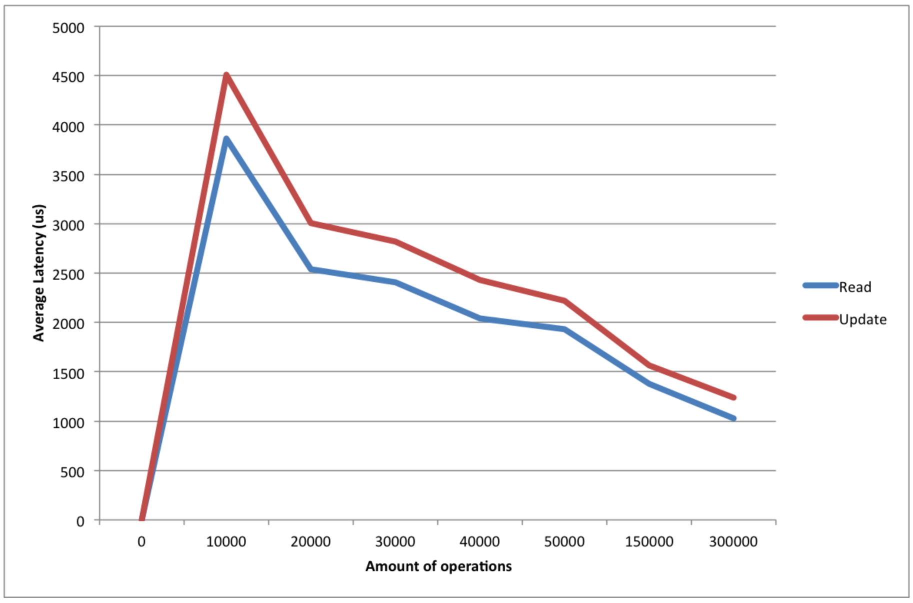


Figure 13.5: MongoDB Workload A

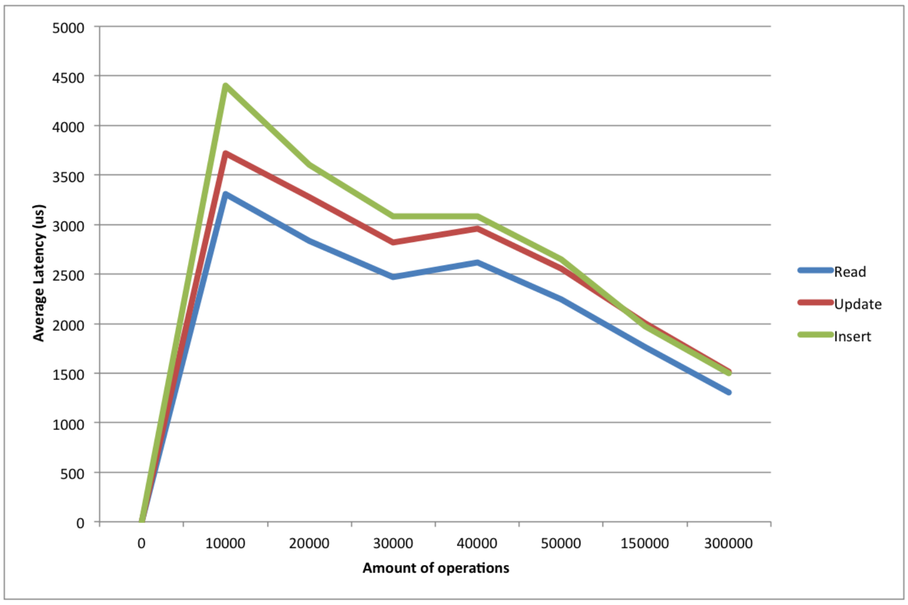


Figure 13.6: MongoDB Workload B

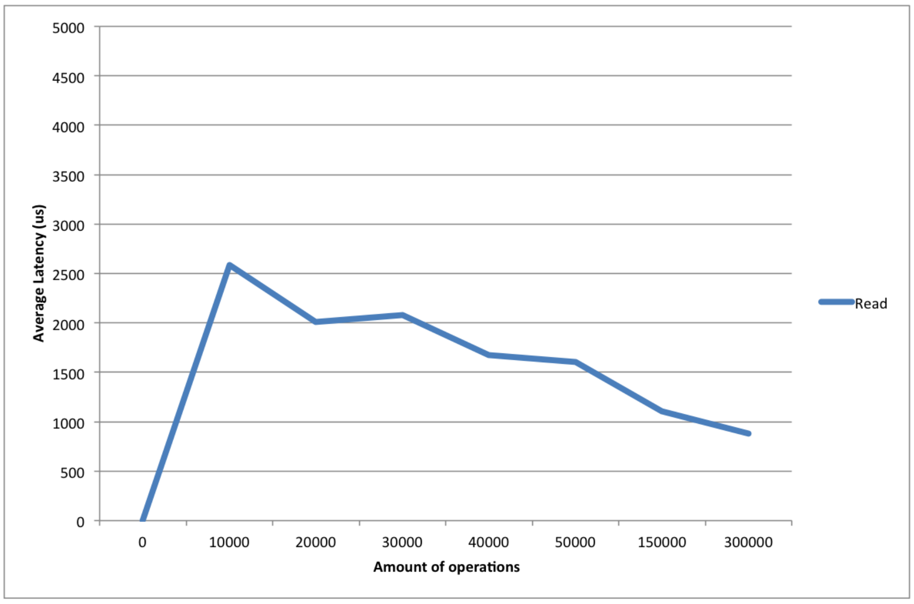


Figure 13.7: MongoDB Workload C

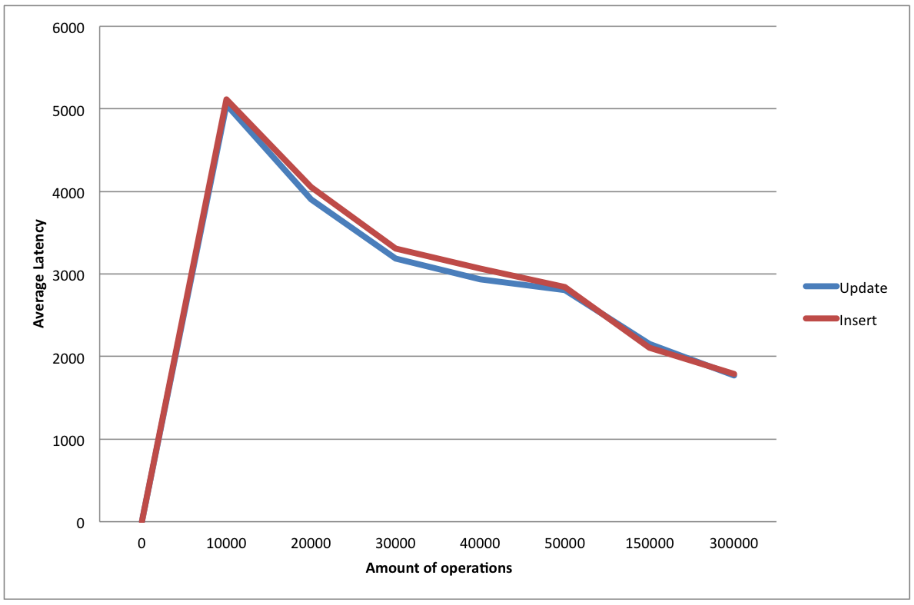


Figure 13.8: MongoDB Workload D

13.12 Couchbase

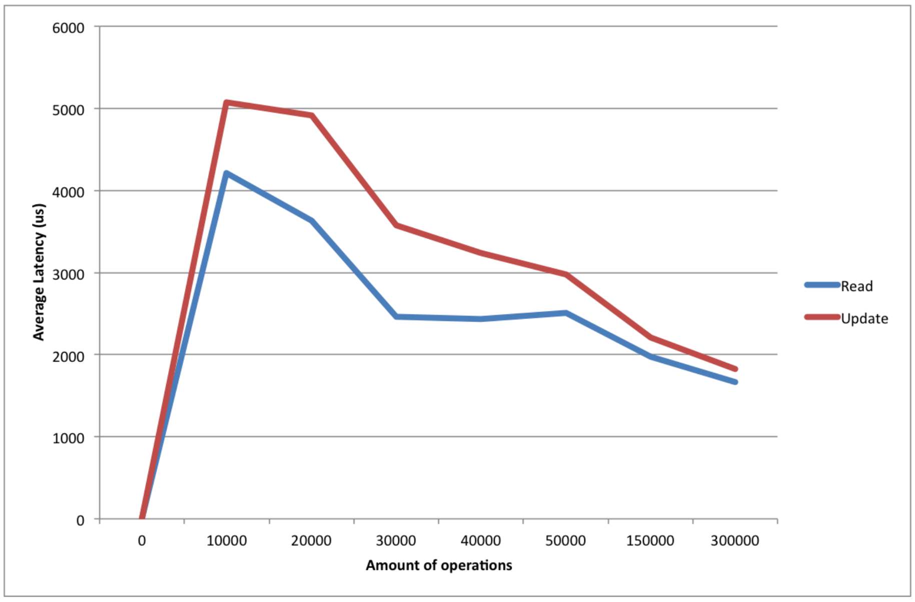


Figure 13.9: Couchbase Workload A

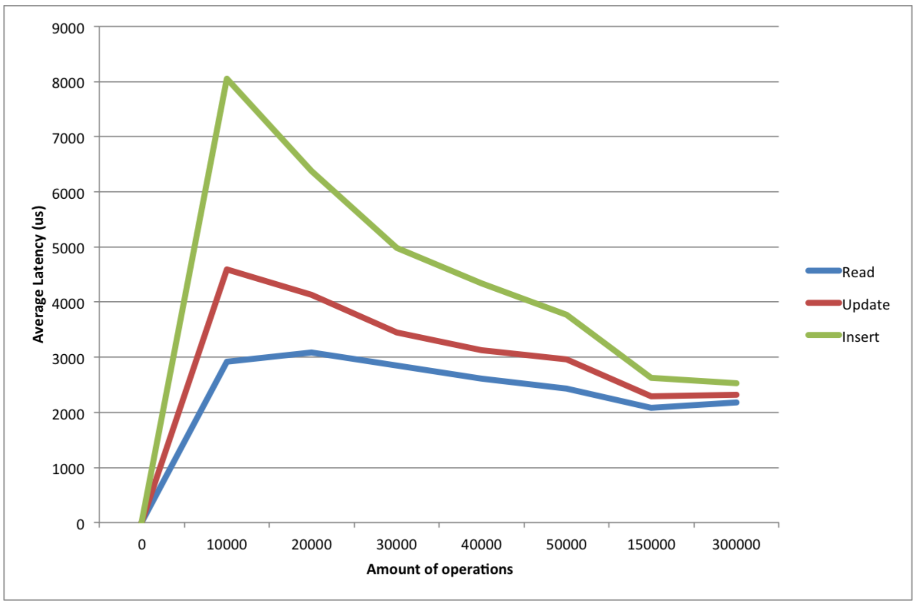


Figure 13.10: Couchbase Workload B

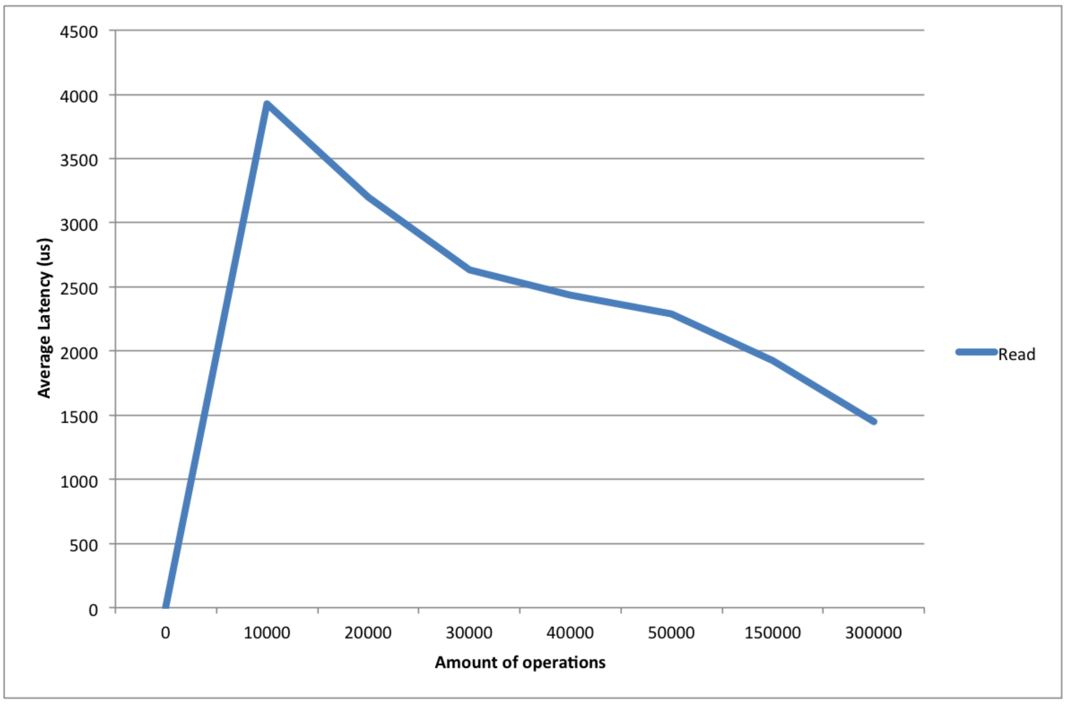


Figure 13.11: Couchbase Workload C

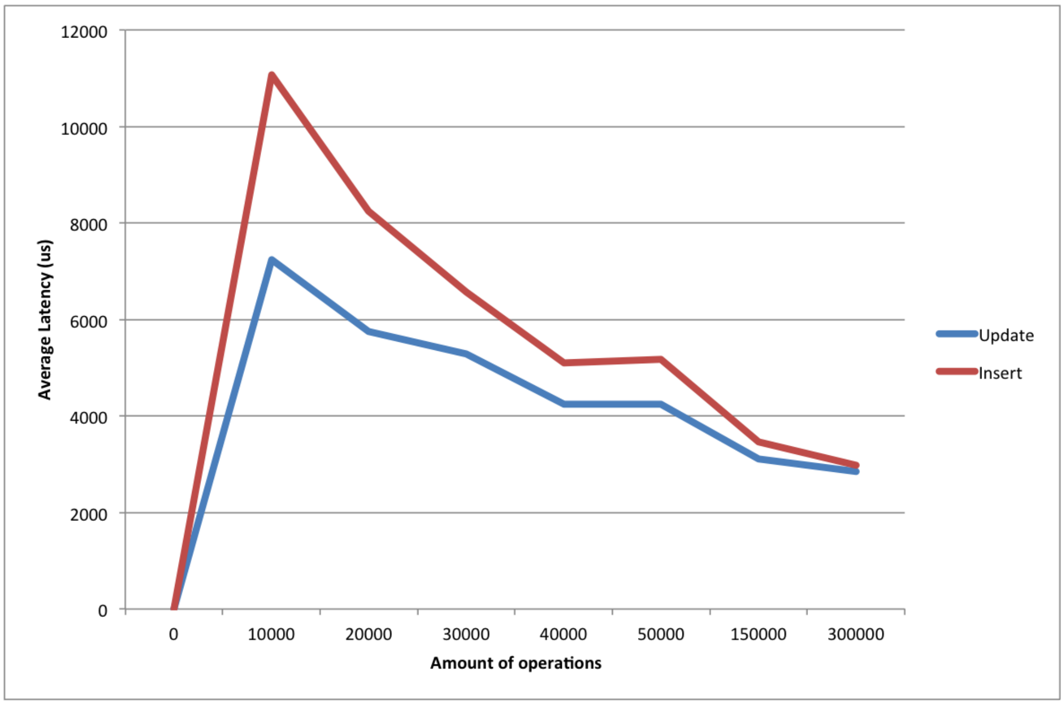


Figure 13.12: Couchbase Workload D

13.13 Cassandra

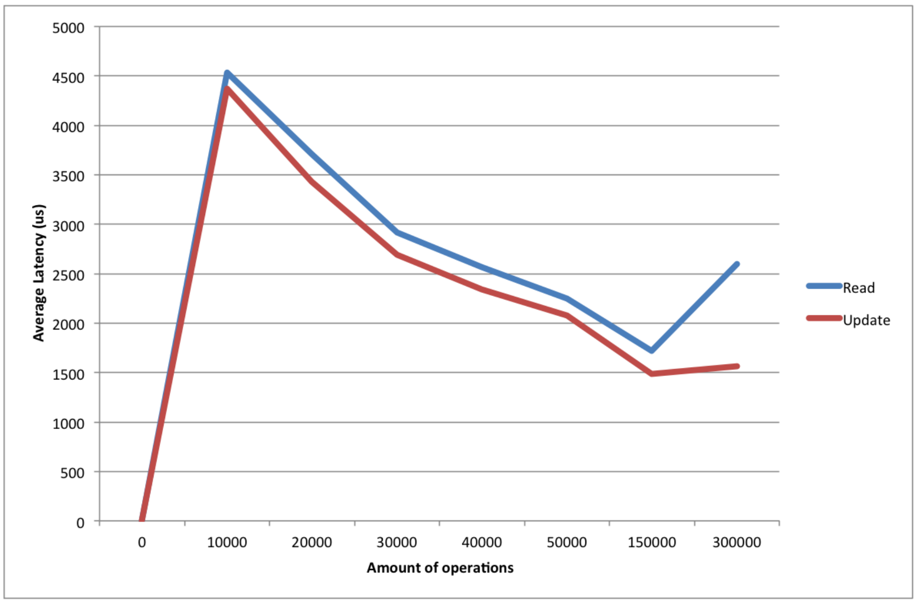


Figure 13.13: Cassandra Workload A

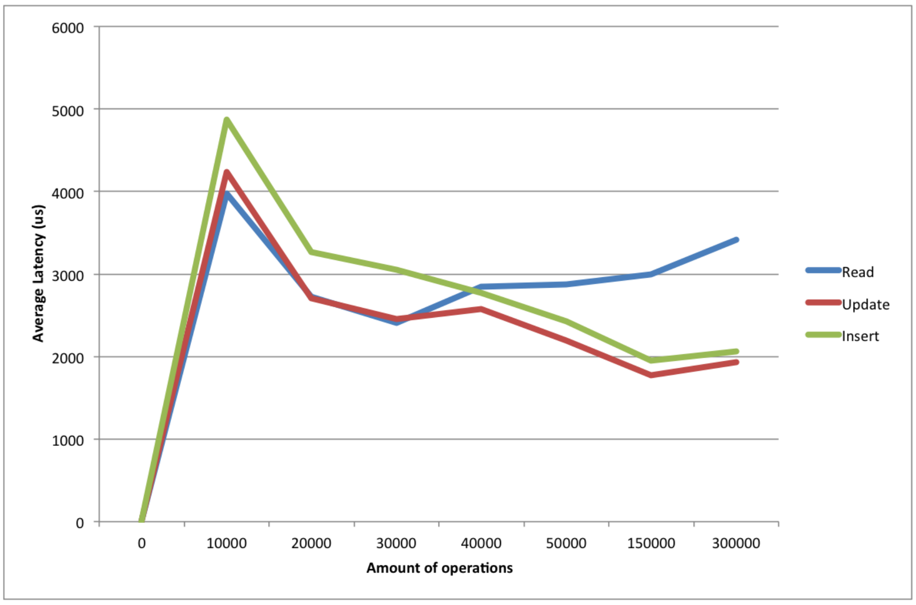


Figure 13.14: Cassandra Workload B

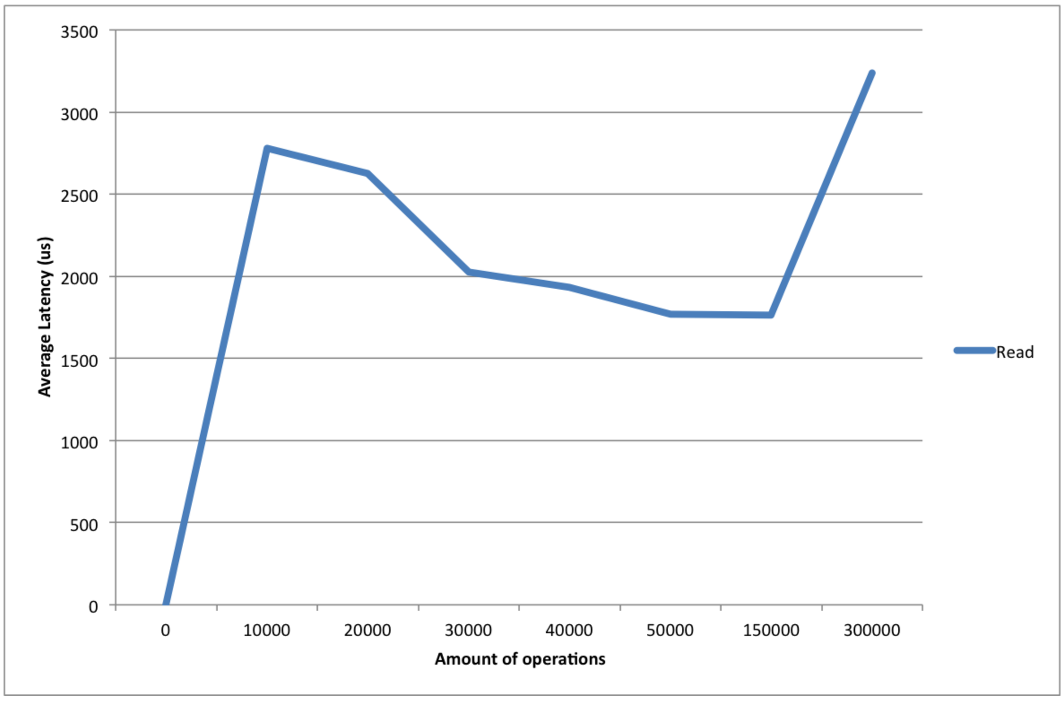


Figure 13.15: Cassandra Workload C

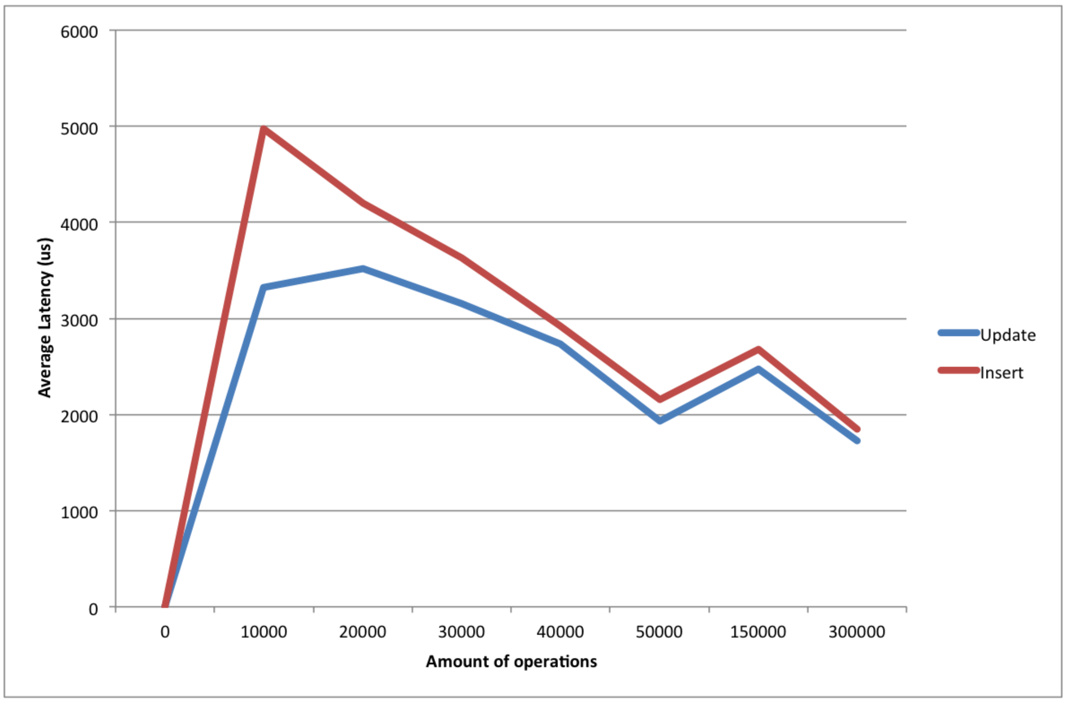


Figure 13.16: Cassandra Workload D

Bibliography

[Not, 2008a] (2008a). Case tools and their effect on software quality - the repository. http://www.cs.nott.ac.uk/~cah/G53QAT/Report08/mxr06u%20-%20WebPage/The%20repository.html.

[Not, 2008b] (2008b). Types of case tools. http://www.cs.nott.ac.uk/~cah/G53QAT/Report08/mxr06u%20-%20WebPage/Types%20of%20CASE%20tools.html.

[Cou, 2011] (2011). Nosql companies couchone and membase merge to form couchbase. http://www.couchbase.com/press-releases/membase-couchone-merge.

[Moc, 2013] (2013). Mockup builder. http://mockupbuilder.com.

[EPF, 2014] (2014). Eclipe (epf). https://eclipse.org/epf/.

[Ama, 2015] (2015). Amazon.com. http://www.amazon.com.

[App, 2015] (2015). Appswatch. http://www.nrgglobal.com/application-testing-tools/performance-testing-and-monitoring-appswatch.

[DB-, 2015] (2015). Db-main. http://www.db-main.be.

[Dox, 2015] (2015). Doxygen. http://www.stack.nl/~dimitri/doxygen/.

[Ecl, 2015] (2015). Eclipse (ide). https://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr2.

[Gan, 2015] (2015). Ganttproject. http://www.ganttproject.biz.

[His, 2015] (2015). History and timeline. http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-corporateTimeline.

[Abadi et al., 2009] Abadi, D. J., Boncz, P. A. et Harizopoulos, S. (2009). Column-oriented database systems. VLDB.

[Baik, 2000] Baik, J. (2000). The effects of case tools on software development effort. Mémoire de D.E.A., University of Southern California.

[Basho, 2015] Basho (2015). Riak docs. http://docs.basho.com/riak/latest/.

[Blakeman, 2002] Blakeman, J. (2002). Benchmarking: Definitions and overview.

[Brewer et Eric, 2000] Brewer, A. et Eric, D. (2000). Podc keynote.

[Bryden,] Bryden, J. Guide to document databases – nosql explained. `http://nosqlguide.com/document-store/nosql-databases-explained-document-databases/`.

[Cattell, 2010] Cattell, R. (2010). Scalable sql and nosql data stores. SIGMOD Record, 39(4):12–24.

[Chang et al., 2006] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. et Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. OSDI.

[Cleve, 2009] Cleve (2009). Program Analysis and Transformation for Data-Intensive System Evolution. Thèse de doctorat, Facultés Universitaires Notre-Dame de la Paix.

[Cleve, 2015] Cleve (2015). Data-intensive system evolution. part of "Information Systems Evolution" course at University of Namur.

[Clève et al., 2010] Clève, Brogneaux et Hainaut (2010). A conceptual approach to database applications evolution.

[Cleve et Hainaut, 2006] Cleve et Hainaut (2006). Co-transformations in database applications evolution.

[Clève, 2009] Clève, A. (2009). Program analysis and transformation for data-intensive system evolution. Mémoire de D.E.A., University of Namur.

[Clover, 2015] Clover (2015). Cloveretl product overview. `http://www.cloveretl.com/products/`.

[Codd, 1990] Codd, E. F. (1990). The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc.

[Cooper et al., 2010] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. et Sears, R. (2010). Benchmarking cloud serving systems with ycsb. ACM symposium on Cloud Computing, pages 143–154.

[Corporation, 2015] Corporation, I. (2015). Informatica data integration powercenter overview. `https://www.informatica.com/products/data-integration/powercenter/`.

[Couchbase,] Couchbase. Couchbase: Eventual consistency. `http://guide.couchdb.org/draft/consistency.html#local`.

[Couchbase, 2015] Couchbase (2015). Couchbase: Introduction. `http://docs.couchbase.com/admin/admin/Couchbase-intro.html`.

[Datastax, 2013] Datastax (2013). Introduction to apache cassandra.

[Datomic, 2015a] Datomic (2015a). Datomic architecture. `http://docs.datomic.com/architecture.html`.

[Datomic, 2015b] Datomic (2015b). Datomic overview. `http://www.datomic.com/overview.html`.

[Datomic, 2015c] Datomic (2015c). Datomic schema. `http://docs.datomic.com/schema.html`.

[Datomic, 2015d] Datomic (2015d). Datomic tutorial. `http://docs.datomic.com/tutorial.html`.

[DB, 2015] DB, F. (2015). The cap theorem. `https://foundationdb.com/key-value-store/white-papers/the-cap-theorem`.

[DeCandia et al., 2007a] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. et Vogels, W. (2007a). Dynamo: amazon's highly available key-value store. In ACM SIGOPS Operating Systems Review, volume 41, pages 205–220. ACM.

[DeCandia et al., 2007b] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. et Vogels, W. (2007b). Dynamo: Amazon's highly available key-value store. SOSP, pages 205–220.

[Devart, 2013] Devart (2013). Choosing the right way of migrating mysql databases. `https://www.devart.com/upload/choosing-the-right-way-of-migrating-mysql-databases-devart.pdf`.

[Dictionary, 2015] Dictionary (2015). Performance. `http://dictionary.reference.com/browse/performance`.

[Englebert et al., 1995] Englebert, V., Henrard, J., Hick, J.-M., Roland, D. et Hainaut, J.-L. (1995). Db-main : un atelier d'ingénierie de bases de données.

[Gajendran, 2011] Gajendran, S. K. (2011). A survey on nosql databases.

[Gosselé, 2014] Gosselé, B. (2014). A survey on nosql and newsql datastores.

[Gray et al., 1981] Gray, J. et al. (1981). The transaction concept: Virtues and limitations. In VLDB, volume 81, pages 144–154.

[Grolinger et al., 2013] Grolinger, K., Higashino, W. A., Tiwari, A. et Capretz, M. A. (2013). Data management in cloud environments: Nosql and newsql data stores. Journal of Cloud Computing.

[Hainaut et al., 2008] Hainaut, Cleve, Henrard et Hick (2008). Migration of Legacy Information Systems, in Software Evolution.

[Hainaut, 1996] Hainaut, J.-L. (1996). Specification preservation in schema transformations?application to semantics and statistics. Data & Knowledge Engineering, 19(2):99–134.

[Hainaut, 2006] Hainaut, J.-L. (2006). The transformational approach to database engineering. In Generative and transformational techniques in software engineering, pages 95–143. Springer.

[Hainaut, 2012] Hainaut, J.-L. (2012). Bases de données. Concepts, utilisation et développement. Dunod.

[Hainaut, 2015] Hainaut, J.-L. (2015). Bases de données. Concepts, utilisation et développement. Dunod, third édition.

[Henry et al., 2005] Henry, Hoon, Hwang, Lee et DeVore (2005). Engineering trade study: Extract, transform, load tools for data migration.

[Hickey, 2013] Hickey, R. (2013). The datomic information model. `http://www.infoq.com/articles/Datomic-Information-Model`.

[IBM, 2015] IBM (2015). What is big data? `http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html`.

[Inc., 2015] Inc., T. S. (2015). Big data business intelligence overview. `http://www.jaspersoft.com/big-data-business-intelligence-instant/`.

[ISTQB, 2015] ISTQB (2015). What is waterfall model- advantages, disadvantages and when to use it? `http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/`.

[J., 2008] J., M. (2008). Should you use an etl tool? `http://www.kimballgroup.com/2008/04/should-you-use-an-etl-tool/`.

[Jarzabek et Huand, 1998] Jarzabek, S. et Huand, R. (1998). The case for user-centered case tools. Communications of the ACM, 41(8):93 – 99.

[Jatana et al., 2012] Jatana, N., Puri, S., Ahuja, M., Kathurita, I. et Gosain, D. (2012). A survey and comparison of relational and non-relational database. International Journal of Engineering Research and Technology, 1(6):1–5.

[Jovanovic, 2015] JOVANOVIC (2015). Comparison of etl operations through selected etl tools. `http://www.essi.upc.edu/~petar/etl-taxonomy.html/`.

[Kemerer, 1992] KEMERER, C. F. (1992). How the learning lurve affects case tool adoption. IEEE Software, pages 23 – 28.

[Klaus, 2009] KLAUS (2009). Towards the industrialization of data migration: Concepts and patterns for standard software implementation projects.

[Lakshman et Malik, 2008] LAKSHMAN, A. et MALIK, P. (2008). Cassandra - a decentralized structured storage system.

[Lamllari, 2013] LAMLLARI, R. (2013). Extending a methodology for migration of the database layer to the cloud considering relational database schema migration to nosql.

[Laney, 2001] LANEY, D. (2001). 3d data management: Controlling data volume, velocity and variety. META Group Research Note, 6.

[Loucopoulos, 1995] LOUCOPOULOS, P. (1995). C.a.s.e. technology. In System Requirements Engineering, chapitre 6. International Series in Software Engineering.

[Memcached, 2015] MEMCACHED (2015). About memcached. `http://memcached.org/about`.

[Merise, 2014] MERISE (2014). Definition of an identifiant. `http://www.commentcamarche.net/contents/659-merise-modele-conceptuel-des-donnees`.

[MongoDB, 2015] MONGODB (2015). Big data explained. `https://www.mongodb.com/big-data-explained`.

[mongoDB, 2015] MONGODB (2015). Mongodb : Index. `http://docs.mongodb.org/manual/core/indexes-introduction/`.

[Mullins, 2015] MULLINS, C. S. (2015). The definition of database performance. `https://datatechnologytoday.wordpress.com/2011/06/03/the-definition-of-database-performance/`.

[Nayak et al., 2013] NAYAK, A., POIRYA, A. et POOJARY, D. (2013). Type of nosql databases and its comparison with relational databases. International Journal of applied Information systems, 5(4):16–19.

[Needham, 2013] NEEDHAM, J. (2013). Disruptive possibilities: how big data changes everything. " O'Reilly Media, Inc.".

[Oscar, 2015] OSCAR (2015). Oscar system. `http://oscar-emr.com/`.

[Paygude et Devale, 2011] PAYGUDE et DEVALE (2011). Automated data validation testing tool for data migration quality assurance.

[Pentaho, 2015] PENTAHO (2015). Pentaho data integration. `http://www.pentaho.com/product/data-integration/`.

[Petruska, 2011] PETRUSKA (2011). Advantages and limtations of case tools. Technology and Mathematics.

[Robinson et al., 2013] ROBINSON, I., WEBBER, J. et EIFREM, E. (2013). Graph Databases. O'Reilly.

[Rodriguez et al., 2012] RODRIGUEZ, LAWSON, MOLINA et GUTIERREZ (2012). Data warehousing tool evaluation – etl focused.

[Rouse, 2007a] ROUSE, M. (2007a). Quality assurance (qa). `http://searchsoftwarequality.techtarget.com/definition/quality-assurance`.

[Rouse, 2007b] ROUSE, M. (2007b). Waterfall model. `http://searchsoftwarequality.techtarget.com/definition/waterfall-model`.

[Rouse, 2015a] ROUSE, M. (2015a). Case (computer-aided software engineering) definition. `http://searchmanufacturingerp.techtarget.com/definition/CASE-computer-aided-software-engineering`.

[Rouse, 2015b] ROUSE, M. (2015b). Web 2.0 - definition. `http://whatis.techtarget.com/definition/Web-20-or-Web-2`.

[Saavedra et Smith,] SAAVEDRA, R. H. et SMITH, A. J. Analysis of benchmark characteristics and benchmark performance prediction.

[Services, 2012] SERVICES, A. W. (2012). What is amazon dynamodb? - developer guide. `http://docs.aws.amazon.com/fr_fr/amazondynamodb/latest/developerguide/Introduction.html`.

[Services, 2015a] SERVICES, A. W. (2015a). Amazon ec2. `https://aws.amazon.com/ec2/?nc1=h_ls`.

[Services, 2015b] SERVICES, A. W. (2015b). Amazon s3. `https://aws.amazon.com/s3/?nc1=h_ls`.

[Services, 2015c] SERVICES, A. W. (2015c). What is cloud computing? `https://aws.amazon.com/fr/what-is-cloud-computing/?nc2=h_l2_cc`.

[Shweta et al., 2014] SHWETA, PRATIKSHA et BENDRE (2014). Data migration in heterogeneous databases (etl).

[Simitsis et Vassiliadis, 2003] Simitsis et Vassiliadis (2003). A methodology for the conceptual modeling of etl processes.

[Sommerville, 2008] Sommerville, I. (2008). Tools and environments. `http://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/CASE/ToolsWBEnv.htm`.

[SQLpro, 2015] SQLpro (2015). Base de données et performances... petites tables et tables obèses ! `http://blog.developpez.com/sqlpro/p10070/langage-sql-norme/base_de_donnees_et_performances_petites`.

[Statista, 2015] Statista (2015). Number of monthly active facebook users worldwide as of 2nd quarter 2015). `http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/`.

[Steemann, 2012] Steemann, J. (2012). Column-oriented database. NoSQL Cologne UG.

[Stonebraker et al., 2005] Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N. et Zdonik, S. (2005). C-store: A column-oriented dbms. VLDB.

[Stonebraker et Hellerstein, 1999] Stonebraker, M. et Hellerstein, J. M. (1999). Readings in Database Systems.

[Strauch et al., 2011] Strauch, C., Sites, U.-L. S. et Kriha, W. (2011). Nosql databases.

[Subharthi, 2008] Subharthi, P. (2008). Database systems performance evaluation techniques.

[Talend, 2015] Talend (2015). Talend products overview. `https://www.talend.com/products/`.

[Thiran et al., 2005] Thiran, Hainaut et Houben (2005). Database wrappers development: Towards automatic generation.

[Thiran et al., 2006] Thiran, Hainaut et Houben (2006). Wrapper-based evolution of legacy information systems.

[Thota,] Thota. Etl tools comparison matrix. `http://www.quickdatainsights.com/data-integration-tools-comparison-2/etl-tools-comparison-matrix/`.

[Tutorialspoint, 2015] Tutorialspoint (2015). Software case tools overview. `http://www.tutorialspoint.com/software_engineering/case_tools_overview.htm`.

[Tziovara et al., 2007] Tziovara, Vassiliadis et Simitsis (2007). Deciding the physical implementation of etl workflows.

[V, 2014] V, M. (2014). Comparative study of nosql document, column store databases and evaluation of cassandra. International Journal of Database Management Systems, 6(4):11–26.

[Vassiliadis et al., 2007] Vassiliadis, Simitsis, Gerogantas, Terrovitis et Skiadopoulos (2007). A generic and customizable framework for the design of etl scenarios.

[Veronika Abramova, 2014] Veronika Abramova, Jorge Bernardino, P. F. (2014). Which nosql database? a performance overview. Research Online Publishing, 1(2):17–24.

[Webopedia, 2015] Webopedia (2015). Data modeling. `http://www.webopedia.com/TERM/D/data_modeling.html`.

[WhatIs, 2015a] WhatIs (2015a). First-orger logic. `http://whatis.techtarget.com/definition/first-order-logic`.

[WhatIs, 2015b] WhatIs (2015b). Performance. `http://whatis.techtarget.com/definition/performance`.

[Xerox, 2015] Xerox (2015). Xerox website. `http://www.xerox.ca/about-xerox/75th-anniversary/enca.html`.

[Yao et Hevner, 1984] Yao, S. B. et Hevner, A. R. (1984). A guide to performance evaluation of database systems.

[Yousuf et Rizvi, 2011] Yousuf et Rizvi (2011). A comparative study of etl tools.