Testing SQL-compliance of current DBMSs

Elias Spanos

Master of Science

School of Informatics

University of Edinburgh

2017

Abstract

Database Management Systems (DBMSs) are popular for storing, manipulating and retrieving a variety of information. Also, these systems have been utilized in many applications such as banking and financial sectors by different companies. As the demand for such systems have been increased rapidly within recent years, different implementations have been provided. For example, a lot of companies provide their own DBMS implementation, and inevitability there are significant differences between these systems regarding their internal architecture. For making the transition between these systems smoother a common language was adopted early.SQL is a database language which is used to manipulate and retrieve data and is supported by all current DBMSs. Albeit different systems implementations are exists, in principle, they should follow the SQL standards with the same manner. A crucial question that we want to investigate by conducting this project is whether DBMSs are implemented the SQL standards in the same way. The SQL language should pledge that identical SQL code should always return identical answers when is evaluated on the same database independently of which DBMS is running on. However, there are indications that there are cases where SQL standards are interpreted and implemented differently. So that identical SQL queries does not always retrieve the same answers. The aim of this project is to investigate and highlight the differences that might exist among current DBMSs.

Acknowledgements

I would like to thank you my supervisors Paolo Guagliardo and Leonid Libkin who were always willing to advise and help by giving me valuable ideas. In addition, i would like to thank you my family who is always to my side to overcome all the difficulties that i had to face [3].

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Elias Spanos*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

DBMSs are widely used in many fields and find application in many companies as they provide relatively easy way of performing various common operations on data, such as insertion, deletion and update, and at the same time they hide the internal complexity of such system. More precisely, DBMS is a software that is designed to allow the creation, querying and update of databases [2,3]. The main role of a DBMS is to store and manipulate data efficiently and consistently. Figure 1.1 shows in a high level the structure of modern DBMSs. Additionally, Structured Query language known as SQL is a standardised programming language for managing relational databases [1]. By having such standards, it can be provided easily a common interface for all DBMSs in order to manipulate and retrieve data from any given database without worrying about the internal implementation.

The aim of this project is to evaluate five systems such as MySQL, IBM DB2, Microsoft SQL Server, PostgreSQL and Oracle Database in order to identify if they interpret the SQL standards in exactly the same way.

As it was mentioned all current DBMSs currently support the SQL standards, meaning that there is a common language, namely SQL, that is used by all systems to access, manipulate and retrieve data. Nevertheless some aspects of the standards are not well defined which make the process of interpret and implement it a difficult task and as a results, companies implement the SQL standards in a different manner. As a consequence programs that are written in SQL are partially portable among different DBMS.

$\frac{2}{3}$

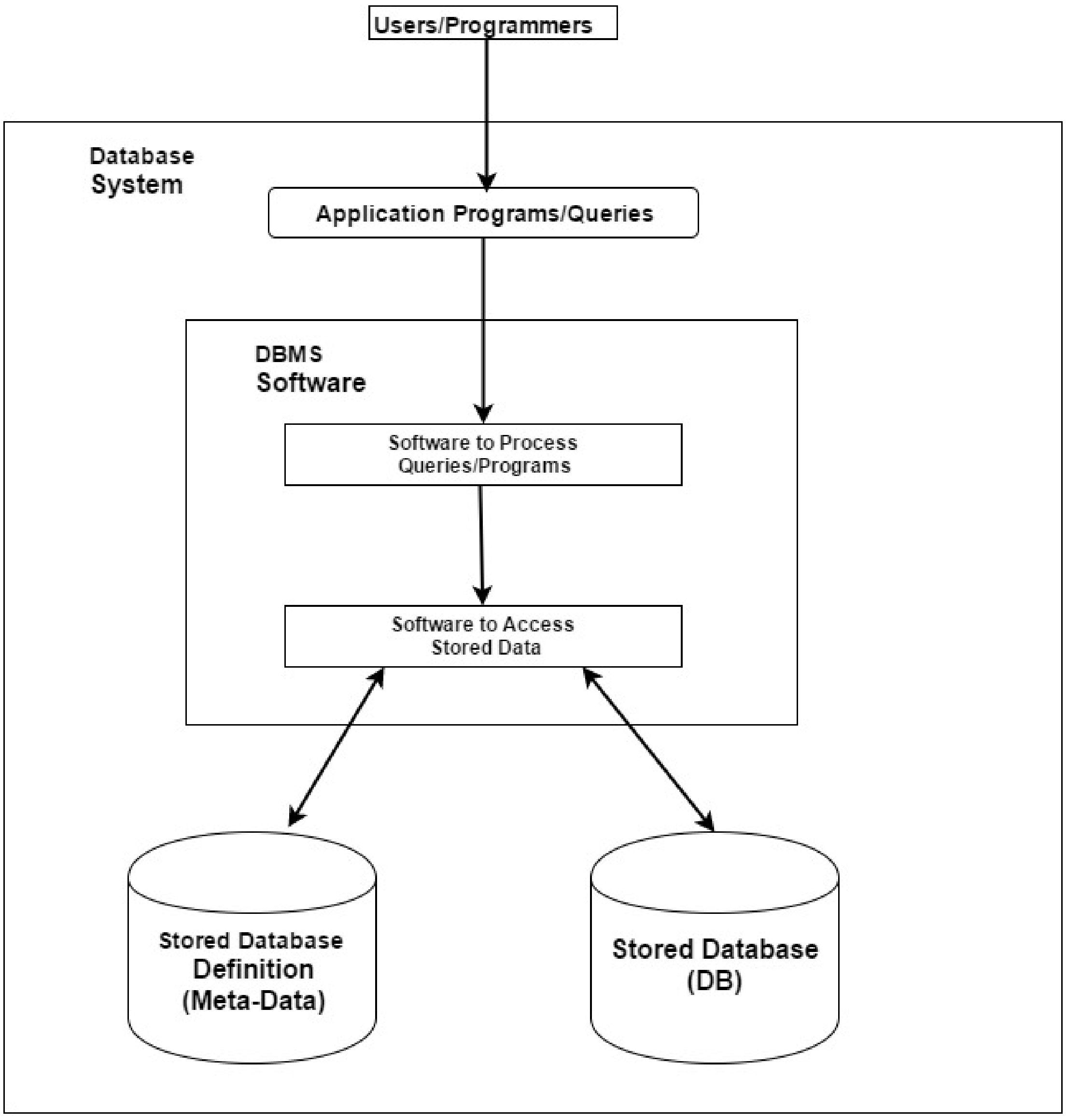
Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database (DB)

Figure 1.1: DBS architecture

Table 1.1: My caption

Is it a bug?

	Y	**N**
Bugs	True Positive (TP)	False positive
Reported	False Negative (FN)	True Negative

Chapter 2

Background

SQL is a domain specific language that is used in many DBMSs for accessing and storing data. SQL is extremely popular because it offers two capabilities. Firstly, it can access many tuples using just one command and secondly it does not need to specify how to reach a tuple, for example by using an index or not. In that way, it is an easy language for managing data. Moreover, SQL is divided into three parts such as data definition language, data manipulation language and data control language. In this project, our attention will be on data manipulation language as we want to investigate whether current DBMSs use the language in the same way. Albeit the existence of SQL standards, it worthy mentioning that SQL code is not completely portable between current DBMS without changing the code. SQL implementations are not compatible between different DBMSs as most vendors do not completely follow the SQL standards in the same way. Nevertheless, PostgreSQL tries to follow the SQL standard [10].

Below is provided an example which illustrates that SQL code is not completely portable.

Example 1:

For example the SQL query below does not return identical results on both PostgreSQL and Oracle.

Q1: SELECT * FROM (SELECT S.A, S.A FROM S) R

While we expect Q1 to return identical results independently on which systems is executed on, this is not the case. It can be observed that Q1 will output a table with two columns named A in PostgreSQL. On the other hand, in Oracle database, the SQL query will return an compile-time error. ndisputably, there are differences between current DBMSs [1]. It can supposed that in most of the cases these differences are minor but if we take into account that these systems are used in many different fields, then we can quickly realise that small differences might be critical for some companies. The key idea is to be conducted an experimental evaluation of current DBMSs. For achieving that we need to designed and implemented many tools, namely random SQL generator engine, execute and compare results tool. In addition, an open-source tool will be used to generate realistic random data sets of the DBMS. That is, store the same data in all DBMSs and execute thousands of queries on them in order to identify whether there are differences or not. Apparently, we are expecting in all of the cases the query results to be identical. On the contrary, we have provided a concrete example that the results are not always identical in different DBMSs, even though the SQL query was exactly the same. Therefore, this work intends to provide concrete evidence of these differences.

Chapter 3

Implementation

In this chapter we describe in details about the framework that we have designed and implemented for testing the SQL-compliance for current DBMSs. The framework is composed by different tools such random SQL generator engine, SQL results comparison tool and random data generator.

3.1 Random SQL generator engine

An important component of the framework is the random SQL generator tool which can be used to generate thousand of different queries for evaluating the current DBMSs. It is worthy mentioning that the tool can be used by its own use as well. For that reason, we have designed and implemented this tool from scratch using Java programming language. In addition, the tool has been designed in a way that is modular and reusable for supporting easily new DBMSs in the future by adding a small portion of code. More precisely, the main idea of the tool is to generate a diversity of queries for testing the DBMSs.

3.1.0.1 Implementation details

The random SQL generator tool is consisted by different Java classes and a configuration file. The most important decision that we had to take was how we will design our engine in a way that will make it possible to generate a diversity of different valid SQL queries. Hence, we have an internal representation for each SQL query and each class is responsible for generating a different statement that contribute to the overall query. For example, one of the classes that constitutes the tool is the SELECT class. Apparently, this class is responsible for generating the SELECT clause of each SQL query. Having different classes for each clause, it makes it easier to extend the tool in order to add new functionality and at the same time there is no need to change other part of the code. The final SQL query is converted to an SQL string and is executed to the current DBMSs. Obviously, we could not generate SQL strings directly because we need to track things. If you generating strings, it will not possible to check if in the WHERE clause you mentioning attributes that exist in the FROM clause or they comes as parameters from the outer query. Thus, there is a need to track attributes for each clause. In that way we have achieving implement a tool with high cohesion. Another important problem that we have to handle was how to avoid generating completely random SQL queries. Even though we need to generate random SQL queries to stress the current DBMSs in different situations, we need to control some characteristics of the SQL query. For example, imagine an SQL query that performs a cartesian product with a large number of tables. In this situation, we will have as a result the query to be executed for a long time or even for ever and it is not so useful for our case. Parameters such as maximum level of nesting and max number of tables in a FROM clause can be given in the configuration file. As a consequence, our tool supports a configuration file that can be used to control the randomly generated SQL queries. Below is provided a part of the configuration file and some parameters are explained.

It can be seen from the configuration file that we can control many parameters. For example, we can set an upper bound of tables that we can have in the FROM clause. In this way, we avoid having an enormous table from cartesian product. In addition, even it is not so usual to have constant comparison in an SQL query, nevertheless SQL standards support this. Thus, we generate SQL queries which have constant comparisons but we do not need to have a lot of such queries. As a result, we add a parameter for determine the probability of having such cases. The parameters name is probWhrConst which have a domain between 0 and 1 meaning that if the value is 1 that there will be exist for sure a constant comparison. Another important decision that should be taken into account is how we can provide the relations and attributes to the tool. An initial approach was to be given as parameters in the configuration file. Albeit this approach works pretty well, it makes our tool not portable. Image if the DBMSs have lots of tables with many columns. Hence, It will be time-consuming to give these parameters via the configuration files. Thus, an efficient approach is to retrieve the whole schema from DBMSs automatically. As a result, our tool has the capability to automatically retrieve the whole schema for any DBMS just by providing the credential for connecting to the database in the configuration file.

3.2 Comparison Tool

3.3 Generate data

Datafiller is a well-known open-source project that provides the capability of generating random data.Thus we will use this project for generating a data set in order to evaluate it in all DBMS. More precisely, the datafiller script generates random data, based on a data schema which is given as a parameter, and by taking into account constraints of that schema. For example, it takes into account the domain of each field and if the field should be unique, foreign key or primary key. Another important parameter is the df: null=x.xx

Additionally, more complex parameters can be provided as well, such as a number of tuples per table using –size SIZE parameter. It worthy mentioning that these parameters should be defined within the schema script and should start with – df. Further, we can generate more realistic data by providing some information in the schema script of the database. For example, if we have a field that represents a date, then we can provide a specific range in order for the datafiller to generate dates only within this range. This can be achieved by specifying the following parameter: range – df: start=year-month-day end=year-month-day beside the date field. Subsequently, we need to add the –filter parameter while running the script. These are only some of the important parameters that the datafiller provides but apart from these, it provides more sophisticated properties which are out of importance for our project.

It is important mentioning that the datafiller does not support importing data to other databases such as Oracle database, MS Server or IBM DB2 except postgresSQL. As a result, our approach is to import the data in postgreSQL and then export the random generated data in a CSV files. In this way, we can import the CSV files in all the DBMS, as all of them support importing data from CSV files.

Chapter 4

Experimental Evaluation

In this chapter we will present the procedure that we follow in order to conduct our experiments. Subsequently, we will present our findings and highlight the main differences that we have found. Also, it will be described the environment and the DBMSs that we have tested.

4.1 Experimental Set Up

We conduct all the experiments on Windows 8 with i7 CPU, 12GB Ram and solid state disk. Also, we install the mainstream DBMSs such as MySQL, IBM DB2, Microsoft SQL Server, PostgreSQL and Oracle Database

4.2 Experiment Results

In this section we present our findings by illustrates the differences that we have detected and give an explanation about those differences

4.2.0.1 Difference 1

```
SELECT r41.A AS A0
FROM r4 AS r41
WHERE 1*1
```

Mysql and IBM DB2 work properly if there are arithmetic calculations in the where clause such as 1*1. Even though expressions in the where clause should be evaluated to true or false, these two DBMSs convert the arithmetic expression to boolean type.

Table 4.1: My caption

Difference 1	
DBMS	**Result**
Mysql	**Works**
PostgreSQL	**[42804] ERROR: argument of WHERE must be type boolean, not type integer**
MS Server:	**[S0001][4145] An expression of non-boolean type specified in a context where a condition is expected, near ’1’.**
Oracle	**[42000][933]ORA-00933: SQL command not properly ended**
DB2	**Works**

Nevertheless, the rest three DBMSs do not do that, as a result, they throw an exception while they executing the query.

4.2.0.2 Difference 2

```
SELECT r21.A AS A0, r21.B AS A1, r21.B AS A1
FROM r2 AS r21
WHERE true
```

Table 4.2: My caption

Difference 1	
DBMS	**Result**
Mysql	**Works**
PostgreSQL	**[42804] ERROR: argument of WHERE must be type boolean, not type integer**
MS Server:	**[S0001][4145] An expression of non-boolean type specified in a context where a**
Oracle	**[42000][933]ORA-00933: SQL command not properly ended**
DB2	**Works**

DBMSs evaluate each expression in the where clause to a boolean type. Thus, instead of having an expression, it can be specified the true/false keyword which is a boolean type. Nevertheless, not all the DBMSs support this. MS server and Oracle do not support this keyword.

Chapter 5

Conclusions

Bibliography

[1] Arvin, T. (2006). Comparison of different sql implementations. *Troels Arvin's home page*.

[2] Codd, E. F. (1990). *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc.

[3] Date, C. J. and Darwen, H. (1987). *A Guide to the SQL Standard*, volume 3. Addison-Wesley New York.

[4] Elmasri, R. (2008). *Fundamentals of database systems*. Pearson Education India.

[5] Ramakrishnan, R. (2000). Database management systems . pdf.

[6] Ullman, J. D., Garcia-Molina, H., and Widom, J. (2002). Database systems: The complete book.