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 the financial sector, the academia and other online services and are very crucial in the day to day management of data. The correct and efficient operation of such systems is an important factor when choosing the right DBMS software. In the past few decades, the amount of data has increased exponentially, causing a rapid increase in the demand for software that can store, organise and manipulate such data. With this requirements in mind, databases and DBMSs such as Oracle DB, Microsoft SQL Server, MySQL and PostgreSQL were created, each implementing its own set of rules. However, since these DBMSs have been implemented with a significant amount of differences, a set of standards was set by the American National Standards Institute (ANSI) in 1986 in order to keep all these implementations to a same standard and ensure compatibility for the SQL language adopted in all implementations. Even though, the SQL Standard has been introduced and adopted by most of the implementations, there are some differences that still exist, probably due to different interpretations of or noncompliance to the aforementioned Standards, and also due to other issues regarding performance of their systems.Since there do exist some differences between each implementation, migrations and changes from one DBMS to another might lead to some bottlenecks such as incompatibility or semantic issues. Therefore, there is a need for alleviating such a problem by exposing such issues between DBMSs, offering proposed solutions that have minimal negative effects to the performance of a system and also for evaluating their conformance to the common Standard. This project will investigate five different implementations and evaluate their conformance to the SQL Standard. 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.

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 me and help me in order to overcome any difficulty. In addition, I want to thank my family who is always by 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

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Motivation . . . 2
1.3 Related Work . . . 3
1.4 Thesis Structure . . . 3

2 Background **5**
2.1 The SQL Standard . . . 5
2.2 The SQL Language . . . 6
2.3 Commands of SQL . . . 6
2.3.1 Missing values . . . 9
2.3.2 SQL standard issues . . . 10

3 Methodology **12**
3.1 Methodology . . . 12

4 Implementation **14**
4.1 Implementation . . . 14
4.2 Comparison Tool . . . 18
4.3 Generate data . . . 19

5 Experimental Evaluation **21**
5.1 Experimental Set Up . . . 21
5.2 Experiment Results . . . 21
5.2.1 Dif 1 . . . 21
5.2.2 Dif 2 . . . 22
5.3 Summarize features . . . 22

6 Conclusions 24

6.0.1 Conclusions 24

6.0.2 Summary of the findings 25

6.0.3 Suggestions for future work 25

Bibliography 26

Chapter 1

Introduction

1.1 Introduction

Database Management System (DBMS) has thrived since its first appearance and it remains the dominant manner for storing various kinds of information. Specifically, DBMSs have been extensively used in many fields and have applications in almost all companies as they provide a relatively easy way of performing various operations on data, such as insertion, deletion and modification[4,6,7]. For that reason, applications can be implemented efficiently and reliably without the need for handling low-level issues such as concurrent and efficient access of data which it is taken cared by DBMS. Hence, the main role of a DBMS is not only to store data but also to provide a common interface for manipulating data. Figure 1.1 shows from a high-level point of view the structure of a modern DBMS.

In fact, in its early stage, each database system had its own interface and as a result migrating your SQL code among analogous systems it was almost impossible. Thus, all the code should be written again according to the specific DBMS interface. Nevertheless, as these systems were promising from their first appearance, a standardised language was unavoidable. Structured Query language (SQL) has become a standardised programming language for querying and managing data and has rapidly become the most widely used DBMS language[1]. Since then, comparable languages have been emerged over the years, SQL has persisted to be the dominant language since it has been easy to learn. In contrast with programming languages, where each language has its own benefits and usage, with SQL users and programmers can take advantage of it in order to learn a new language that it will be used by essentially all modern DBMSs and write SQL code that with minor changes it can be executed on any system.

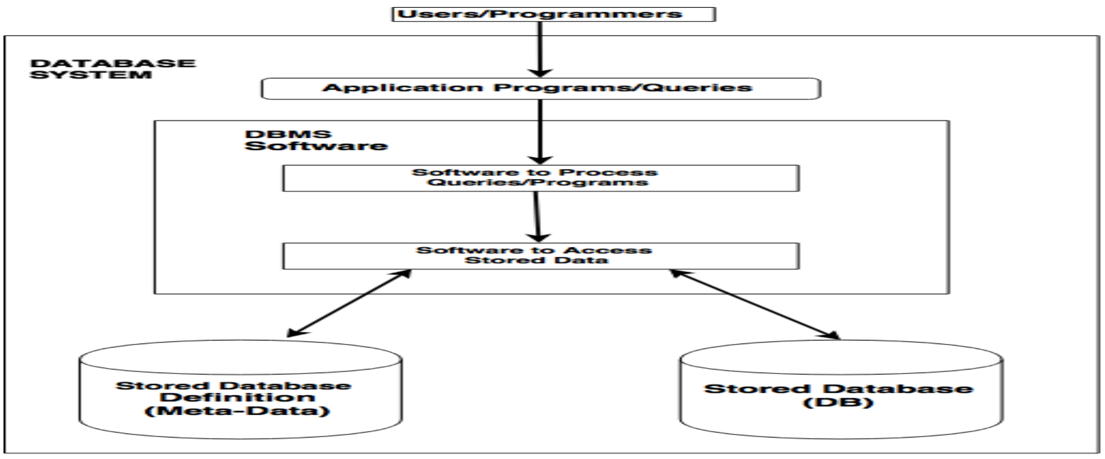
Users/Programmers
DATABASE SYSTEM
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database (DB)

Figure 1.1: Abstract DBMS architecture

Moreover, the relational model (RM) is the mainstream model for the DBMSs for managing data and it is by far the most popular model for current DBMSs. It was proposed by Edgar F.Codd in 1969 [15,16] and it has brought a revolution in the area of data management due to its simplicity. A database that uses the relational model is known as a relational database. According to the relational model, data is stored in a database as tables and each table is composed of rows and columns. Also, each row of a table is known as a tuple based on RM and each column as characteristic or attribute.

1.2 Motivation

The SQL Standard is established a long time ago, and aims to provide a common interface among modern DBMSs. As each vendor implements its own DBMS, the extend of which this implementation complies with the SQL Standard is essential and needs to be studied. Database Systems have been evolved rapidly, and new features have been continuously added. As a result, users, programmers and companies that make use of such systems are facing enormous problems when it comes to migrating their existing SQL code in another DBMS as most of the times the SQL code is not completely portable. Mainly three major reasons are involved in the aforesaid problem. Firstly, DBMS vendors do not always comply immediately to the latest SQL Standard because they have to cope with many considerations such as performance issues and better system management or even sometimes it is not practical to implement the entire Standard. Therefore vendors usually adopted new changes of the updated Standard into their next product releases. Secondly, DBMS vendors offer their own feature in order to be distinctive among other DBMSs which make SQL code less portable. Thirdly, the Standard is written in a natural language which makes the process of interpreting all the aspect of it in exactly the same way by all vendors, almost impossible. Currently, there is no well-known architecture that can be used to test the SQL-Compliance among these systems and detect any incompatibility. Hence, this research aims to build a complete architecture for systematically discovering and highlighting both existing and new differences that may arise between any DBMSs. Afterwards, experimental evidences with a comprehensive explanation with respects to the SQL standard are provided. The experimental evidences will be significant for vendors of such systems, users, and companies that utilize these systems. Unfortunately, DBMSs do not always provide a meaningful message whenever an SQL code has a syntax error. As a consequence, it can take considerable time to detect and resolve any issue. By conducting this research, it is also intended to disclose all the incompatibilities that may exist and highlight all the differences, with such a way that it will make users and programmers aware about the current problems.

1.3 Related Work

As DBMSs are necessary for many fields some work has been conducted for implementing the standard of the SQL language. In fact, some parts of the Standard is interpreted differently and SQL code is not portable among these systems. Albeit, some notorious differences are demonstrated [1, 19], there is no well-known tool for evaluating these systems for identify differences. As a matter of fact, queries that executed without raising any error but interpreted differently by DBMSs can cause different result but without a tool to check the result is almost impossible to be exposed. As a result, the current studies show results which have been detected by working with these systems. Having a tool for regularly checking the systems is imperative.

1.4 Thesis Structure

The structure of the remainder of this thesis report is organized as follow:

- **Chapter 2** introduces SQL language, SQL standards, the core commands of SQL and briefly describes the usage of the most important commands, and, afterwards it is introduced the main issues by illustrates problematic SQL queries.

- **Chapter 3** describes the main methodology for this research and briefly describes the explanation about the architecture which is implemented for systematically checking the SQL-compliance of current DBMSs.
- **Chapter 4** provides a detailed explanation about the complete architecture and an explanation for each tool that composed the architecture is given. Also, it is discussed briefly the main issues that are addressed.
- **Chapter 5** illustrates the main findings and provides an explanation about differences between current DBMSs with appropriate reference to the SQL standard.
- **Chapter 6** concludes this research project with explanation of what it is been achieved, and a summarize table of all the differences is provided. Afterwards, they are provided futures ideas and extensions.

Chapter 2

Background

2.1 The SQL Standard

DBMS from its first appearance shows that it will be the dominant trend for managing data. Consequently, different implementations have been emerged from various vendors and inevitable, a standardized language should be implemented in order to provide portability among different systems. If applications were implemented using only SQL commands which are defined in that standard and vendors implemented these commands in exactly the same way, then SQL code could be migrated on any DBMS without the need to be adopted. As it was mentioned, the common language for relational database management system (RDMS) is SQL.

The first appearance of the SQL language was in 1970 where IBM developed the first prototype of RDMS. Subsequently, the first SQL standard arose in 1986 by American National Standard Institute (ANSI) with the name SQL-86 for bearing conformity among vendors implementations. Since then different flavors of the standard have being emerged for revising previous versions or for adding new features such as SQL-87, SQL-89, ANSI/ISO SQL-92 and ANSI/ISO SQL: 1999 which has been approved also by International Standards Organization (ISO) [11]. SQL standard has been continuously developed with current version being SQL:2016 or ISO/IEC 9075:2016. The ANSI SQL standard is divided into several parts and this project focus on the SQL/Foundation part. This part contains central elements of SQL. Explanations about the findings are explained according to SQL:2016 since it is the newest version of the standard, though this parts remains the same in comparison with earlier flavors.

2.2 The SQL Language

SQL operations are in the form of commands known as SQL statements. More precisely, SQL is consisted primarily by two sublanguages such as data definition language (DDL) and data manipulation language (DML). DDL is a part of SQL language and can be used to create, modify, delete tables and views, and usually DDL statements start with keywords CREATE, DROP and ALTER. Also, it supports a command that give the capability to be defined new domains. Moreover, in general tables and rows are denoted as relations and tuples and sometimes it referred to these terminologies. DML is also a part of the SQL language that is consisted by a family of commands like any programming language and is used for the creation of a query for inserting, modifying and deleting rows in a Database. This sublanguage is consisted by SELECT-FROM-WHERE commands as to be the fundamental for any query. In addition, SQL standard supports more complex rather than just these simple commands for performing various tasks on data. For example, aggregation functions such as Sum, Max, Min, Avg are used with combination with Group By, and Having SQL statements. The purpose of having such commands is to perform a calculation on a specific columns in order to return a value. An example can be if we want to calculate the average salary of a department. Then, we need to perform a group by based on all the salaries of the employees of that department. Hence, 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.

2.3 Commands of SQL

This subsection intends to introduce in a high-level overview the basic commands of SQL which are defined in the SQL standard and it is described briefly the usefulness of each command. As some these commands are used to generate random SQL queries, it is important to provide a basic background of the usage and purpose of each command.

SQL Basic Structure

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

The SQL basic structure is used to retrieve data from a database according to some criterias which are specified in the WHERE clause. Each SQL query should contain at least SELECT and FROM clause and an optional WHERE clause. The main idea of the basic SQL query is to go through all the rows of the tables listed in the FROM Clause, and each row that satisfied the search criteria is selected. Then, only the specified columns of the selected rows are appear in the result. The DISTINCT keyword is optional and it is used to eliminate duplicates rows in the result. Also, instead of having columns_list in the SELECT clause, it can be used the keyword which indicates that all the columns will be appeared in the result. Different conditions can appear in the WHERE clause using logical connectivities such as AND, NOT and OR.

Example:

SELECT $St.Student_Name$
FROM Students AS St
WHERE $St.age >= 20ANDSt.age <= 24$

The above example uses the basic SQL structure to build a query. Thus, it returns all the name of students who are between 20 and 24 year old.

SQL Basic aggregation Syntax:

SELECT $[DISTINCT]Columns_list$
FROM $Tables_list$
WHERE Condition1 $AND|OR$ Condition2
GROUP BY $Columns_list$
HAVING Condition1 $AND|OR$ Condition 2

Queries with aggregation have as goal to perform a calculation on a specific columns in order to return a value. GROUP BY and HAVING are used to perform an aggregation. HAVING is an optional clause, and aggregation can be still perform using aggregate commands in the SELECT clause. A concrete example is given subsequently.

Aggregate commands can be used both in SELECT and HAVING Clause with a combination with the existence of GROUP BY clause in the SQL query. The below example illustrates the proper use of aggregate commands.

Example: :

[h] SELECT $COUNT(St.student_id), St.Country$
$FROMStudentsASSt$
$GROUPBYSt.Students$

Table 2.1: My caption

Aggregate Commands	
Command	**Usage**
MIN()	Finds the minimum value,of a column
COUNT()	Counts the total number,of rows
MAX()	Finds the maximum value,of a column
SUM()	Calculates the sum of,values of a column
AVG()	Calculates the average of,values of a column

```
$HAVINGCOUNT(St.student_id)¿3$
```

The above query makes proper uses of the aggregation commands. The concrete query list the number of students in each country where there are more than three students in a specific Country.

Table 2.2: My caption

Commands	
Command	**Usage**
EXISTS	Returns true if there is,at least one row in the subquery
Op ALL	Returns true if all the,comparisons using an operator OP are true
Op ANY	Returns true if at least,one comparison using an operator returns true
Op IN	Returns true if an,element exist in a given set
LIKE	Returns true if an,attribute matches with a pattern

Example: :

```
[h] SELECT *
FROM Students AS St
WHERE St.Country IN (UK, Cyprus, Netherland)
```

The above query retrieves all the students who come from UK, Cyprus and Netherland.

By default SET commands remove duplicates in the SQL result. Nevertheless, if it is needed to have duplicates in the result then the ALL keyword is used.

Example: :

Table 2.3: My caption

SET Commands	
Command	**Usage**
UNION [ALL]	Returns the combination,of the results of two SQL queries
INTERSECT [ALL]	Return the combination of,the results of two SQL queries for rows that,app
EXCEPT [ALL]	Return each row that,appear to the first query but does not appear to the sec

Table 2.4: My caption

String Commands	
Command	**Usage**
TRIM()	Returns the string,without leading/trailing characters
CONCAT()	Concatenate two or more,strings
REPLACE()	Replaces a subset of a,string with another string

```
[h] SELECT Country FROM Students
UNION
SELECT Country FROM Professor
```

The above query retrieves the Countries where there both Students and professors.

Example: :

```
[h] SELECT SUBSTRING (SQLSTANDARD, 1, 3 ) AS SQLExtraction
```

The above SQL query extract from the string which is given as parameter to the SUBSTRING function the first three characters starting from the position 1. Thus, the results is: SQL

2.3.1 Missing values

This section aims to provide a basic background regarding NULLs and introduce the problems that can be arised from having NULLs in a DBMS. In the section of experiments, it is illustrated that many problems can be appeared by using NULLs. SQL uses NULL marker for missing or unknown values in a database and for that reason NULL is a reserved word. It worthy mentioning that NULL should not be confused with a value of zero or an empty string. Nevertheless, Oracle treats the empty string as NULL [12]. An important consideration is that it cannot be tested if a value of a field

Table 2.5: My caption

Data Types	
Types	**Description**
SMALLINT	Returns the string,without leading/trailing characters
INT	Concatenate two or more,strings
BIGINT	Replaces a subset of a,string with another string
VARCHAR	Takes as input the length,of a variable string which can contains up to 255 character
CHAR	Takes as input the length,of a fixed size string which can contains up to 255 charact

is NULL using usual comparison operators such as $<>, =$ and $<$ but instead IS NOT NULL and IS NULL commands are used. In general the existence of NULL is the fundamental source of issues and incompatibilities among current DBMS. For evaluating each comparison with the existence of NULLS a three-valued logic (3VL) is proposed which is an extension of common boolean logic. In boolean logic, there are two values that an expression can be evaluated, namely, TRUE AND FALSE, where the negation evaluate to the opposite values. On the contrary with 3VL, in 3VL there is an addition value called unknown and the opposite of it remains the same. In addition, all comparisons involving NULL should be resulted to be unknown according to SQL Standard. Below it is illustrated a truth table for the different comparisons with the suitable outcomes.

2.3.2 SQL standard issues

Below it is provided a few concrete examples which demonstrate that indeed some aspects of the Standard is implemented differently by each vendor.

Example:

For example the SQL query below does not return identical results on both PostgreSQL and Oracle [14].

```
[h] Q1:SELECT *
FROM ( SELECT S.A, S.A FROM S ) R
```

While it is expected 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. The key idea is to be conducted an experimental evaluation of current DBMSs.

Chapter 3

Methodology

3.1 Methodology

This chapter describes the primary Methodology that is used to test the SQL-Compliance of current DBMSs.

As it was mention in the first chapter, the key idea of the current research is to examine whether current DBMSs comply according to SQL standards and highlight the main differences among those systems. More precisely, it will be examined five popular systems such as MySQL, IBM DB2, Microsoft SQL Server, PostgreSQL and Oracle Database. Aside from that this research has as target to provide a complete architecture that can be used to systematically inspect the SQL-Compliance as these systems have been evolved rapidly. The architecture is consisted by three different tools. More precisely, it has been implemented two different tools such as Random SQL Query Generator Tool and the Comparison Tool. The third one which is used to produce random realistic data realistic data is open-source tool namely Datafiller. The data can be generated based on a database schema which is given to the tools as a parameter. Complex data can be generated by specifying some parameter to the tool. The purpose of the SQL generator tool is to generate a diversity of SQL queries by using SQL commands which are defined in the SQL standard. In that way, it can be examined whether answers to SQL queries are the same independently of which DBMS they are executed on. Therefore, the comparison tool is used to identify if the results are identical. A detailed explanation of the internal implementation and structure of those tools is given in the following Chapter. The implementation of the tool is in such a way that a new DBMS or functionality could be added without affecting the rest implementation.The comparison of DBMSs results will be performed using a mainmemory data structure for achieving efficiency. Differences among current DBMSs are automatically documented by the tool. For example, a query may not be executed on one of the DBMSs and raise an error. Hence, this error is logged into the log file that the comparison tool generates.

Chapter 4

Implementation

In this chapter it is described in details the framework that it has been implemented for assess the SQL-compliance for current DBMSs.

4.1 Implementation

The architecture composes by three different tools as their were introduced in the methodology chapter. The first two tools are implemented in Java programming language and the other one is implemented in python programming language which it is an open source project that can generate random realistic data. It is chosen Java programming language since it is very popular and it can be used to build cross-platform systems. The complete architecture consists by twenty java classes.

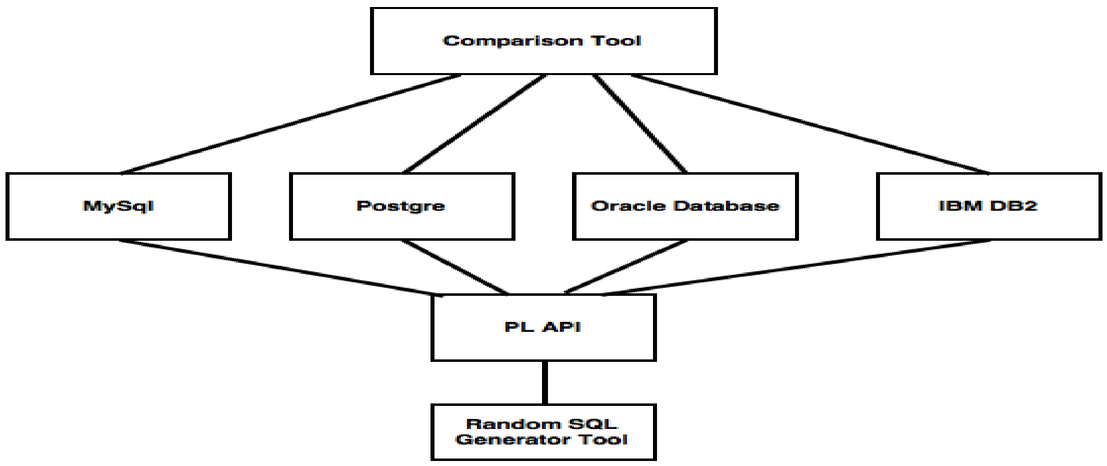

Figure 4.1: Random SQL Generator Architecture

4.1.0.1 Random SQL generator engine

An important component of the architecture is the random SQL generator tool which is used to generate random SQL queries for assessing modern DBMSs. SQL language consists by numerous SQL commands and therefore some of them are simple in terms of their usage such as SELECT, FROM, WHERE where some others are more trickier such as GROUP BY, HAVING or aggregation functions where it is needed to be cared about the proper use of these commands and generating syntactically correct SQL queries. As a result, for achieving generating meaningful and syntactically valid queries, the current implementation of generator tool composes by various java classes specifically seventeen classes, and each class is used for generating different SQL Clause.

In addition, the tool has been designed in such a way that it is modular and reusable and therefore new systems can easily added without the need of changing the whole structure of the tool.

An important decision which should be taken it was with regards to the internal generator tool in order to make it feasible to generate different valid SQL queries and simultaneously syntactically correct. Hence, it is implemented an internal representation and each class is responsible for generating a different SQL clauses that contribute to the overall query. For example, one of the java classes generate the SELECT clause. Having different classes for each SQL statement, 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 tool. The final SQL query is converted to an SQL string which then is executed to the current DBMSs with the contribution of the comparison tool. An important note is that it is not feasible to generate SQL strings directly because it is needed to track attributes names and types. If it was being generated just strings, it would 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. Thusly, there is a need to track attributes for each clause and to achieve that they have been used to main data structure such as LinkedList and HashMaps.

4.1.0.2 Configuration file

The generator tool consists also by a configuration file for partially control the random SQL queries and a detailed explanation is given as follow: The configuration file is used to provide information to the generator tool. Therefore, many parameters can be specified from the configuration file. Below, it is provided firstly the format of the configuration file and subsequently a comprehensive explanation is given for every parameter.

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.

All the parameters are described as follow:

- **relations and attributes** = parameters are used to provide to the generator tool the tables and columns for generating SQL queries according to the database schema. Albeit this approach works pretty well, it makes our tool not portable. Image if the DBMS has a large number of tables with many columns. Hence, the current architecture had the capability to automatically retrieve the whole database schema from DBMSs by just providing the credential in the configuration file by configure user, pass and dbName parameters.

- **MaxTablesFrom** = parameter is used to set an upper bound of the number of tables that an FROM clause can have. If the upper bound is greater than the total number of tables in the schema, then the upper bound is automatically defines the the total number of tables.

- **MaxAttrSel** = parameter indicates an upper bound of projections that the Select clause can have. In other words, it is the total number of attributes that can be selected in the Select clause.

- **MaxCondWhere** = parameter represents the total number of comparison that the Where clause can have.

- **ProbWhrConst** = parameter represents the probability of having comparisons with constant or Null in the Where clause. Therefore, a number between 0 and 1 can be given for this parameter.

- **RepAlias** = parameter indicates the probability of having repetition of alias in the Select clause.
- **NestLevel** = parameter represents the maximum level of nesting that a query can have. 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.
- **ArithCompSel** = parameter represents the probability of having arithmetic operations in the Select clause.
- **Dinstinct** = parameter represents the probability of appearing the distinct SQL statement in a query.
- **StringInSel** = parameter indicates the probability of projecting an attribute of type string or having string functions.
- **StringInWhere** = parameter represents the probability of having string comparison in Where clause.

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
```

There are plenty of SQL commands that can be used to manipulate the data. Below we summarize all the SQL commands which our random query generator use to generate SQL queries.

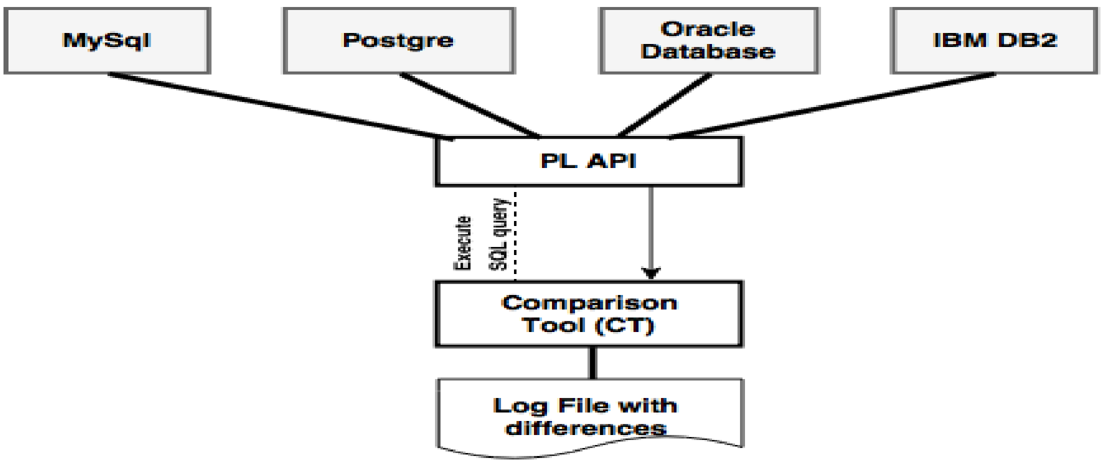

Figure 4.2: Simple Architecture of Comparison Tool

4.2 Comparison Tool

As it was mentioned, the overall objective of the implemented architecture is to be capable of detecting any difference that may exist among current systems. For accomplishing that, aside from the random SQL generator tool, the comparison tool (CT) is implemented as well. Each random SQL query should be executed on five systems. As a consequence, the whole process should be automated and differences that may arise should be documented into a log file. Below, it is illustrated the comparison tool (CT) which has been implemented for achieving the aforementioned purposes and further it is provided a detailed explanation of the internal implementation of the tool.

It has been implemented in Java programming language in order to compare the result of each query on each DBMSs. 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 all the DBMSs. For comparing the results, some challenges had to be overcome. For instance, each DBMS use different algorithms to evaluate each query and it returns the rows of the result in different order or situations with various format. For overcoming the problem of different rows ordering, it is used a data structure such as LinkedList to store the result and then an efficient in-memory sorting algorithm is performed to sort the result. Apparently now, it is expected all the result among the DBMSs to be identical and therefore they can be compared. In the event of a difference is found or a query raises an error, it is documented in the log file. More precisely, the log file contains the SQL query that causes the problem and the exact systems that the difference is found. In addition, if a system raises an error without even to be possible to execute the query,

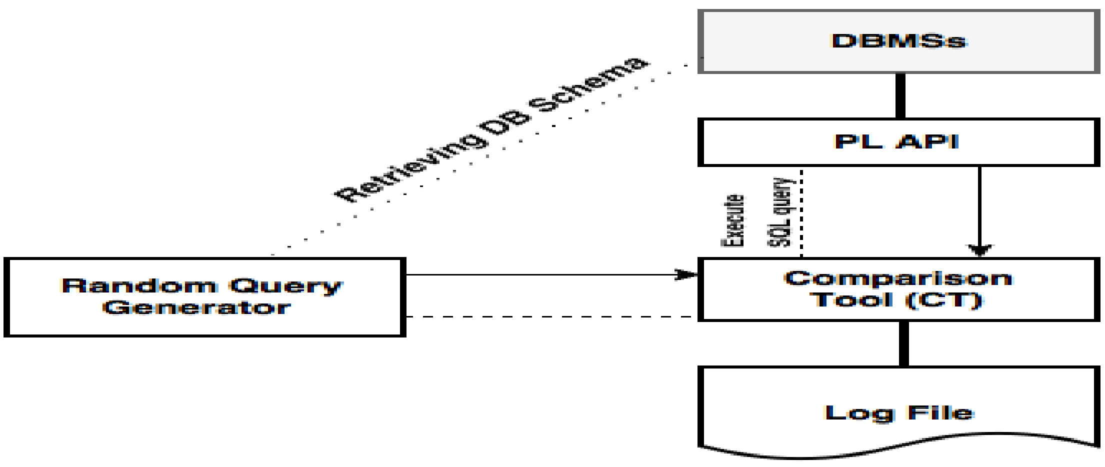

Figure 4.3: Architecture of Comparison Tool

the specific error is recorded with the associated system that raises the error. An important consideration for implementing this tool is how to implementing in a way that in the future new systems can be efficiently added.

As a result, the core idea behind this, it is that there is one method that executes each query and compares the results for any DBMS. Thus, this method takes the connection for accessing the systems and retrieve the result. In the future, if a new DBMS need to be checked can be easily added by providing the connection to this method without changing the internal state of the method. It is illustrated in the experiments chapter that indeed differences exist and in some situations may be significant depends on the context that the queries are used. Also, the tool is an important component of this project as it is used to conduct the experiments and identify the main differences that exist between current DBMSs.

4.3 Generate data

Datafiller is a well-known open-source project that provides the capability of generating random data. For our project data filler is very important as it will be used 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 provided as a parameter, and it takes into account constraints of that schema for generating valid data. 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=rate

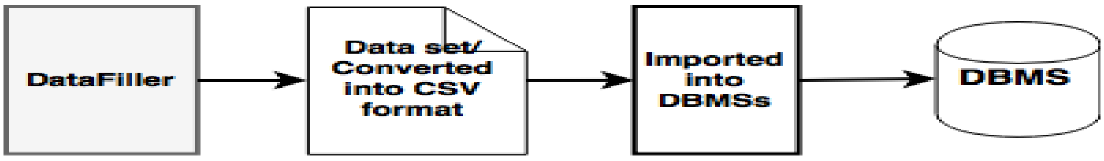

Figure 4.4: Method of importing csv files

Additionally, more complex parameters can be provided as well, such as the 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 schema SQL script. For instance, if there is a field which represents a date, then we can provide a range in order for the datafiller to generate dates only within that 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.

Datafiller supports data importing only into PostgreSQL. Nevertheless, for our project we need to import the random generated data into five DBMSs. For this reason, we convert the data into CSV format, 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

6.0.1 Conclusions

In this project it is implemented an entire architecture that can be used to systematically check the SQL-compliance among DBMSs. Furthermore, it is demonstrated that the implemented architecture is competent to reveal crucial differences in the sense that otherwise they will be difficult to be revealed without using a similar tool. Since now there has not been implemented a similar tool and only some documentations exist which show some obvious issues. The goal of the project has successfully been achieved and more than twenty differences have been disclosed which make clear that a significant number of parts of the Standard is implemented differently.

Demonstrating and analyzing these incompatibilities make aware both users and programmers for these issues. Albeit of the beginning of this project some evidences about some incompatibilities and different interpretations of the Standard were notorious, it was somewhat surprisingly that so many differences have been emerged which makes clear that the standard is difficult to be interpreted and implemented in exactly the same way from all DBMSs. In addition, awareness of these differences issues may cause irreversible consequences in companies if we take into account that such systems are utilized in almost every field. Moreover, it is demonstrated that some DBMSs have more incompatibilities than other which makes the migration process notoriously difficult.

Aside from the issues which have been detected, the implemented architecture is portable and they can extended efficiently. For example, although it is provided experimental evidences for numbers and strings data types, the random generator tool is implemented in such a way that can track any data type such as Date. In that way, it can be extended efficiently to generate queries with attributes of date as data type.

6.0.2 Summary of the findings

6.0.3 Suggestions for future work

Several issues have been arisen by testing various databases with integers and strings. As a future work can be to extend the generator tool to generate queries with attribute of data type date. It is important mentioning that the current implementation can hold any data type which make the specific extension quite simple and apparently more issues will be arise using this data type as well. Yet another future extension should be to include new DBMSs for testing the SQL-compliance of them. This extension also will not be quite challenging as the way that the comparison tool is implemented makes the process of adding new systems easy.

Bibliography

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

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

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

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

Ramakrishnan, R. (2000). Database management systems . pdf.

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