UNIVERSITÉ LIBRE DE BRUXELLES

Academic year 2015 - 2016

INFO-H-415 : Project Report – eXist-db

Raphael DUHEN

Abstract

This main goal of this report is to give a detailed study of XML databases and their use in the applying of eXist-db. This report will first describe the theoy related to XML databases and the theory related to relational databases, then it will look in detail in the creation, the use and the managmenent of eXist-db. This second part will take a look at the flexibility of eXist-db (how easy it is to migrate from SQL, how easy it is tomigrate to SQL, ...). Finally, the report will present several benchmarks, allowing a comparison between a relational database (MySQL in this case) and an XML database.

Keywords : databases, XML databases, relational databases, MySQL, eXist-db.

Contents

1 Introduction **3**
1.1 Introduction . . . 3
1.2 Introduction to relational databases . . . 4
1.2.1 General concept . . . 4
1.2.2 Limitation . . . 5
1.3 Introduction to XML databases . . . 7

2 Integration of eXist-db **10**
2.1 Introduction . . . 10
2.2 Import of a relational database in eXist-db . . . 10
2.2.1 From MySQL to XML . . . 12
2.2.2 From XML to eXist-db . . . 14
2.2.3 Conclusion : MySQL to XML database . . . 14
2.3 Import of a XML database in MySQL . . . 15
2.3.1 XML file to MySQL . . . 15
2.3.2 Conclusion : XML database to MySQL . . . 17
2.4 eXist-db and web semantic . . . 18
2.4.1 The web semantic . . . 18
2.4.2 eXist-db support . . . 18
2.4.3 Not supported by eXist-db . . . 19

3 Benchmarks **20**
3.1 Creation of the databases . . . 21
3.1.1 Parsing of the file . . . 21
3.1.2 Creation of the MySQL database . . . 21
3.1.3 Creation of the eXist-db database . . . 23
3.2 First Query : search an edge in function of its ID . . . 23
3.2.1 SQL Request . . . 23
3.2.2 XQuery . . . 23
3.2.3 Results . . . 24
3.3 Second Query : search all reachable destination (level 2) . . . 25
3.3.1 SQL Request . . . 25
3.3.2 XQuery . . . 25
3.3.3 Results . . . 25
3.4 Third Query : search all reachable destination (level 3) . . . 27
3.4.1 SQL Request . . . 27
3.4.2 XQuery . . . 27
3.5 Conclusion . . . 27

4 Conclusion **29**

A XML file generated by the mysqldump command **31**

B Python script for parsing from Google file to XML format **35**

C Benchmark code **37**

C.1 Main file . . . 37

C.2 Mysql file . . . 39

C.3 eXist file . . . 41

C.4 Benchmark program output . . . 43

1 Introduction

1.1 Introduction

As part of the course of INFO-H-415 - Advanced database, we have been asked to choose a database type and to develop a report allowing to understand its opeation and its particularity. In this context, I'll describe the advantages and the disadvantages of the chosen database in comparison with another type of database.

My choice was the XML database, and I'll use eXist-db in order to make the comparison with the other type of database. Here are the main reason why I've chosen eXist-db over other XML database :

- It's a NoSQL database, and one important thing as NoSQL database that supports XML files and JSon files, which are two types of data format files widely used in the world.
- It's an open source project, meaning that that the code is free, making it easy to get and to make tests on at low costs.
- It's pretty well documented on its official website, making the learning quick and easy
- It supports a lot of the XML databases technologies
- It supports a lot of API's

Now that I'vs described my motivations and my choice, here's how the report will be divided :

1. Introduction to the two database types I'll compare
2. Further study eXist-db
3. Actual comparison with benchmarks, and the different used queries.

1.2 Introduction to relational databases

1.2.1 General concept

This kind of database has been wildely used during the 80's and is still wildely used today. MySQL, PostgreSQL and SQLite are such type of databases. This type odf database use SQL to make the queries. The paradigm of this type of database is the relational model : all datas are represented in terms of tuples, grouped into relations. A good way to represent this relation is by using a table :

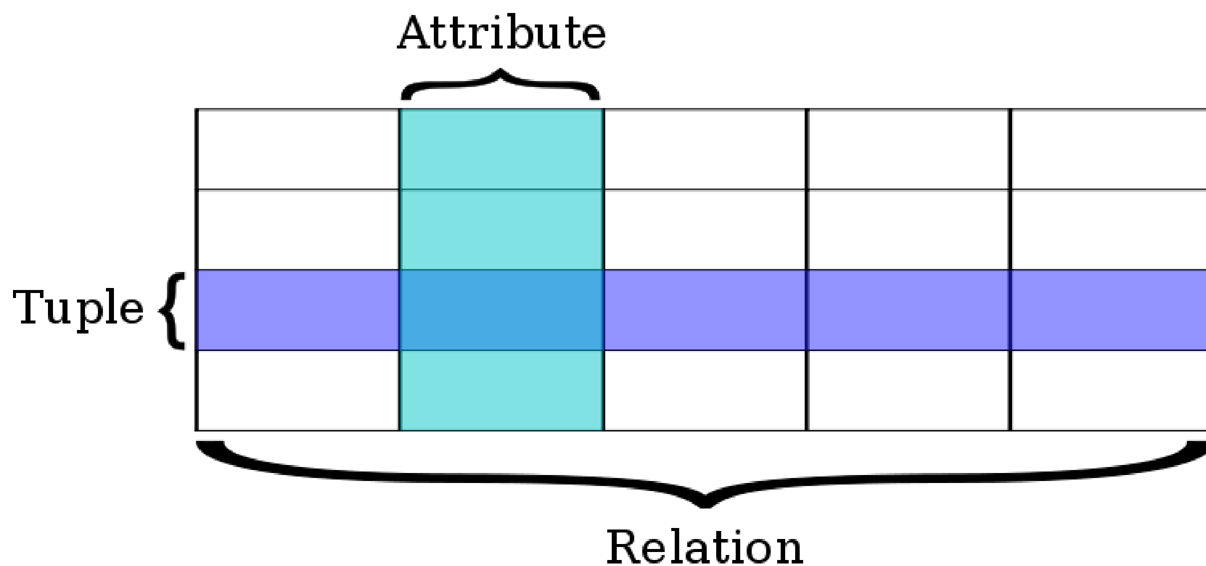


Figure 1.1: Exemple of relational table

Or, in a more concrete way, here's an example of how we could represent a set of computers with their main characteristics :

ID	Processor	RAM	Storage	Graphics card chip
1	intel i7	8Go	1To	GeForce 950
2	intel i3	4Go	360Go	Radeon R9
3	AMD K8	16Go	2To	GeForce 980

Table 1.1: Exemple of relational table

The relation presented here could be resumed as a composition relation and a sentence that could resume this composition relation is, for example : "the computer 1 is composed by an intel i7, 8Go of RAM, 1To of 1To storage capacity and a GeForce 950 Graphic card chip". Note that the ID needs to be unique in the table because it represents the computer in an unique way.

1.2.2 Limitation

The relational model has shown some weaknesses in the recent years, because, mainly of two different causes :

- The complexity of realising some databases, even some common one. To demonstrate this complexity, let's takeour previous example an enrich it with the possibility of detailled graphic card description and the possibility for a PC to have multiple graphic card. We will then have a special table for the PC which we already have, a table for the graphic card specification and a table for the correspondance PC/Graphic card.

ID	Processor	RAM	Storage
1	intel i7	8Go	1To
2	intel i3	4Go	360Go
3	AMD K8	16Go	2To

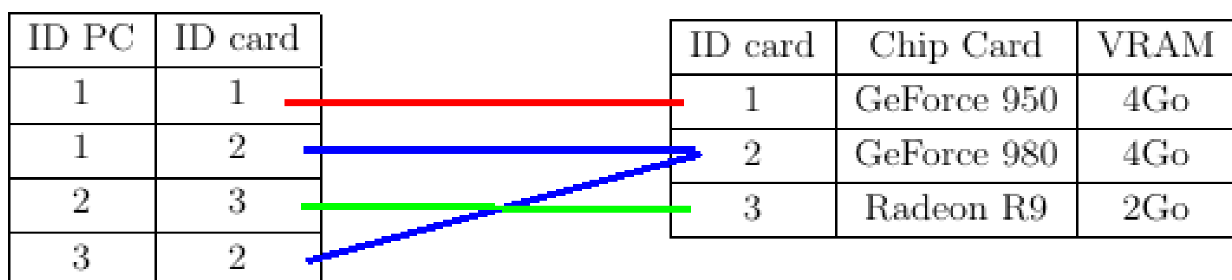
ID PC	ID card
1	1
1	2
2	3
3	2

ID card	Chip Card	VRAM
1	GeForce 950	4Go
2	GeForce 980	4Go
3	Radeon R9	2Go

Table 1.2: From left to right : PC table, correspondance PC/Graphic card table and Graphic card table

  This is still an easy example, but the problematic of interactivity of datas in the recent years has become a real problem, leading to a lot of time spent on mapping code and relational databases.

- The complexity of several operations combined with the still growing amount of data stored. Several queries for example multiple join operations (combination of two tables, see below), even though really optimised, can lead to a long processing time in table with a lot of data.



ID PC	ID card
1	1
1	2
2	3
3	2

ID card	Chip Card	VRAM
1	GeForce 950	4Go
2	GeForce 980	4Go
3	Radeon R9	2Go

Figure 1.2: Example of an inner join operation on the ID card from the correspondance table and the ID card from the graphic card table

ID PC	ID card	ID card	Chip card	VRAM
1	1	1	GeForce 950	4Go
1	2	2	GeForce 980	4Go
2	3	3	Radeon R9	2Go
3	2	2	GeForce 980	4Go

Table 1.3: Result of the previous inner join operation

Those problems have led to the development of new database types that work with relational databases, or replace then totally.

1.3 Introduction to XML databases

The XML databases is one of the response found to the problem of complexity of implementation by allowing data to be specified, and sometimes stored, in XML format. XML databases are thus a type of document-oriented databases, which implies that XML databases are a category of NoSQL database.

Here's an example of such a database in the XML format [1] :

Code 1.1: XML database example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
  <article mdate="2011-01-11" key="journals/acta/Saxena96">
    <author>Sanjeev Saxena</author>
    <title>Parallel Integer Sorting and Simulation Amongst CRCW Models.</title>
    <year>1996</year>
  </article>
  <article mdate="2011-01-11" key="journals/acta/Simon83">
    <author>Hans-Ulrich Simon</author>
    <title>Pattern Matching in Trees and Nets.</title>
    <year>1983</year>
  </article>
  <article mdate="2011-01-11" key="journals/acta/GoodmanS83">
    <author>Nathan Goodman</author>
    <author>Oded Shmueli</author>
    <title>NP-complete Problems Simplified on Tree Schemas.</title>
    <year>1983</year>
  </article>
</dblp>
```

The data structure produced by this XML will be a tree, each node of the tree being attributes or elements of the XML file, and the attributes of the XML elements being the leaves of the tree.

The role of the database will be to browse the tree in order to find the information relative to the query.

[1]Reduced version of the dblp database : http://dblp.uni-trier.de/

The corresponding tree to the previous XML File is the following (for readability reason, the attributes are represented by a @ followed by the name of their field, instead of their real value) :

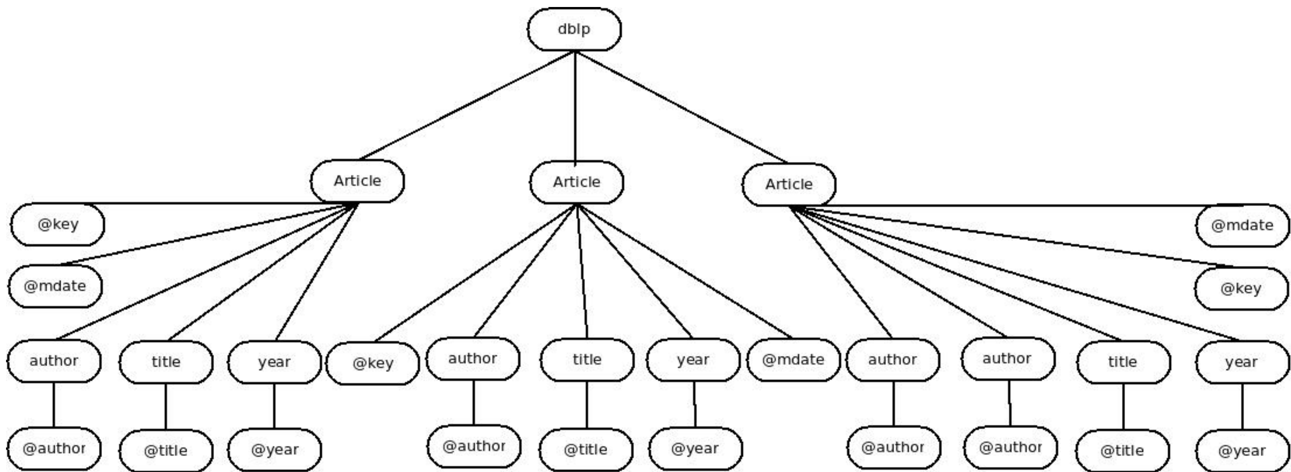


Figure 1.3: Tree corresponding to the previous XML file

We can directly notice that this type of database solves the problem of underlying the difficulty of database creation desribed section 1.2.2. Indeed, there's no problem of multiple table for a relation anymore.

For example, here's one of the possible result our previous database described in section 1.2.2 could look like if we translated it in the XML format, if we suppose that the database name is pcs :

Code 1.2: Possible XML result

```
<pcs>
  <pc id="1">
    <processor>intel i7</processor>
    <RAM>8Go</RAM>
    <storage>1To</storage>
    <graphic_card>
      <chip_card>GeForce 950</chip_card>
      <VRAM>4Go</VRAM>
    </graphic_card>
    <graphic_card>
      <chip_card>GeForce 980</chip_card>
      <VRAM>4Go</VRAM>
    </graphic_card>
  </pc>
  <pc id="2">
    <processor>intel i3</processor>
    <RAM>4Go</RAM>
```

```
    <storage>360Go</storage>
    <graphic_card>
      <chip_card>Radeon R9</chip_card>
      <VRAM>2Go</VRAM>
    </graphic_card>
  </pc>
  <pc id="3">
    <processor>AMD K8</processor>
    <RAM>16Go</RAM>
    <storage>2To</storage>
    <graphic_card>
      <chip_card>GeForce 980</chip_card>
      <VRAM>4Go</VRAM>
    </graphic_card>
  </pc>
</pcs>
```

2 Integration of eXist-db

2.1 Introduction

This section will have as main goal to take a closer look at the flexibility of the eXist-db database. In this context, I will detail :

- The complexity relative to the migration from a relational database to a XML database.
- The complexity relative to the migration from a XML database to a relational database.
- The langages allowing the use eXist-db.

2.2 Import of a relational database in eXist-db

As explained earlier, relational databases stores data under the form of tables.

In our project, I will use MySQL database to make the comparisons. MySQL uses the SQL language to make its queries. SQL (Standard Query Language) is the most wildely used language for the exploitation of databases, which implies that it is supported by most of the DBMS at this date.

In order to make a migration from a relational database (MySQL) to XML database (eXist), we will need to create a database. In order to make this migration possible on as much relational database, we will need an example as wide as possible, containing the different type of relationships a relational database can offer.

Those different relations are :

- one-to-one relation
- one-to-many
- many-to-many

Here's the database created in order to implement those realtionships :

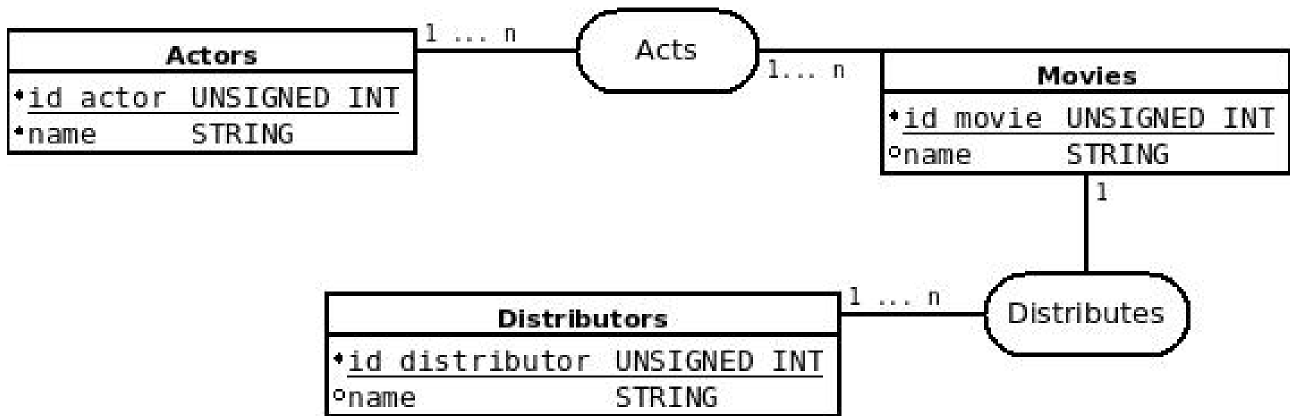


Figure 2.1: Diagram corresponding to our example of relational database

Here are the relationship in more details :

- many-to-many relation :
  - An actor acts in one or several movies
  - A movie has one or several actors acting in it
- one-to-many relation :
  - A distributor ditributes one or several movies
  - A film is distributed by one distributor

Remarks :

- We can notice that the one-to-one relation is not implemented because its treatment in a relation database is the same as a one-to-many relation, i.e by a foreign-key attribute in the relation representing the entity whose maximal cardinality in R is 1.
- In practice, the relation presented between the Distributors and the Movies could be solved by adding a column in the Movies database containing the name of the distributor. The database presented here is for demonstration purpose only.

An example of this database could be filled is the following :

id distributor	name
1	New Line Cinema
2	20th Century Fox

id actor	name
1	Elja Wood
2	Christopher Lee
3	Ewan McGregor

id relation	id actor	id movie
1	1	2
2	2	1
3	2	2
4	3	1

id movie	name	id distributor
1	Star Wars II	2
2	The Lord of the Rings	1

Table 2.1: Filled database, top-left : the Distributors table, top-right : the Actors table, bottom left : the table resulting from the many-many relation between Actors and Movies, bottom-right : the Movies table with the id distributor column resulting from the one-to-many relation between Distributors and Movies

2.2.1 From MySQL to XML

An easy way to export a MySQL database to a eXist-db database is by converting the database to a XML file, any XML file being a potential database for a XML database.

This export from MySQL to XML can be easily achieved with the following command on a UNIX system:

Code 2.1: MySQL to XML command

```
mysqldump --xml $database_name [$table_name] > $save_file_name.xml
```

or

Code 2.2: Other MySQL to XML command

```
mysqldump -X $database_name [$table_name] > $save_file_name.xml
```

Where $name are variables, and more precisely :

- $database_name is the name of the database we want to convert in XML.
- $table_name (optional) is the name of the table we want to convert in XML.
- $save_file is the name of the XML file representing the database.

The result of those commands have the following format :

Code 2.3: Mysqldump xml return format

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name=$(database name)>
  <table_structure name=$(name of the table)>
    <field Field=$(name of the column) Type=$(type of the column) Null=$("YES" /
        "NO") Key=$(is the column a key (empty if not key)" Extra=$(extra info about
        column)" Comment="" />
    <field Field=$(name of another column) ... />
    <key Table=$(name of the table) Non_unique="0" Key_name=$(type of key)
        Seq_in_index="1" Column_name=$(name of the key column) Collation="A"
        Cardinality="3" Null="" Index_type="BTREE" Comment="" Index_comment="" />
    <options Name=$(name of the table) Engine=$(engine type) Version="10"
        Row_format="Compact" Rows=$(number of rows) Avg_row_length="5461"
        Data_length="16384" Max_data_length="0" Index_length="0" Data_free="9437184"
        Auto_increment="4" Create_time=$(date and hour of creation)
        Collation="latin1_swedish_ci" Create_options="" Comment="" />
  </table_structure>
  <table_data name=$(name of the table)>
  <row>
    <field name=$(name of the first column)>$(value)</field>
    <field name=$(name of the second column)>$(value)</field>
  </row>
  <row>
    <field name=$(name of the first column)>$(value)</field>
    <field name=$(name of the second column)>$(value)</field>
  </row>
    .
    .
    .
  </table_data>
  <table_structure name=$(name of an other table)>
  .
  .
  .
</database>
```

The words between paranthesis are preceded by a "$" symbol are variables, which won't be detailled here[1].

[1] The result of the command Code 2.1 or Code 2.2 on the table 2.1 database is noted in the appendix Code A

The result is indeed a XML file, but it is much less human-readable than the XML files we have seen previously (Code 1.1 or Code 1.2).

Unfortunately there's no easy way to transcript a MySQL database in a human-readable type we have seen before. The only way is by creating scripts parsing the generated thanks to the Code 2.1 or the Code 2.2 command, and combining it. An other way would be by creating a script browsing the MySQL database itself and store the datas in a format known XML file. Those solutions are both time-consuming both by the implementation time needed for the creation of the parser and by the resources (in memory and time) required to compute the corresponding XML file.

2.2.2 From XML to eXist-db

This part will describe how to use an XML file exported from a MySQL database as described in the section 2.2.1, to create an eXist-db database.

Because the base of a XML database is a XML file, the only thing we will need to do is configure the eXist-db engine in order to make it use the XML file previously generated. This operation can be achieved with the following operation :

- Start the eXist engine
- Start the eXide
- In the file option, select "Manage"
- Import the XML file

We can now execute XQuery on the file previously generated.

Here's an example of such a command that will select all the actors name, if we suppose the imported file is in the path "/db/apps/test/moviesFromMySQL.xml" :

Code 2.4: Simple Xquery with eXist-db

```
xquery version "3.0";

doc("/db/apps/test/moviesFromMySQL.xml")//table_data[@name="Actors"]/row/field[@name="name"]
```

2.2.3 Conclusion : MySQL to XML database

The conclusion of a migration from MySQL to a XML database is that it is relatvely easy, thanks to a good use *mysqldump*. However, it is important to note that the output format of this command won't give a good human-readable XML file, and that in order to get such a file, we will need further parsing and recomposing operations.

2.3 Import of a XML database in MySQL

This part will detail how to migrate from a XML database to a MySQL database.

The part concerning how to get a XML file from a XML database is trivial, the basis of a XML database being a XML file (or a list of XML files). So we're going to skip it in order to take a closer look at the part concerning how to use a XML file in order to generate a MySQL database.

2.3.1 XML file to MySQL

The import of a XML database into MySQL can be done thanks to the *LOAD XML* command. There are several ways of importing XML files with different formats. We will first take a look at the full command, and then detail how to use it in a concrete way .

The command is :

Code 2.5: Full syntax of a Load XML command

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Here are the details of the clauses[2] :

- If we use LOW_ PRIORITY, the execution of the LOAD DATA statement is delayed until no other clients are reading from the table.
- The LOCAL keyword affects where the file is expected to be found.
- The REPLACE and IGNORE keywords control handling of input rows that duplicate existing rows on unique key values.
- CHARACTER SET specifies the encoding used
- ROWS INDENTIFIED BY <tagname> allows to change the default element considered to be equivalent to a database row (<row> by default).
- IGNORE number LINES — ROWS causes the first number rows in the XML file to be skipped.

[2]More details at https://dev.mysql.com/doc/refman/5.5/en/load-xml.html

- (field_name_or_user_var,...) is a comma-separated list of one or more XML fields or user variables. The name of a user variable used for this purpose must match the name of a field from the XML file, prefixed with @. Field names can be used to select only desired fields.
- SET allows to perform transformations on user variables before assigning the result to columns.

Those clauses allow a pretty good flexibility on the data importing. About the file format, there are 3 main types recogneized :

- Column names are the name attributes of <field> tags, and values are the contents of these tags :

Code 2.6: First accepted XML format

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

This is a look-like format from what we have seen at Code 2.3.

- Column names as attributes and column values as attribute values :

Code 2.7: First accepted XML format

```
  <row column1="value1" column2="value2" .../>
```

- Column names as tags and column values as the content of these tags :

Code 2.8: First accepted XML format

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

Although there's one big disadvantage : these commands require the MySQL database and all the structure of the database to be already done (the tables need to be created).

Here's an example of a command Code2.5 on the database Table2.1 :

Code 2.9: Example of load xml command

```
LOAD XML INFILE "MoreMovies.xml"
INTO TABLE Movie.Movies;
```

with "MoreMovies.xml" as following :

Code 2.10: MoreMovies.xml file

```
<?xml version="1.0"?>
<table_data name="Movies">
<row>
  <field name="id_movie">3</field>
  <field name="name">Snakes on a Plane</field>
  <field name="id_distributor">2</field>
</row>
<row>
  <field name="id_movie">4</field>
  <field name="name">Avatar</field>
  <field name="id_distributor">1</field>
</row>
</table_data>
```

Gives indeed the expected result :

id_ movie	name	id_distributor
1	Star Wars II	2
2	The Lors of the Rings	1
3	Snakes on a Plane	2
4	Avatar	1

Table 2.2: Example of Code 2.9 with Code 2.10 on the Movie table from database 2.1

2.3.2 Conclusion : XML database to MySQL

We can say that, once the database has been created, the filling of this database with a XML file with a close format to the one of the database is pretty quick to setup.

Otherwise, the process can be longer to adapt the XML file to make it compatible with the database structure.

2.4 eXist-db and web semantic

In this section we will talk about integration of eXist-db in applications. In order to do that, we will need to explain

- What is the standard of communication on the internet (necessary to make the communication with the database possible on the internet)
- How does eXist-db supports this standard
- What are the elements not supported yet

2.4.1 The web semantic

The web semantic is an enterprise trying to méthodes for the exchange of datas on the internet. It tries to find an answer to the following problem : how to organize datas so that the content of a web, originally designed for humans, can also be treated by computers ?
W3C will then try to structure informations so that both humans and computers can interptrepret the datas online.

2.4.2 eXist-db support

As seen before, W3C implies a lot of standards on how to be able to browse datas.
Here are the W3C standards supported by W3C :

- XPath : query language for selecting nodes from an XML document.
- XQuery : query and functional programming language that queries and transforms collections of structured and unstructured data, usually in the form of XML, text and with vendor-specific extensions for other data formats (JSON, binary, etc.)
- XSLT : language for transforming XML documents into other XML documents,[1] or other formats such as HTML for web pages, plain text or into XSL Formatting Objects
- XSL-FO : markup language for XML document formatting which is most often used to generate PDF
- webDAV : extension of HTTP that allows clients to perform remote Web content authoring operations.
- SOAP : protocol specification for exchanging structured information in the implementation of web services in computer networks.
- XInclude : generic mechanism for merging XML documents
- XProc : W3C Recommendation to define an XML transformation language to define XML Pipelines.

We can see that eXist-db provides a lot of W3C features making it a good engine to use when creating a web application.

2.4.3 Not supported by eXist-db

Although the strong support of W3C standards, there are several remarks to be done about its support :

- EXpath is not supported by eXist-db.
- XQuery 1.0 is supported at 99.4% by eXist-db, meanwhile XQuery 3.0 is not fully supported (no data available on how much is supported)

3 Benchmarks

In this chapter, we will lookat some benchmarks to be able to make a good comparison between MySQL and eXist-db in term of performances.

In order to compare the performances of those two database engines, we need a relatively big amount of data that we will store in the two types ofdatabase wewant to study the performances. In order to do that, I've downloaded a file representing a graph used by Google during Google Programming Contest of 2002[1].
This file contains 875713 nodes and 5105039 nodes. We won't use all the datas provided, we will only use a certain amount

The file format is described below :

Code 3.1: Format of the file used for the benchmarking

```
# comment
origin_node destination_node
origin_node destination_node
.
.
.
```

The benchmarks will be performed on a computer running under Kubuntu with the following hardware :

- Processor : Intel Core i7 4790 (3.6 GHz) - 4 cores - 8 threads - 8 MB cache - 64 bits , with a
- RAM : 8Go - DDR3 - 1866MHz - CL10

The language used to perform the benchmarks will be Java 1.7 and we will use the following API's :

- JDBC for the MySQL connectivity
- XML:DB for the eXist-db connectivity

[1] More details at http://snap.stanford.edu/data/web-Google.html

3.1 Creation of the databases

This section details how to create the 2 different types of database starting from the file provided by Google (Code 3.1).

3.1.1 Parsing of the file

The first step of the database creation is to parse the data file. This action has been achieved by using script I've programmed (the full code is provided in the appendix B.1).
Here are the main steps of the program :

- Opening the file
- Create an empty list that will contain the x first nodes (x being a paramater provided as input at the beginning of the program)
- While we haven't reached the end of the file and we haven't added x nodes to the previous list, we format the current line to one of the XML format accepted by MySQL (see 2.6, 2.7 or 2.8)
- return the list containing the x first nodes

The python script is used like this :

Code 3.2: Command used to parse the Google file (Code 3.1)

```
python converter.py $google_file_name $number_of_line_to_parse > $xml_ouput_file
```

- $ google_file : name of the input file with the Google format
- $ number_of_line_to_parse : number of nodes to save (corresponds to the x variables detailed above)
- $ xml_ouput_file : name of the file where the output will be written

In this report I've chosen to make 6 different benchmarks at 500, 1000, 1500, 2000, 2500 and 3000 first encountered nodes.
In order to have a good approximation of the computational time needed for each request and each file, I will do the tests 3 times and compute the average time needed.
If further details are needed, the code is available in the appendix (**??**)

3.1.2 Creation of the MySQL database

First, we need to create the database and the tables needed in order to receive the datas parsed at the previous step. In our case, we will only need one table, called Digraph, which will contain all edges of the digraph corresponding to the data contained in the original Google file.

This table has only 3 attributes :

- id_node : unique id of the relation between the to nodes.
- origin : origin node
- destination : destination node

Here's the SQL syntax used to create the database and the table.

Code 3.3: Creation of the MySQL database

```
CREATE DATABASE INFOH415;

USE INFOH415

CREATE TABLE Digraph (
    id_node INT UNSIGNED NOT NULL AUTO_INCREMENT,
    origin INT UNSIGNED NOT NULL,
    destination INT UNSIGNED NOT NULL,
    PRIMARY KEY(id_node),
    UNIQUE(origin, destination)
) engine=innodb;
```

Now our database structure is ready, we can fill it with the data comming from the XML file parsed at the previous step.

In order to do that, we will first need to move the XML files to /var/lib/mysql/$database_name/ so that the mysql program will find then when using the loading command [2].

Code 3.4: Loading of the XML file

```
LOAD XML INFILE $file
    INTO TABLE db_name.table_name;
```

Where $file is the name of the XML file to import.

The datas contained in the original google file have now been fully imported in the MySQL database. In order to do the on all the files, we have two possibilities :

- Keep the table created at the previous step and add the new nodes by using the IGNORE option usable with the LOAD command.
- Delete the content of the table with the command below and recreate it with the command Code 3.4.

  Code 3.5: Command deleting all the content in the Digraph table

  ```
  DELETE FROM Digraph;
  ```

[2]The loading command has been detailed at Code2.5

3.1.3 Creation of the eXist-db database

Because the database of eXist-db is a XML file, we can directly use the XML file created at section 3.1.1

3.2 First Query : search an edge in function of its ID

The first query that will be performed is a simple one : it consists in searching an edge in function of its unique id number (with a index on it in SQL).

3.2.1 SQL Request

The sql request in this case is trivial and requires a simple SELECT request with a WHERE clause corresponding to the searched node id.

Code 3.6: First SQL request for benchmarking

```
SELECT DISTINCT *
FROM Digraph
WHERE Digraph.id_node = 0;
```

3.2.2 XQuery

Code 3.7: First XQuery for benchmarking

```
collection("/db/apps/test/INFOH415/")/table_data[@name=500]/row[@id_edge=4]
```

3.2.3 Results

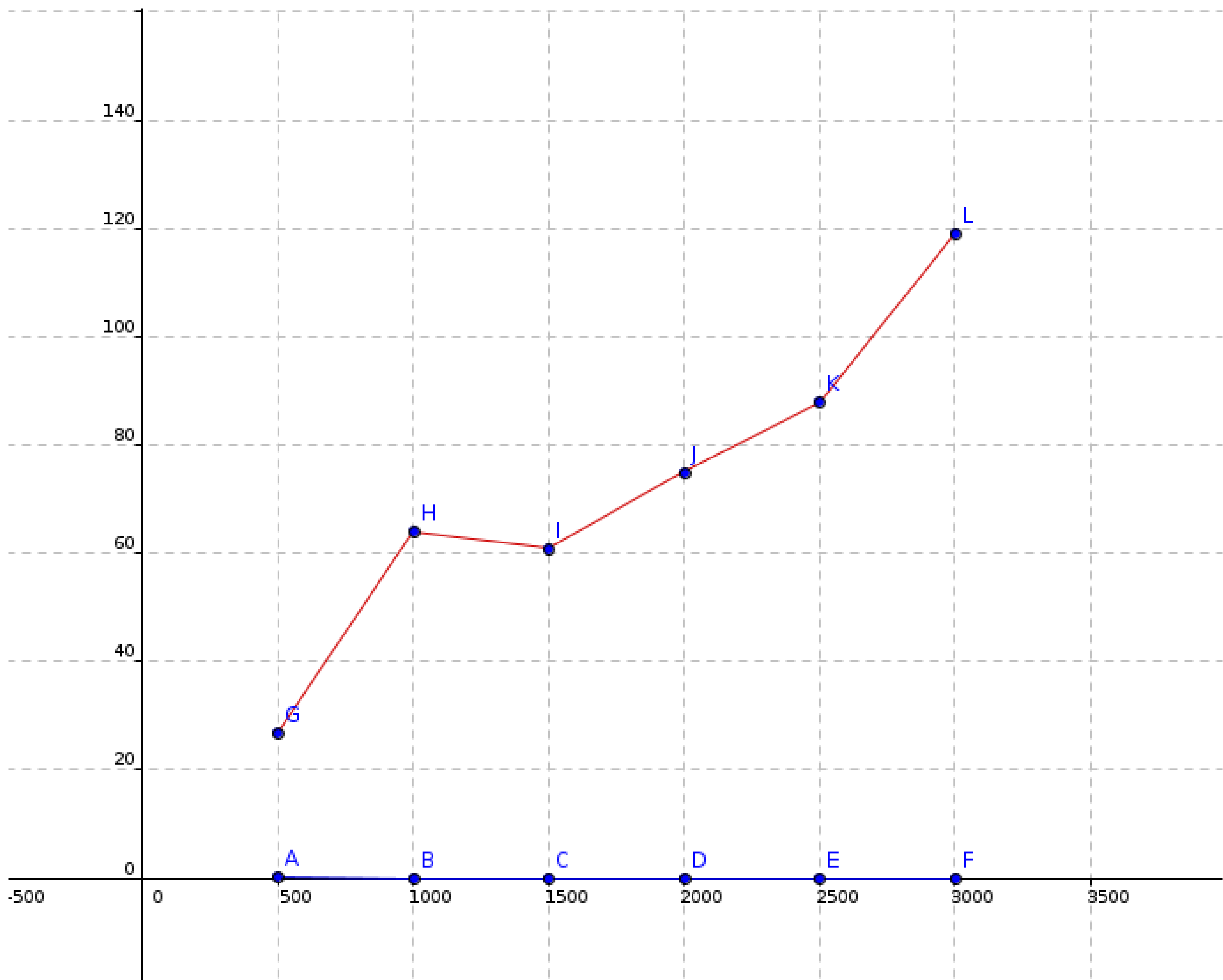


Figure 3.1: Comparison beween the time of search of an element in function of its id in eXist-db and MySQL

Legend :

- the X-axis represents the number of origin nodes contained in the database
- the Y-axis represents the time needed to search an element (in milisecond)
- The points A, B, C, D and F represent the collected data for MySQL
- The blue line joining the previously quoted points are the evolution of time in function of the number of element for MySQL
- The points E, F, G, H and I represent the collected data for eXist-db
- The red line joining the previously quoted points are the evolution of time in function of the number of element for MySQL

We can clearly see that MySQL holds the high ground, which is not that astonishing, as we know that MySQL puts an index on the column that serves as primary key for the table, implying high performences, with a search time very low (always less than a milisecond), even with a lot of datas.

EXist on the other has a growing time at almost each step, being even almost 120 time slower than MySQL for the last test.

3.3 Second Query : search all reachable destination (level 2)

The goal of this request is to make all the paths possible with a distance of 2 (if we suppose the edges all have a distance 1).

3.3.1 SQL Request

The search of all the paths of size 2 can easily be achieved by using an INNER JOIN[3] with the destination of an edge being equal to the origin of another edge.

Code 3.8: Second SQL request for benchmarking

```
SELECT DISTINCT *
FROM Digraph AS D1
INNER JOIN Digraph AS D2
  ON D2.origin = D1.destination;
```

3.3.2 XQuery

Code 3.9: Second XQuery for benchmarking

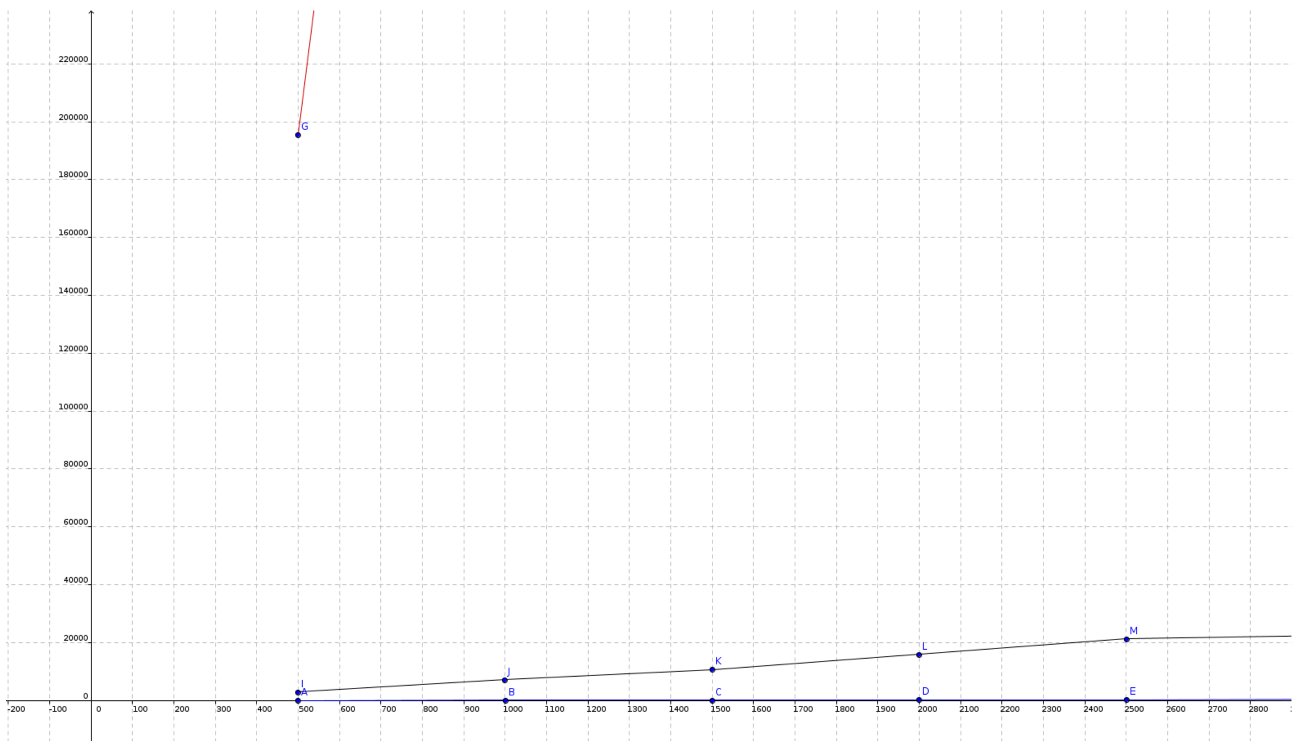
```
collection("/db/apps/test/INFOH415/")/table_data[@name="500"]/row[@origin =
    collection("/db/apps/test/INFOH415/result500.xml")/table_data[@name="500"]/row/@destinatio
```

3.3.3 Results

Legend :

- the X-axis represents the number of origin nodes contained in the database
- the Y-axis represents the time needed to search an element (in milisecond)
- The points A, B, C, D and E represent the collected data for MySQL for the join operation

[3] An example of INNER JOIN is available at figure1.2



- The blue line joining the previously quoted points are the evolution of time in function of the number of element for MySQL
- The points G and H represent the collected data for eXist-db for the join operation
- The red line joining the previously quoted points are the evolution of time in function of the number of element for eXist-db
- The points I, J, K, L and M represent the data collected for MySQL for the double join operation (see 3.4 for more details)
- The black line joining the previously quoted points are the evolution of time in function of the number of element for MySQL double join

We can see a huge gap in term of performances between eXist-db and MySQL. Indeed, let's take a closer look at each lines, comparing it two by two :

- Blue line and black line : we can see a pretty big difference here, which is not a surprise as we knpw that the black line requires a join operation more than the blue line. This shows that the join operation is a costly operation.
- Blue line and red line : this is where the gap in term of performance is huge, we can indeed see that for the same request, there's (for the smaller one) a difference factor of almost 4162. Making the join operation in eXist-db almost unusable with a lot of datas (the join operation with 3000 nodes taking up to 10 minutes).
- Black line and red line : this comparison allows to show even more the difference in term of performances. Indeed, we can see that even with a double join on 3000 nodes, MySQL stays faster than eXist-db with a simple join and only 500 nodes

3.4 Third Query : search all reachable destination (level 3)

The idea here is to oblige the two engines to make a very costly request (a double join)

3.4.1 SQL Request

Code 3.10: Third SQL request for benchmarking

```
SELECT DISTINCT *
FROM Digraph AS D1
INNER JOIN (
  Digraph AS D2
  INNER JOIN Digraph AS D3
    ON D2.destination = D3.origin
  )
  ON D2.origin = D1.destination;
```

3.4.2 XQuery

Code 3.11: Third XQuery for benchmarking

```
collection("/db/apps/test/INFOH415/")/table_data[@name="500"]/row[@destination =
    collection("/db/apps/test/INFOH415/")/table_data[@name="500"]/row[@destination =
        collection("/db/apps/test/INFOH415/")/table_data[@name="500"]/row/@origin]/@origin]
```

For practical reasons of computational time, the double join operation with eXist-db has not been tested in this report (more details on the join operation in section 3.3.3).

The result for the double join operation with MySQL is available at the same section.

3.5 Conclusion

In conclusion, we can see that in term of pure performances, eXist-db is highly inferior to MySQL concerning more complex operations (such as joins), with differences of computational times being counted in tens of minutes in term for the same operations !

Although, there are some remarks to be done :

- MySQL and eXist-db did not use the same API to compute the results. This might imply in term of performances.

- During the benchmarking process, I've noted a very high CPU consumption when performing the operations concerning the MySQL database (Up to 600% of the CPU usage !). Meanwhile, eXist-db only used less calculation power (120% of the CPU max).

4 Conclusion

MySQL and eXist-db are two really different type of databases with advantages and disadvantages, which could be summarized with the following list :

- Performances : in term of performances, MySQL clearly has the high ground, with a realy fast time of execution
- Outlying deployment : in term of deployment, eXist-db clearly is much easier to use, the only thing needed is a XML file. It is also much easier for returning result, XML being a very used format with a lot of API available in order to treat the received XML result.

We can then say that if the database should be a highly used database with a lot of datas interlying with each other, MySQL should be the preferred one.

If the database should be less used, but requires a high flexibility of the stored datas, or a quick deployment, eXist-db should ne the preferred one.

Bibliography

[1] Suciu Dan. On database theory and xml. http://xml.coverpages.org/SuciuDatabaseTheoryXML.pdf.

[2] eXist db. exist documentation. http://exist-db.org/exist/apps/doc/documentation.xml.

[3] eXist db. Xquery in exist-db. http://exist-db.org/exist/apps/doc/xquery.xml.

[4] Joshua W. Green. A comparison of the relative performance of xml and sql databases in the context of the grid-safe project. 2008.

[5] MySQL. Load xml syntax. https://dev.mysql.com/doc/refman/5.5/en/load-xml.html.

[6] MySQL. Mysqldump. http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html.

[7] Matthias Nicola. 5 reasons to use xml databases. http://nativexmldatabase.com/2010/09/28/5-reasons-for-storing-xml-in-a-database/.

[8] Oracle. A relational database overview. https://docs.oracle.com/javase/tutorial/jdbc/overview/datab

[9] Elmasri Ramez and Navathe Shamkant B. *Fundamentals ofDatabase Systems*. Pearson Education, Inc.

[10] Adam Retter and Erik Siegel. *eXist*. O'Reilly Media, Inc., 2014.

[11] Bloor Robin. The failure of relational database, the rise of object technology and the need for the hybrid database. 2004.

[12] tutorialspoint. Jbdc simple example. http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm.

[13] w3c. Xpath specifiactions. http://www.w3schools.com/xsl/xpath_intro.asp.

[14] w3c. Xquery specifiactions. http://www.w3schools.com/xsl/xquery_intro.asp.

A XML FILE GENERATED BY THE MYSQLDUMP COMMAND

Code A.1: Result of the Code 2.1 or Code 2.2 command on the previously created database

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="Movie">
  <table_structure name="Actors">
    <field Field="id_actor" Type="int(10) unsigned" Null="NO" Key="PRI"
        Extra="auto_increment" Comment="" />
    <field Field="name" Type="text" Null="NO" Key="" Extra="" Comment="" />
    <key Table="Actors" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1"
        Column_name="id_actor" Collation="A" Cardinality="3" Null=""
        Index_type="BTREE" Comment="" Index_comment="" />
    <options Name="Actors" Engine="InnoDB" Version="10" Row_format="Compact"
        Rows="3" Avg_row_length="5461" Data_length="16384" Max_data_length="0"
        Index_length="0" Data_free="9437184" Auto_increment="4"
        Create_time="2015-12-12 13:40:28" Collation="latin1_swedish_ci"
        Create_options="" Comment="" />
  </table_structure>
  <table_data name="Actors">
  <row>
    <field name="id_actor">1</field>
    <field name="name">Elja Wood</field>
  </row>
  <row>
    <field name="id_actor">2</field>
    <field name="name">Christopher Lee</field>
  </row>
  <row>
    <field name="id_actor">3</field>
    <field name="name">Ewan McGregor</field>
  </row>
  </table_data>
  <table_structure name="Acts">
    <field Field="id_relation" Type="int(10) unsigned" Null="NO" Key="PRI"
        Extra="auto_increment" Comment="" />
    <field Field="id_actor" Type="int(10) unsigned" Null="NO" Key="MUL" Extra=""
        Comment="" />
    <field Field="id_movie" Type="int(10) unsigned" Null="NO" Key="MUL" Extra=""
        Comment="" />
```

```
  <key Table="Acts" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1"
     Column_name="id_relation" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
  <key Table="Acts" Non_unique="1" Key_name="id_actor" Seq_in_index="1"
     Column_name="id_actor" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
  <key Table="Acts" Non_unique="1" Key_name="id_movie" Seq_in_index="1"
     Column_name="id_movie" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="Acts" Engine="InnoDB" Version="10" Row_format="Compact" Rows="4"
     Avg_row_length="4096" Data_length="16384" Max_data_length="0"
     Index_length="32768" Data_free="9437184" Auto_increment="5"
     Create_time="2015-12-12 18:12:19" Collation="latin1_swedish_ci"
     Create_options="" Comment="" />
</table_structure>
<table_data name="Acts">
<row>
  <field name="id_relation">1</field>
  <field name="id_actor">1</field>
  <field name="id_movie">2</field>
</row>
<row>
  <field name="id_relation">2</field>
  <field name="id_actor">2</field>
  <field name="id_movie">1</field>
</row>
<row>
  <field name="id_relation">3</field>
  <field name="id_actor">2</field>
  <field name="id_movie">2</field>
</row>
<row>
  <field name="id_relation">4</field>
  <field name="id_actor">3</field>
  <field name="id_movie">1</field>
</row>
</table_data>
<table_structure name="Distributors">
  <field Field="id_distributor" Type="int(10) unsigned" Null="NO" Key="PRI"
     Extra="auto_increment" Comment="" />
  <field Field="name" Type="text" Null="NO" Key="" Extra="" Comment="" />
  <key Table="Distributors" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1"
     Column_name="id_distributor" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
```

```
  <options Name="Distributors" Engine="InnoDB" Version="10" Row_format="Compact"
     Rows="2" Avg_row_length="8192" Data_length="16384" Max_data_length="0"
     Index_length="0" Data_free="9437184" Auto_increment="3"
     Create_time="2015-12-12 13:39:51" Collation="latin1_swedish_ci"
     Create_options="" Comment="" />
</table_structure>
<table_data name="Distributors">
<row>
  <field name="id_distributor">1</field>
  <field name="name">New Line Cinema</field>
</row>
<row>
  <field name="id_distributor">2</field>
  <field name="name">20th Century Fox</field>
</row>
</table_data>
<table_structure name="Movies">
  <field Field="id_movie" Type="int(10) unsigned" Null="NO" Key="PRI"
     Extra="auto_increment" Comment="" />
  <field Field="name" Type="text" Null="NO" Key="" Extra="" Comment="" />
  <field Field="id_distributor" Type="int(10) unsigned" Null="NO" Key="MUL"
     Extra="" Comment="" />
  <key Table="Movies" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1"
     Column_name="id_movie" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
  <key Table="Movies" Non_unique="1" Key_name="id_distributor" Seq_in_index="1"
     Column_name="id_distributor" Collation="A" Cardinality="2" Null=""
     Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="Movies" Engine="InnoDB" Version="10" Row_format="Compact"
     Rows="2" Avg_row_length="8192" Data_length="16384" Max_data_length="0"
     Index_length="16384" Data_free="9437184" Auto_increment="3"
     Create_time="2015-12-12 13:42:40" Collation="latin1_swedish_ci"
     Create_options="" Comment="" />
</table_structure>
<table_data name="Movies">
<row>
  <field name="id_movie">1</field>
  <field name="name">Star Wras II</field>
  <field name="id_distributor">2</field>
</row>
<row>
  <field name="id_movie">2</field>
  <field name="name">The Lors of the Rings</field>
  <field name="id_distributor">1</field>
</row>
```

```
  </table_data>
</database>
</mysqldump>
```

B Python script for parsing from Google file to XML format

Code B.1: Python script used to parse the Google format (Code 3.1) to a XML file compatible with MySQL and eXist-db (Code 2.6, 2.7 or 2.8)

```
import sys

def convert_to_xml(filename, lines_to_parse):
  f = open(filename, "r")
  l = f.readlines()
  f.close()

  i = 0
  j = 0
  encountered_nodes = []
  entry_table = "<table_data name=\"" + str(lines_to_parse) + "\">"
  converted = [entry_table]
  while (i < len(l)) and (j <= lines_to_parse):
    if l[i][0] != "#":
      j, converted_line = line_to_xml(l[i], i, j, encountered_nodes)
      converted.append(converted_line)
    i += 1

  converted.pop()
  converted.append("</table_data>")

  for i in range(len(converted)):
    print converted[i]

def line_to_xml(line, i, j, encountered_nodes):
  splitted = line.split()
  if not (splitted[0] in encountered_nodes):
    j += 1
    encountered_nodes.append(splitted[0])
  return j, "  <row id_edge=\"" + str(i) + "\" origin=\"" + str(splitted[0]) + "\"
     destination=\"" + str(splitted[1]) + "\"/>"

if __name__ == "__main__":
  filename = sys.argv[1]
  lines_to_parse = sys.argv[2]
  convert_to_xml(filename, int(lines_to_parse))
```

C Benchmark code

Those are the different file with the java code required to launch the whole benchmarking process :

C.1 Main file

Code C.1: Main.java

```
public class Main {

   private static long m_count = 3;

   private static String m_mysqlFile1 = "result500.xml";
   private static String m_mysqlFile2 = "result1000.xml";
   private static String m_mysqlFile3 = "result1500.xml";
   private static String m_mysqlFile4 = "result2000.xml";
   private static String m_mysqlFile5 = "result2500.xml";
   private static String m_mysqlFile6 = "result3000.xml";
   private static String m_mysqlFiles[] = {m_mysqlFile1, m_mysqlFile2,
      m_mysqlFile3, m_mysqlFile4, m_mysqlFile5, m_mysqlFile6};

   private static String m_mysqlQuery1 = "SELECT * FROM Digraph WHERE
      Digraph.id_node = 4;";
   private static String m_mysqlQuery2 = "SELECT DISTINCT * FROM Digraph AS D1
      INNER JOIN Digraph AS D2 ON D2.origin = D1.destination;";
   private static String m_mysqlQuery3 = "SELECT DISTINCT * FROM Digraph AS D1
      INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3 ON D2.destination =
      D3.origin) ON D2.origin = D1.destination;";
   private static String[] m_mysqlQueries = {m_mysqlQuery1, m_mysqlQuery2,
      m_mysqlQuery3};
   private static double m_mysqlResults[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0};

   private static String m_eXistFiles[] = {"500", "1000", "1500", "2000", "2500",
      "3000"};

   private static String m_XQuery1 =
      "/table_data[@name=\"TO_REPLACE\"]/row[@id_edge=4]";
   private static String m_XQuery2 =
      "/table_data[@name=\"TO_REPLACE\"]/row[@origin =
      collection(\"/db/apps/test/INFOH415/\")/table_data[@name=\"TO_REPLACE\"]/row/@destinati
```

```
private static String m_XQuery3 =
    "/table_data[@name=\"TO_REPLACE\"]/row[@destination =
    collection(\"/db/apps/test/INFOH415/\")/table_data[@name=\"TO_REPLACE\"]/row[@destinati
    =
    collection(\"/db/apps/test/INFOH415/\")/table_data[@name=\"TO_REPLACE\"]/row/@origin]/@
private static String[] m_XQueries = {m_XQuery1, m_XQuery2};//, m_XQuery3};
private static double m_eXistResults[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0};

public static void main(String[] args){
    MySQL_DB mysql = new MySQL_DB();
    EXist_DB eXist = new EXist_DB();
    for(int toRun = 0; toRun < m_count; ++toRun) {
        System.out.println("TEST NO : " + toRun + "/" + m_count);
        for (int i = 0; i < m_eXistFiles.length; ++i) {
            System.out.println("File operated : " + m_mysqlFiles[i]);
            System.out.println("MYSQL");
            mysql.loadTable(m_mysqlFiles[i]);
            for (int j = 0; j < m_mysqlQueries.length; ++j) {
                System.out.println("Query : " + m_mysqlQueries[j]);
                mysql.query(m_mysqlQueries[j]);
                m_mysqlResults[(m_mysqlQueries.length * i) + j] +=
                    mysql.getCount();
            }
            mysql.emptyTable();

            System.out.println();
            System.out.println("XQUERY");
            String query = m_XQuery1.replace("TO_REPLACE", m_eXistFiles[i]);
            System.out.println("Query : " + query);
            eXist.query(query, "test/INFOH415/");
            m_eXistResults[(m_XQueries.length * i)] += eXist.getCount();

            query = m_XQuery2.replace("TO_REPLACE", m_eXistFiles[i]);
            System.out.println("Query : " + query);
            eXist.query(query, "test/INFOH415/");
            m_eXistResults[(m_XQueries.length * i) + 1] += eXist.getCount();

        }
    }

    mysql.closeAll();
    System.out.println();
    System.out.println("======================================== RESULTS
        ========================================");
```

```
        System.out.println();
        for (int i = 0; i < m_eXistFiles.length; ++i) {
            for (int j = 0; j < m_mysqlQueries.length; ++j){
                System.out.println("Average time spent on query : ");
                System.out.println(m_mysqlQueries[j]);
                System.out.println("with file " + m_mysqlFiles[i]);
                System.out.println("TIME : " + m_mysqlResults[(m_mysqlQueries.length
                    * i) + j] / m_count / 1000000 + "ms");
                System.out.println();
            }
        }

        for (int i = 0; i < m_eXistFiles.length; ++i){
            System.out.println("Average time spent on query : ");
            System.out.println(m_XQuery1.replace("TO_REPLACE", m_eXistFiles[i]));
            System.out.println("with file " + m_eXistFiles[i]);
            System.out.println("TIME : " + m_eXistResults[(m_XQueries.length * i)] /
                m_count / 1000000 + "ms");

            System.out.println("Average time spent on query : ");
            System.out.println(m_XQuery2.replace("TO_REPLACE", m_eXistFiles[i]));
            System.out.println("with file " + m_eXistFiles[i]);
            System.out.println("TIME : " + m_eXistResults[(m_XQueries.length * i) +
                1] / m_count / 1000000 + "ms");

            System.out.println("Average time spent on query : ");
            System.out.println(m_XQuery3.replace("TO_REPLACE", m_eXistFiles[i]));
            System.out.println("with file " + m_eXistFiles[i]);
            System.out.println("TIME : " + m_eXistResults[(m_XQueries.length * i) +
                2] / m_count / 1000000 + "ms");
        }
    }
}
```

C.2 Mysql file

The class concerning the communication with the MySQL database.

Code C.2: MySQL_DB.java

```
import java.sql.*;

public class MySQL_DB {
```

```
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/INFOH415";

// Database credentials
static final String USER = "";
static final String PASS = "";

private Connection m_conn;
private Statement m_stmt;
private double m_count;

public MySQL_DB() {
    try {
        Class.forName("com.mysql.jdbc.Driver");

        m_conn = DriverManager.getConnection(DB_URL, USER, PASS);

        m_stmt = m_conn.createStatement();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void loadTable(String xmlFile){
    String sql = "LOAD XML INFILE \"" + xmlFile + "\" INTO TABLE Digraph;";
    try {
        ResultSet rs = m_stmt.executeQuery(sql);
        rs.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void emptyTable(){
    String sql = "DELETE FROM Digraph;";
    try {
        int rs = m_stmt.executeUpdate(sql);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void query(String sql) {
```

```
        try {
            long tStart = System.nanoTime();
            ResultSet rs = m_stmt.executeQuery(sql);
            long tEnd = System.nanoTime();
            this.m_count = (tEnd - tStart);
            System.out.println("RESULT : " + this.m_count);
            rs.close();
        } catch (SQLException se) {
            //Handle errors for JDBC
            se.printStackTrace();
        } catch (Exception e) {
            //Handle errors for Class.forName
            e.printStackTrace();
        }
    }

    public double getCount(){
        return this.m_count;
    }

    public void closeAll(){
        try {
            m_stmt.close();
            m_conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

C.3 eXist file

The class concerning the communication with the MySQL database.

Code C.3: EXist_DB.java

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;

public class EXist_DB {

    private static String URI =
        "xmldb:exist://localhost:8080/exist/xmlrpc/db/apps/";
```

```
private Database m_database;
private double m_count;

public EXist_DB(){
    final String driver = "org.exist.xmldb.DatabaseImpl";

    Class cl = null;
    try {
        cl = Class.forName(driver);
        m_database = (Database) cl.newInstance();
        m_database.setProperty("create-database", "true");
        DatabaseManager.registerDatabase(m_database);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (XMLDBException e) {
        e.printStackTrace();
    }
}

public void query(String query, String filename){
    Collection col = null;
    try {
        col = DatabaseManager.getCollection(URI + filename);
        XQueryService xqs = (XQueryService) col.getService("XQueryService",
            "1.0");
        xqs.setProperty("indent", "yes");

        CompiledExpression compiled = xqs.compile(query);
        double tStart = System.nanoTime();
        ResourceSet result = xqs.execute(compiled);
        double tEnd = System.nanoTime();
        this.m_count = tEnd - tStart;

    } catch (XMLDBException e) {
        e.printStackTrace();
    } finally {
        //dont forget to cleanup
        if(col != null) {
            try { col.close(); } catch(XMLDBException xe) {xe.printStackTrace();}
        }
```

```
        }
    }

    public double getCount() {
        return m_count;
    }
}
```

C.4 Benchmark program output

Example of the benchmark program output

```
======================================== RESULTS ========================================

======================================== RESULTS ========================================

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result500.xml
TIME : 0.3333333333333333ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result500.xml
TIME : 47.0ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result500.xml
TIME : 2972.3333333333335ms

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result1000.xml
TIME : 0.3333333333333333ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result1000.xml
TIME : 99.66666666666667ms

Average time spent on query :
```

```
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result1000.xml
TIME : 7223.333333333333ms

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result1500.xml
TIME : 0.0ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result1500.xml
TIME : 148.66666666666666ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result1500.xml
TIME : 10712.0ms

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result2000.xml
TIME : 0.3333333333333333ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result2000.xml
TIME : 313.3333333333333ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result2000.xml
TIME : 15973.0ms

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result2500.xml
TIME : 0.0ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result2500.xml
```

```
TIME : 264.6666666666667ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result2500.xml
TIME : 21281.0ms

Average time spent on query :
SELECT * FROM Digraph WHERE Digraph.id_node = 4;
with file result3000.xml
TIME : 0.0ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN Digraph AS D2 ON D2.origin = D1.destinat
with file result3000.xml
TIME : 418.3333333333333ms

Average time spent on query :
SELECT DISTINCT * FROM Digraph AS D1 INNER JOIN (Digraph AS D2 INNER JOIN Digraph AS D3
with file result3000.xml
TIME : 22403.0ms

Average time spent on query :
/table_data[@name="500"]/row[@id_edge=4]
with file 500
TIME : 27.186995ms
Average time spent on query :
/table_data[@name="500"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data[
with file 500
TIME : 195579.427206ms
Average time spent on query :
/table_data[@name="1000"]/row[@id_edge=4]
with file 1000
TIME : 64.28772866666667ms
Average time spent on query :
/table_data[@name="1000"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data
with file 1000
TIME : 780695.4224136666ms
Average time spent on query :
/table_data[@name="1500"]/row[@id_edge=4]
with file 1500
TIME : 61.79751166666665ms
```

```
Average time spent on query :
/table_data[@name="1500"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data
with file 1500
TIME : 1776834.0850523333ms
Average time spent on query :
/table_data[@name="2000"]/row[@id_edge=4]
with file 2000
TIME : 75.16550133333332ms
Average time spent on query :
/table_data[@name="2000"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data
with file 2000
TIME : 2851473.7605643333ms
Average time spent on query :
/table_data[@name="2500"]/row[@id_edge=4]
with file 2500
TIME : 88.921065ms
Average time spent on query :
/table_data[@name="2500"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data
with file 2500
TIME : 4821692.128955ms
Average time spent on query :
/table_data[@name="3000"]/row[@id_edge=4]
with file 3000
TIME : 119.386111ms
Average time spent on query :
/table_data[@name="3000"]/row[@origin = collection("/db/apps/test/INFOH415/")/table_data
with file 3000
TIME : 6405930.657636667ms
```