Testing SQL-compliance of current DBMSs

Elias Spanos

Master of Science

School of Informatics

University of Edinburgh

2017

Abstract

Database Management Systems (DBMSs) are widely used in various fields such as in banking and financial sectors. Hence, the correct and efficient operation of such systems are crucial. As the demand for these systems have been increased rapidly in recent years, different implementations have been proposed and implemented. For example, different companies have been implemented their own system, and inevitably there are significant differences between these systems regarding their internal implementation and architecture. For making easier the transition between these systems a common language was adopted. More precisely, 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 exist, in principle, they should follow the SQL standards in the same manner. A crucial question that will be investigated by conducting this project is whether DBMSs have been implemented the SQL standards in the same way. The SQL language should pledge that identical SQL code should always return identical answers when it is evaluated on the same database independently of which DBMS is running on. However, there are indications that this is not the case, as SQL standards are interpreted and implemented differently so that identical SQL queries do not always retrieve the same answers. The aim of this project is the implementation of a random SQL query generator and a comparison tool for investigating and highlighting the differences that may exist among current DBMSs. Further, we aim to provide a detailed explanation in regards to SQL standards of potential differences and explain how they might affect the transition between current DBMSs.

Acknowledgements

I would like to thank my supervisors Paolo Guagliardo and Leonid Libkin who were always willing to advise and help me in order to overcome any difficulty. In addition, I want to thank my family who is always to my side.

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 Introduction

DBMSs are widely used in many fields and find application in many companies as they provide relatively easy way of performing various operations on data, such as insertion, deletion, manipulation and update and at the same time they hide their internal complexity [2,3]. For this reason, applications can be implemented efficiently without worrying for handling low-level issues such as concurrent access to data and efficient access which it is taken cared by DBMS. Hence, the main role of a DBMS is to store and manipulate data efficiently and consistently. Figure 1.1 shows in an abstract point of view 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 an easily common interface for all DBMSs in order to manipulate and retrieve data from any given database without caring about the internal implementation.

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 manipulate 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.

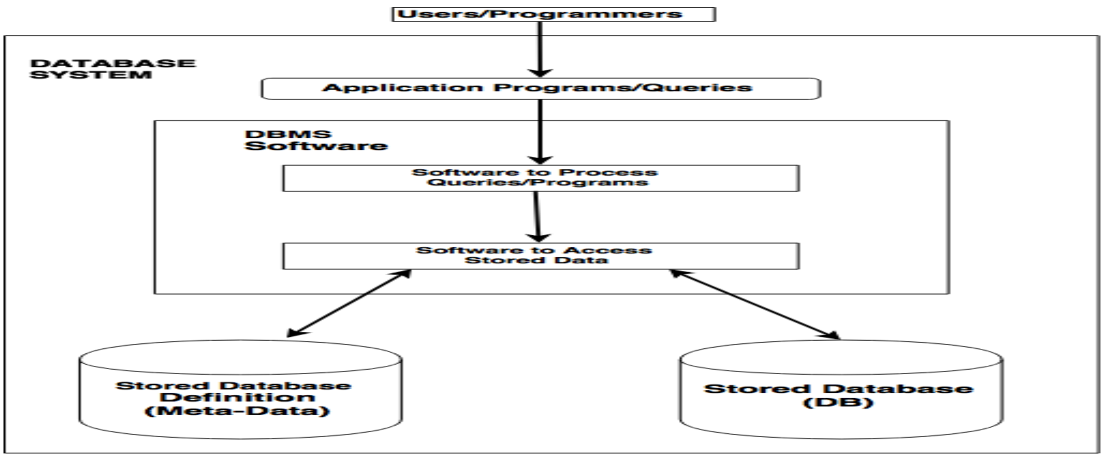
Users/Programmers
DATABASE SYSTEM
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database (DB)

Figure 1.1: Abstract DBMS architecture

1.2 Motivation

SQL standards are set up for a long time, and they aim to provide a common use for modern DBMSs. As no specific DBMS predominates to each other, by studying in which extend various implementations differ in terms of their features that provide, is almost essential. These systems have been evolved rapidly, and new features are continuously added. As a result, many companies which use these systems, struggle to change their existing DBMS as their SQL code is not portable among other popular DBMSs. Hence, our initial goal is to have a tool to systematically discovers and highlight existing and new differences that may arise. Our tool will be used to evaluate five popular systems which are used by most companies, such as MySQL, IBM DB2, Microsoft SQL Server, PostgreSQL and Oracle Database. The results of this study will be significant for the vendors of the systems, users and companies that use these systems. Unfortunately, DBMSs do not always provide a meaningful message whenever an SQL code has a syntax error. As a consequence, it takes a time to detect and resolve the errors. By conducting this study, we intend to disclose all the incompatibilities that may exist.

1.3 Thesis Structure

The structure of the remainder of this thesis report is as follow: Chapter 2 provides an explanation about SQL language which is used with existing DBMSs and in addition it illustrates the main problem which should be investigated. Chapter 3 describes the main methodology that is used to investigate the problem. Chapter 4 gives a detailed explanation about the implementation. Chapter 5 presents our main findings and provide an explanation about the differences that exist between current DBMSs. Chapter 6 conclude this thesis project and provide ideas for future work.

Chapter 2

Background

2.1 Background

SQL is a domain specific language that is used primarily in many DBMSs for accessing and storing data. SQL is extremely popular as 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. SQL operations are in the form of commands written as statements known as SQL statement. 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. The standard language for relational database management system (RDMS) is SQL. The first appearance of SQL was in 1970 where IBM developed the first prototype of RDMS and it becomes a standard by American National Standard Institute (ANSI) in 1986 and by ISO in 1987 for being use as a base with most DBMSs. Since then different flavors of the standard have been emerged for revising previous versions or for adding new features such as SQL-87, SQL-89, ANSI/ISO SQL-92,ANSI/ISO SQL: 1999 and many more. The new ANSI SQL standards is divided into nine parts where for our project the most important part is the SQL/Foundation. This part contains central elements SQL. It can be easily understand that DBMSs cannot follow exactly the standard. The reason for that is based on the fact that standard is considerably complex and it is not practical to implement the whole standard. In addition, each vendor need to distinguish its product among other and as a result it adds extensions. In this project, our attention will be on SQL/Foundation part 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: 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

Methodology

3.1 Methodology

In this chapter will be describe the main strategy that will be used in order to test the SQL compliance of current DBMSs.

As it was mention, the key idea of the current project is to examine whether current DBMSs respect the SQL standards and to highlight the differences that might exist among those systems. One of the core component that will be used for testing the SQL compliance is the Random SQL Generator tool. Thus, the first goal is to design and implement such a tool which will generate a diversity of SQL queries by exposing the expressivity of SQL language. It is worthy mentioning that this tool can be important of its own use by utilize it for testing a new DBMSs. The implementation of the tool will be in such a way that a new DBMS can be added without affect almost anything of the current implementation. A detailed explanation about the implementation and the internal structure of the tool is given in the following Chapter. Additionally, by having such a tool, it will be possible to generate thousands of different SQL queries and evaluate them in five systems, namely PostgreSQL, Microsoft SQL Server, IBM DB2, Oracle Database and MySQL for identifying differences. Subsequently, another useful tool that it will be designed and implemented is the run and compare tool. Apparently this tool must be compatible with all the DBMSs and it should evaluate each query and compare the results of each DBMS whether they are identical with concerning the rest systems. The comparison of DBMSs results will be performed using main-memory data structure for achieving efficiency. In case where a difference between any system will be found, then a log file will be created to record all the differences. Again a detailed explanation of tools internal implementation is given in the next chapter. Lastly but not least, a tool is needed for generating random data in order to perform our experiments. For that purpose, datafiller will be used which is a well-known open-source tool that provides the capability of generating realistic data. The data are generated based on a database schema which is provided to the tools as a parameter and many parameters can be specified for generating realistic data with different sizes.

Chapter 4

Implementation

In this chapter we describe in details the framework that we implemented for testing the SQL-compliance for current DBMSs.

4.1 Implementation

The framework is composed by different tools such as random SQL generator engine, comparison tool and random data generator. All tools are implemented in Java programming language which can run on all platforms that support Java except the random data generator which we use an open source python script that can generate realistic datasets.

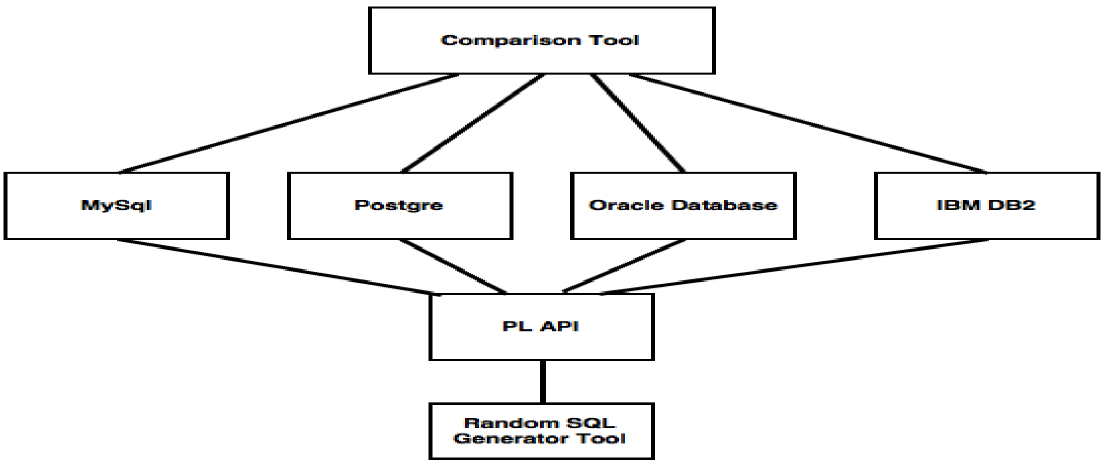

Figure 4.1: Random SQL Generator Architecture

4.1.0.1 Random SQL generator engine

One of the most important components of the framework is the random SQL generator tool which can be used to generate thousand of different queries for evaluating the current DBMSs. Obviously SQL language has a strong power expressivity and as a consequence it supports many commands where some of them are simple in terms of their use such as SELECT, FROM, WHERE and some other more trickier such as GROUP BY, HAVING or aggregation functions where we need to care about the proper use of these commands and generating syntactically correct SQL queries. As a result, for achieving that our implementation is composed by various classes where each of them is used for different purpose. It is worthy mentioning that the tool can be used by its own use for other purposes. Thus, in this work, we have designed and implemented this tool from scratch using Java programming language. In addition, the tool has been designed in a way that it is modular and reusable for supporting new DBMSs in the future without the need of changing the whole structure of the tool. Thus, the main idea of the tool is to generate a diversity of queries for testing the DBMSs. The random SQL generator tool is consisted by different Java classes and a configuration file. An important consideration was how to design our generator tool in a way that will be feasible to generate different valid SQL queries and at the same time to be syntactically correct. Hence, we have implemented an internal representation for generating randomly SQL queries 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 which obviously 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. It is not feasible to generate SQL strings directly because we need to track things. If we generating just strings, it will not possible to check if in the WHERE clause it is mentioning attributes that appear in the FROM clause or they comes as parameters from the outer query. Thus, there is a need to track attributes for each clause. Another important consideration that we had to think about was how to avoid generating completely random SQL queries. Even though we need to generate random SQL queries to stress the existing 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 consequence the query to be executed for a long time or even for ever and it is not so useful for our goal. Many parameters can be specified from the configuration file such as maximum level of nesting, max number of tables in a FROM, probability of having arithmetic comparisons. More details about the configuration file is given subsequently. 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 of the main parameters are explained.

It can be seen from the configuration file that we can control many parameters, nevertheless it does not imply that we restrict the diversity of SQL query that can be generated. For example, we can set an upper bound of tables that appear in the FROM clause. In that way, we avoid having an enormous table size 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. An example of such query is as follow:

SELECT r41.A AS A0 FROM r4 AS r41 WHERE 1 ¿ 2

We can specify the probability of having such cases by setting the parameter probWhrConst which have a domain between 0 and 1 meaning that if the value is 1 then there will be exist for sure a constant comparison. Another important parameter which can be specified from the configuration file is the level of nesting. We can randomly generate SQL queries with a specific level of nesting. The example below demonstrates a query with level of nesting three. For generating such a query many consideration should be taken into account. For example, we should track attributes for outer queries, as inner query can access outer attributes or attributes from its FROM clause. The opposite is not true, meaning that we cannot access attributes from an inner query.

SELECT r12.A AS A0, r12.B AS A1, r22.A AS A2 FROM r1 AS r12, r2 AS r22 WHERE NOT(NULL ¡¿ 9 OR r12.A ¿= r22.A OR (NOT(5 ¿ 3)) AND r12.A = ANY((SELECT r43.B AS A0 FROM r4 AS r43, r1 AS r13, r2 AS r23 WHERE (NULL ¿= 3)AND r22.B IN (SELECT r43.A AS A0 FROM r4 AS r44, r2 AS r24 WHERE (15 ¡ 2 AND NOT(NULL ¿= 14))

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

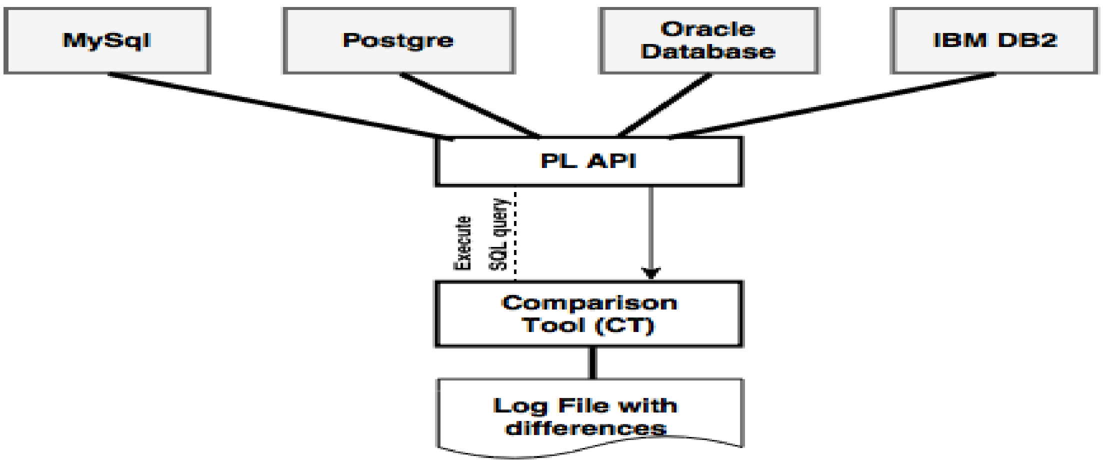

Figure 4.2: Simple Architecture of Comparison Tool

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.

4.2 Comparison Tool

As it was mentioned the purpose of the random query generator tool is to generate thousand of SQL queries which can be then used to evaluate current DBMSs. The result of each query can be huge in terms of its cardinality with different order. As a consequence, there was a need to automate the process and identify any difference that may exist by comparing the results efficiently. Therefore, we illustrate below our comparison tool (CT) which has been implemented for achieving the aforementioned goal and further we provide a detailed explanation about the internal implementation of the tool.

We have implemented the CT in Java programming language in order to compare the output results of each DBMSs. The tool is an important component of our project as it is used to conduct our experiments and identify the main differences that exist between current DBMSs. Apparently, differences may be minor or extremely difficult to be identified as DBMSs try to follow the SQL standards. Nevertheless, we illustrate in our experiments that differences exist and in some of the cases may be significant depends on the context that DBMSs are used. Comparing the results of each SQL

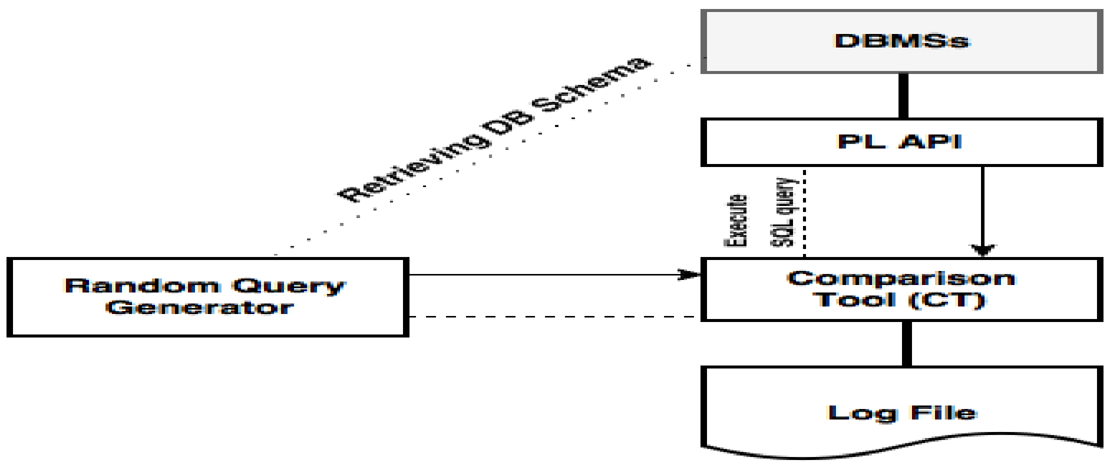

Figure 4.3: Architecture of Comparison Tool

query is not an easy task because each DBMSs use different algorithms to evaluate them and they return the output results in different order or their output format differ. The CT is fully compatible with the random generator engine and it takes as input each SQL query which is generated by the random SQL generator and subsequently evaluates each query to the current DBMSs. As a result, we use a data structure, namely LinkedList to store each row and then we use an efficient in-memory sorting algorithm to sort the rows. Then, we compare each rows of each DBMS and check if any row differ or does not exist. In case where the results are not identical, a log file is generated and we record which DBMSs differ and what is the SQL query that cause the problem.

4.3 Generate data

Datafiller is a well-known open-source project that provides the capability of generating random data. Thus it will be used in this project for generating a diversity of data sets in order to evaluate all major DBMSs. 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 5

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.

5.1 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

5.2 Experiment Results

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

5.2.1 Dif 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 5.1: 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.

5.2.2 Dif 2

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

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.

5.3 Summarize features

The below table summarizes the main features of SQL language and illustrated which of them are not supported by all popular DBMSs. These findings have been discovered by conducting experiments using the random generator query tool and the comparison tool. We have generated a huge number of SQL queries in order to identify lot of cases where DBMSs behave differently. It is worthy mentioning that the process of

Table 5.2: 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 condition is expected, near ’1’.**
Oracle	**[42000][933]ORA-00933: SQL command not properly ended**
DB2	**Works**

Table 5.3: My caption

Operation	Mysql	PostgreSQL	MS Server	Oracle	IBM DB2
INTERSECT ALL	X	✔			
AS in FROM					
EXCEPT ALL					

conducting experiments is fully automated and in case where a difference is found, it is recorded in a log file with some useful explanation.

Chapter 6

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.