

# 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Motivation . . . . .	2
1.3	Related Work . . . . .	3
1.4	Thesis Structure . . . . .	3
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	The SQL Standard . . . . .	5
2.2	The SQL Language . . . . .	6
2.3	Commands of SQL . . . . .	6
2.3.1	Missing values . . . . .	11
2.4	SQL standard issues . . . . .	13
2.5	The Database Management Systems . . . . .	14
<b>3</b>	<b>Methodology</b>	<b>15</b>
3.1	Methodology . . . . .	15
<b>4</b>	<b>Implementation</b>	<b>17</b>
4.1	Implementation . . . . .	17
4.1.1	Random SQL generator engine . . . . .	18
4.1.2	Configuration file . . . . .	18
4.2	Comparison Tool . . . . .	21
4.3	Generate data . . . . .	22
<b>5</b>	<b>Experimental Evaluation</b>	<b>24</b>
5.1	The experiment Set-Up . . . . .	24
5.2	Experiment Results . . . . .	25
5.2.1	Dif 1 . . . . .	25

5.2.2	Dif 2 . . . . .	26
5.2.3	Dif 3 . . . . .	27
5.2.4	Dif 4 . . . . .	27
5.2.5	Dif 5 . . . . .	28
5.2.6	Dif 6 . . . . .	29
5.2.7	Dif 7 . . . . .	30
5.2.8	Dif 8 . . . . .	30
5.2.9	Dif 9 . . . . .	31
5.2.10	Dif 10 . . . . .	33
5.2.11	Dif 11 . . . . .	33
5.2.12	Dif 12 . . . . .	34
5.2.13	Dif 13 . . . . .	34
5.2.14	Dif 14 . . . . .	37
5.2.15	Dif 15 . . . . .	37
5.2.16	Dif 16 . . . . .	38
5.2.17	Dif 17 . . . . .	39
5.2.18	Dif 18 . . . . .	40
5.2.19	Dif 19 . . . . .	40
5.2.20	Dif 20 . . . . .	41
5.2.21	Dif 21 . . . . .	42
<b>6</b>	<b>Conclusions</b>	<b>43</b>
6.1	Conclusions . . . . .	43
6.2	Summary of the findings . . . . .	44
6.3	Suggestions for future work . . . . .	45
	<b>Bibliography</b>	<b>46</b>

# 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 care of 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[1].

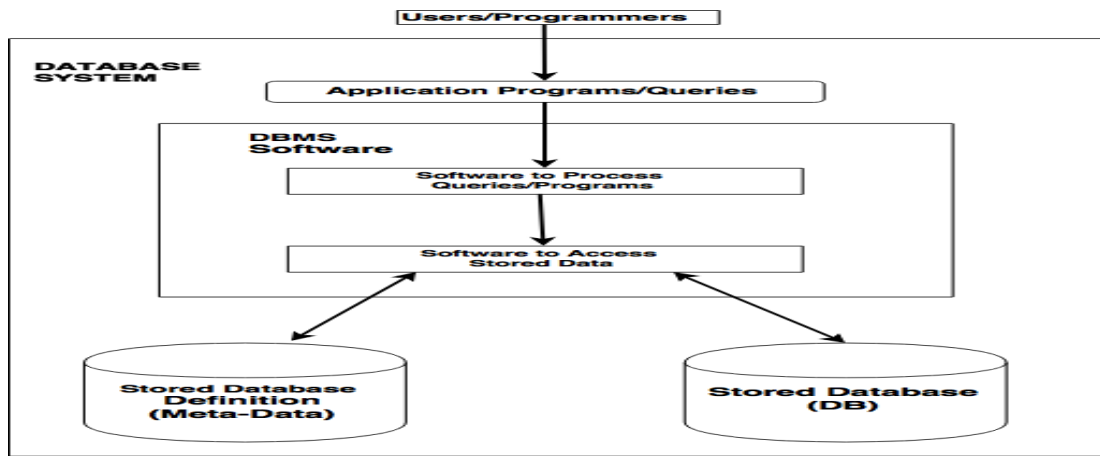


Figure 1.1: Abstract DBMS architecture

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 extent 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 the main issues are introduced by illustrating problematic SQL queries.

- **Chapter 3** describes the main methodology for this research and briefly explains 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 individual tool that the architecture is composed of. Also, it briefly discusses the main issues that are addressed.
- **Chapter 5** illustrates the main findings and provides an explanation about the differences between current DBMSs with appropriate reference to the SQL standard.
- **Chapter 6** concludes this project with explanation of what is been achieved, and a summarised table of all the differences is provided. Afterwards, futures ideas and extensions are provided.

# 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 emerged from various vendors and inevitably, a standardised language should be implemented in order to provide portability among current 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.

The first appearance of the SQL language was in 1970 where IBM developed the first prototype of Relational Database Management System (RDBMS). Subsequently, the first SQL standard arose in 1986 by the 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 the International Standards Organization (ISO) [11]. The 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 focuses 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 part 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 composed of primarily two sublanguages, the data definition language (DDL) and the 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 gives the capability of defining new domains. Moreover, in general, tables and rows are denoted as relations and tuples. DML is also a part of the SQL language that is composed of 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, the SQL Standard supports more complex rather than just 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 column in order to return a value, for example the calculation of the average salary of a department by performing 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 section intends to provide a high-level description of the basic commands of SQL, as defined in the Standard. As these commands are used to generate random queries, a basic background of the purpose of each command is given.

### SQL Basic Structure

```
SELECT [DISTINCT] columns_list  
FROM tables_list  
WHERE Condition1 {AND|OR} Condition 2
```

The SQL basic structure represents the basic SQL commands which are used to retrieve data from a database based on some criterias that are specified in the WHERE clause. Each SQL query should have at least SELECT and FROM clause. The WHERE clause is optional. The basic SQL query is executed as follow: all the rows of the tables list in the FROM Clause are evaluated. Each row that satisfied the search criteria is selected. Then, only the specified columns of the selected rows appear in the result. In addition, the DISTINCT keyword is optional and can used to eliminate duplicates rows in the result. Also, instead of having columns\_list in the SELECT clause, the \* keyword can be used which indicates that all the columns of the tables\_list will appear in the result. The purpose of the WHERE clause is to filter the results and normally SQL queries contains a WHERE clause. Conditions compare expressions using comparison operators. A comparison operator is used to compare two expressions and logical connectivities, such as AND, NOT and OR are used to connect the conditions.

#### Example of basic query:

```
SELECT St.Student_Name
FROM Students AS St
WHERE St.age >= 20 AND St.age <= 24
```

The above example uses the basic SQL structure to build a query. Thus, it retrieves all the Students and based on the criteria of the WHERE clause, it returns 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
```

SQL query with aggregation is used to perform a calculation on specific columns in order to return a value. GROUP BY and HAVING are used to perform an aggregation.

HAVING is an optional clause, where aggregation can still be performed using aggregate commands in the SELECT clause. A concrete example is given subsequently.

Table 2.1: Aggregation Commands

Operator	Use
MIN()	Finds the minimum value of a column
COUNT()	Returns true, if all the comparisons the operator OP return true
MAX()	Returns true, if at least one comparison the operator returns true
SUM()	Returns true, if an element exist in a given set
AVG()	Calculates the average of values of a column

Aggregate commands can be used both in SELECT and HAVING clause in combination with the GROUP BY clause. The example below illustrates the proper use of aggregation commands.

**Example:**

```
SELECT COUNT(St.student_id), St.Country
FROM Students AS St
GROUP BY St.Students
HAVING COUNT(St.student_id) > 3
```

The above query makes proper use of the aggregation commands. More precisely, it lists the number of students in each country where there are more than three students in a specific Country.

Table 2.2: Logical Operators

Operator	Use
EXISTS	Returns true, if there is at least one row in the subquery
Op ALL	Returns true, if all the comparisons the operator OP return true
Op ANY	Returns true, if at least one comparison the operator returns true
Op IN	Returns true, if an element exist in a given set

**Example:**

```

SELECT  *
FROM Students AS St
WHERE St.Country IN ( UK , Netherland )

```

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

Table 2.3: SET Commands

Command	Use
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, appear in both results
EXCEPT [ALL]	Return each row that appear to the first query but does not appear to the second query

By default SET commands remove duplicates in the SQL result. Nevertheless, we can have duplicates in the result by adding the optional ALL keyword for any of the commands above.

**Example:**

```

SELECT Country FROM Students
UNION
SELECT Country FROM Professor

```

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

Table 2.4: String Commands

Operator	Use
TRIM()	Returns the string without leading/trailing characters
SUBSTRING()	Retrieves a subset of the initial string
CONCAT()	Concatenate two or more strings
REPLACE()	Replaces a subset of a string with another string
LIKE	Returns true, if an attribute matches with a pattern

**Example:**

```
SELECT
SUBSTRING ( SQLSTANDARD , 1, 3 ) AS SQLExtraction
```

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

Table 2.5: Data Type

Types	Description
SMALLINT	Can store a number from a range between -32768 and 32767
INT	Can store a number from a range between -2147483648 and 2147483647
BIGINT	Can store a number from a range between -9223372036854775808 and 9223372036854775807
VARCHAR	Takes as input the length of a variable string which can contains up to 255 characters
CHAR	Takes as input the length of a fixed size string which can contains up to 255 characters

**Operator Precedence**

Operator precedence is an important concept for understanding how an SQL query is evaluated and it is also important for determining if major DBMSs follow the same operator precedence. There are cases where the expressions in the WHERE clause are quite complicated and operator precedence define in which order the expression should be evaluated. The sequence of operators is provided as follow, starting with higher priority and ending with lower priority:

- ()
- +, -, ,
- \*, /,
- =, <, >, <=, >=, <>



- 
- NOT , AND
- ALL , ANY , IN , LIKE
- = - variable assignment.

Operators that have the same priority are evaluated from left to the right. In addition, parenthesis abrogate the priority of the rest operators and as a result expressions that enclosed by a parenthesis are evaluated first.

### 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 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.

Table 2.6: Three-Valued logic truth table

Y	Z	Y OR Z	Y and Z	NOT Y
True	True	True	True	False
True	False	True	False	False
True	Unknown	True	Unknown	False
False	True	True	False	True
False	False	False	False	True
False	Unknown	Unknown	False	True
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

In an SQL query each tuple which is evaluated to true is returned in the result, and tuples that are evaluated to false or unknown are not returned. Comparisons that involving NULLs are considered as False in the WHERE Clause. Hence, If we take into consideration that only Oracles db treats empty string as NULL then it can be realized that many problem can be arised.

### Examples

**NULL = 10** is evaluates to **Unknown**

**NULL = NULL** is evaluates to **Unknown**

**NULL <= 3** is evaluates to **Unknown**

```
SELECT St.Student_Name
FROM Students AS St
WHERE NULL <= 20
```

The above query will result to an empty set as for each row the WHERE clause will be evaluated to Unknown and as a consequence none of the rows will be appeared in the result.

```
SELECT St.Student_Name
FROM Students AS St
WHERE NULL = NULL
```

Logically it is expected that the NULL equal NULL should be evaluate to true. Nevertheless, the WHERE clause is evaluate to Unknow for every row, and the result of this query is an empty set.

## 2.4 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] Query 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.

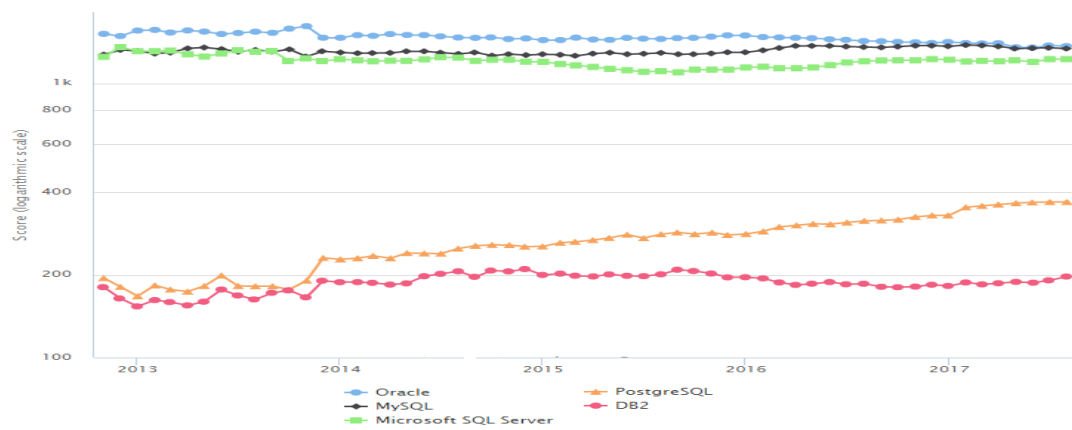


Figure 2.1: Popularity of modern DBMSs

## 2.5 The Database Management Systems

**Microsoft SQL Server:** is a relational database management systems (RDBMS) developed by Microsoft. Microsoft offers different edition of its product such as Standard, Enterprise and Express. Express edition is a free edition of the SQL Server. It is mainly available on Windows.

**MySQL:** is an open-source RDBMS which is freely available which is owned by Oracle. MySQL is used by huge companies such as Google, Facebook and Youtube. It is freely available on MacOS, Linux and Microsoft Windows

**PostgreSQL:** is an open-source RDBMS which is trying to be SQL-Compliance and it focuses on portability. It is freely available both on Microsoft Windows and Linux.

**IBM DB2:** is a RDBMS developed by IBM and it is available on Linux and Microsoft Windows.

**Oracle Database:** is a RDBMS developed by Oracle Corporation. It is available on MacOS, Linux and Microsoft Windows.

The Figure 2.1 illustrates the most popular DBMSs from 2013 to 2017 using a calculated score based on the Google Trend, number of mentions of DBMSs on the web and discussion of the systems in Stackoverflow and DBA Stack Exchange. It can be seen that the popularity of Oracle database, MySQL and Microsoft SQL Server remain constant over the years with a relatively high score. In addition, there is a stable rise for PostgreSQL in the last three years. Lastly, IBM DB2 popularity has decreased slightly in the last year.

# 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 main-

memory 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.

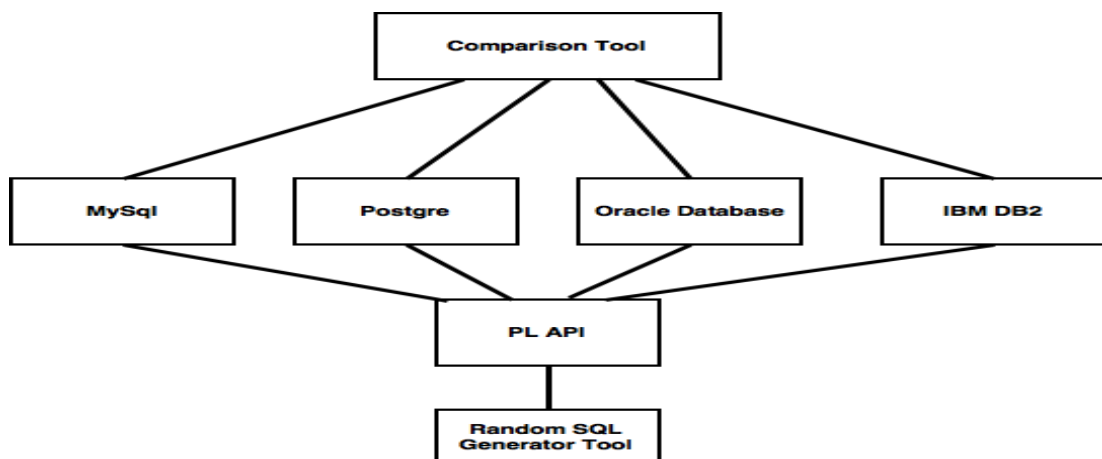


Figure 4.1: Random SQL Generator Architecture

### 4.1.1 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.

### 4.1.2 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.

This configuration file will be used to give various parameter to the SQL Engine

Maximum number of tables in the FROM STATEMENT  
`maxTablesFrom = 3`

Maximum number of attributes in the SELECT STATEMENT  
`maxAttrSel = 5`

Maximum number of comparisons in the WHERE STATEMENT  
`maxCondWhere = 7`

Represents the probability of having constants or NULL comparison in the WHERE STATEMENT  
`probWhrConst = 0.8`

`nestLevel = 4`

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.

## 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

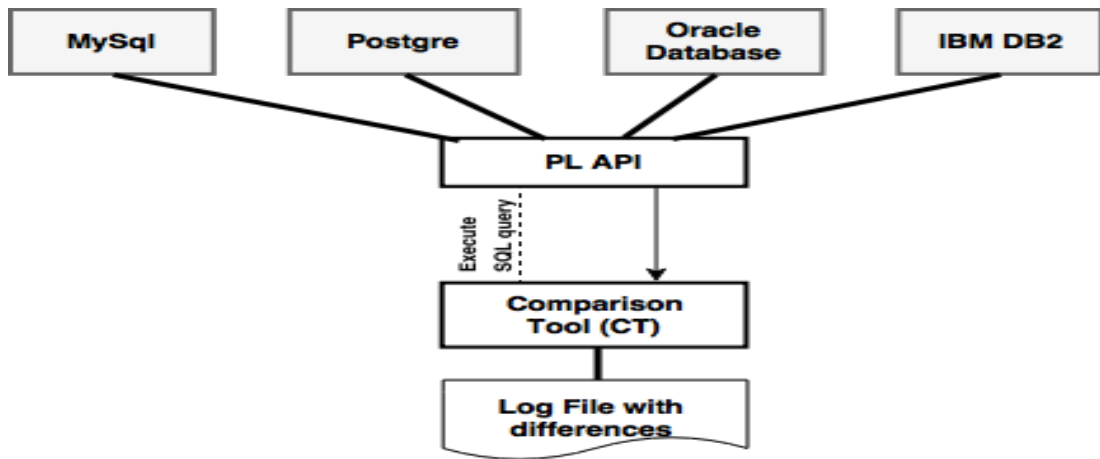


Figure 4.2: Simple Architecture of Comparison Tool

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, 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 gener-

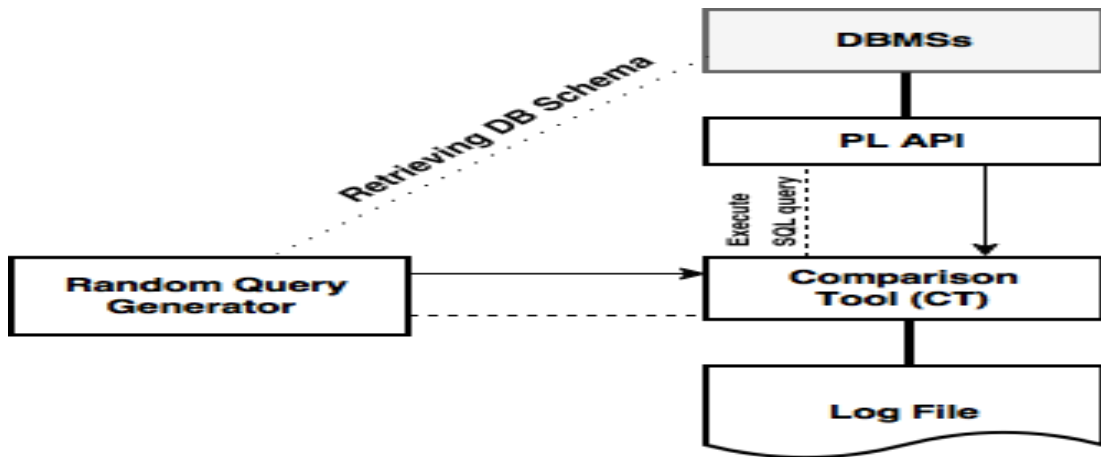


Figure 4.3: Architecture of Comparison Tool

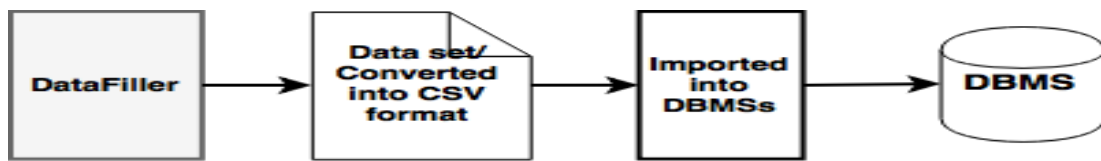


Figure 4.4: Method of importing csv files

ating 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

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

This chapter presents the procedure that it is followed in order to provide the experimental evidences. Subsequently, It is presented the main findings with appropriate explanation according to the Standard. Also, it is described the environment and the DBMSs that they have been tested.

### 5.1 The experiment Set-Up

The evaluation is carried out on Windows 8 with i7 CPU, 12GB Ram and a solid state disk (SSD). Also, the following versions of DBMSs have been installed: PostgreSQL Version 9.6, Microsoft SQL Server Express Edition 2016, IBM Db2 Express-C, Oracle Database 12c and MySQL Community Edition 5.7. Apart of generating huge numbers of SQL queries, it is also checked if new findings can be emerged by conducting the experiments in different schemas with different data types and numbers of relations. As a result, it is checked the following data types: TEXT, CHAR, VARCHAR, INTEGER, SMALLINT, BIGINT. Moreover, the experiments are conducted in different schemas varying from two to ten relations with two to ten attributes. In addition, it is generated data using different rate of NULLs such as 5%, 10%, 20% and 40% for identify if differences can arise depending on the number of NULLs that a database contains. Furthermore, the size of instances varied from two to one thousand rows which were generating using data filler. Having huge instances decrease the throughput of queries that can be executed per second and thus less queries will be checked. Taking into account that issues and different interpretations can be raised even in small instances, it is end up realized that small instances and schema can still reveal crucial differences.

## 5.2 Experiment Results

The implemented architecture is used to conduct the experiments and in that way it is also checked whether the current implementation is capable to detect any differences among modern DBMSs. The results are provided below with the following format: Firstly, It is presented the SQL query that cause an error or a semantic issue on one or more systems. Meaning that some features may not be implemented by all systems or use a different commands or interpret it differently. Then, a table for each query is provided that demonstrates a raised error of any system, otherwise, the keyword works meaning that the SQL query is executed without to raise any error. Thereafter, a comprehensive explanation is given for each problematic query by explaining about the source of the problem and giving an explanation according to the SQL standard.

### 5.2.1 Dif 1

**Q1:**

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

Table 5.1: Difference 1

DBMS	Result Message
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
IBM DB2	Works

According to the Standard each expression in the WHERE clause is evaluated to a boolean type such as True, False and Unknown. Thus, each row is evaluated to a boolean type and rows that returned true are included in the result. The Q1 performs multiplication between two constants in the WHERE clause and cannot be evaluated to

a boolean type, instead it should return an integer type. It is expected that the Q1 will raise an error if it is executed, nevertheless, Mysql and IBM DB2 execute the query without to raise any error. Although the expression in the WHERE clause should return a boolean type, these two DBMSs convert an integer type to a boolean type. On the contrary with the rest three DBMSs which throw an exception.

### 5.2.2 Dif 2

**Q2:**

```
SELECT r21.A AS A0
FROM r2 AS r21
WHERE true
```

Table 5.2: Difference 2

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	[S0001][4145] An expression of non-boolean type specified in a context where a condition is expected, near 'true'
Oracle	[42000][920] ORA-00920: invalid relational operator
IBM DB2	Works

As it is aforementioned, each expression in the WHERE clause is evaluated to a boolean type. Thus, instead of having a specific expression, the keywords True/false can be specified which is the type that each expression is evaluated. As a consequence, it is reasonable to have the True keyword in the WHERE clause indicating that each row selected in the FROM clause will be included in the results. Nevertheless, Q2 raised an error if it is executed on Microsoft SQL Server and Oracle DBMSs, in the contrast with the rest DBMS which the query is executed without to raise an error. With respect to SQL standard there is not explicit mention about the keyword true in the WHERE Clause.



### 5.2.3 Dif 3

**Q3:**

```
SELECT  NULL/NULL, 1/2, NULL-NULL
FROM  r2 AS  r21, r4 AS  r41
WHERE  r41.B > r21.B
```

Table 5.3: Difference 3

DBMS	Result Message
Mysql	Works
PostgreSQL	[42725] ERROR: operator is not unique: unknown / unknown Hint: Could not choose a best candidate operator. You might need to add explicit type casts.
Microsoft SQL Server	Works
Oracle	Works
IBM DB2	Works

Boolean data type comprises the distinct truth values True and False. Apart from these values, boolean data type supports also the truth value Unknown as the NULL value. As a result, the Standard does not make a distinction between NULL value of the boolean data type and the truth value Unknown and based on the Standard a division between NULL should result to a NULL. The evaluation of the NULL/NULL in the Q3 should result to Unknown value. Nevertheless, PostgreSQL raises an error when executes this query, in the contrary with the rest DBMSs which are executed the query without any error.

### 5.2.4 Dif 4

**Q4:**

```
SELECT  r21.B/3
```

```
FROM  r2 AS r21, r4 AS r41
WHERE NOT (NOT (r41.A <> 18 ) )
```

Table 5.4: Difference 4

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	Works
IBM DB2	Works

The Q4 is executed on Databases that contain smallint data type for the attribute r21.B. Thus, this query is executed without cause an error on any of five DBMSs. Nevertheless, the results differ slightly in terms of their return type. In DB2, PostgreSQL and MS Server the results for the column r21.B/3 are returned as integer type, on the contrary with MySQL and Oracle DBMSs where the results are returned as decimal data type.

### 5.2.5 Dif 5

**Q5:**

```
SELECT ( MIN (r41.B) % AVG (r41.A) )
FROM  r4 AS r41
WHERE (10 >= 19 )
GROUP BY r41.A, r41.B
```

An important consideration is that arithmetic operations such as addition, multiplications and subtraction are supported in the SELECT clause. Another important arithmetic operation which is also supported in the SELECT clause is the (modulo). Nevertheless, the syntax of % (modulo) operator differ in Oracles database. More precisely, Oracle database instead of using % which is supported by the rest DBMSs, it uses a function called MOD. Thus, the Q5 needs to replace the operator % with function mod in order to be syntactically correct in Oracles db, that is, MOD( MIN(r41.B), AVG(r41.A) ).

Table 5.5: Difference 5

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	[22019][911] ORA-00911: invalid % character
IBM DB2	Works

### 5.2.6 Dif 6

**Q6:**

```
SELECT r41.A AS A0
FROM r4 AS r41
```

Table 5.6: Difference 6

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	[42000][933] ORA-00933: SQL command not properly ended
IBM DB2	Works

With respect to the Standard, alias feature is optional and it is not compulsory to be implemented in the modern DBMSs. The purpose of this feature is to rename tables or columns. Thus this feature can be used both with attributes in the **SELECT** clause and for tables in the **FROM** clause. Aliases are given using **AS** keyword. In addition, it can be used to rename a subquery in the **FROM** clause and subsequently to access it using its alias. The Oracles database support to use **AS** when defining column aliases, but it does not allow use **AS** when defining table aliases (in the **FROM** clause), on the contrary with the rest DBMSs which support the alias feature both to rename tables or columns.

### 5.2.7 Dif 7

**Q7:**

```
(SELECT r41.A AS A0
FROM r4 AS r41 )
EXCEPT ALL
(SELECT r21.A AS A0
FROM r2 AS r21, r4 AS r42 )
```

Table 5.7: difference 7

DBMS	Result Message
Mysql	[42000][1064] You have an error in your SQL syntax; check the manual that,corresponds to your MySQL server version for the right syntax to use near 'EXCEPT ALL
PostgreSQL	Works
MS Server	S0002][324] The 'ALL' version of the EXCEPT operator is not supported.
Oracle	[42000][933] ORA-00933: SQL command not properly ended
IBM DB2	Works

EXCEPT ALL is an optional feature according to the Standard. The usage of this feature is to return all rows from the outer query which are not present in the inner query without removing the duplicates. Thus, this operator is fully supported by the Standard but it is optional meaning that it is not compulsory to be implemented by modern DBMSs. MySql and Oracle database do not support this operator. Nevertheless, Oracle, supports the same operator but with different name called MINUS ALL keyword. As a result, MINUS ALL has exactly the same behaviour on Oracles database with the EXCEPT ALL. Lastly, PostgreSQL and IBM DB2 support the EXCEPT ALL operator.

### 5.2.8 Dif 8

**Q8:**

```

(SELECT r41.A AS A0
 FROM r4 AS r41, r2 AS r21, r3 AS r31
 WHERE NOT (r31.B <> r41.A ) )
EXCEPT
(SELECT r21.A AS A0
 FROM r2 AS r21, r4 AS r41
 WHERE r41.A <> r41.B )

```

Table 5.8: Difference 8

DBMS	Result Message
Mysql	[42000][1064] You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'EXCEPT'
PostgreSQL	Works
MS Server	Works
Oracle	[42000][933] ORA-00933: SQL command not properly ended
IBM DB2	Works

EXCEPT is a mandatory feature according to the Standard and all DBMSs must support this feature in order to comply to the Standard. The usage of this feature results to return all rows from the outer query which are not present in the inner query with removing the duplicates. Only MySQL does not support the EXCEPT even though it is a compulsory feature according to the Standard.. Nevertheless, Oracle database instead of use EXCEPT as keyword, it uses MINUS which has an identical behavior. The rest DBMSs support this operator.

### 5.2.9 Dif 9

Q9:

```

(SELECT r41.A AS A0
 FROM r4 AS r41, r3 AS r31
 WHERE (NULL <= 6 OR NOT(r31.B <> r41.A ) ) )
INTERSECT ALL
(SELECT r21.A AS A0
 FROM r2 AS r21, r4 AS r41
 WHERE r41.A <> r41.B OR ( 0 <> 14) AND r41.A > r21.B)

```

Table 5.9: Difference 9

DBMS	Result Message
Mysql	[42000][1064] You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ALL
PostgreSQL	Works
MS Server	[S0001][324] The 'ALL' version of the INTERSECT operator is not supported.
Oracle	[42000][928] ORA-00928
IBM DB2	Works

With respect to the Standard INTERSECT ALL is an optional feature. The usage of this feature is to return all the rows which are presented in both inner and outer queries result, and without removing duplicates. MySQL, Microsoft SQL Server and oracle do not support this feature at all. On the contrary, PostgreSQL and IBM DB2 support this feature.

### 5.2.10 Dif 10

**Q10:**

```
SELECT 7/0 AS ART0, 1%NULL AS ART1
FROM r1 AS r11
WHERE (NULL = 19 AND r11.b <> 5)
```

Table 5.10: Difference 10

DBMS	Result Message
Mysql	Works
PostgreSQL	[22012] ERROR: division by zero
MS Server	[S0001][8134] Divide by zero error encountered.
Oracle	ORA-01476: divisor is equal to zero
IBM DB2	[22012][-801] Division by zero was attempted..

A division with zero should not allowed according to the Standard and it should always raise an error. Nevertheless, MySQL does not raise an error and the result of the division with zero is NULL.

### 5.2.11 Dif 11

**Q11:**

```
SELECT r11.a AS A1, r11.b AS A2
FROM r1 AS r11
WHERE (r11.b, r11.a) IN (SELECT r12.a AS A4, r12.b AS A3
FROM r1 AS r12)
```

Table 5.11: Difference 11

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	[S0001][4145] An expression of non-boolean type specified in a context where a condition is expected, near ','.
Oracle	Works
IBM DB2	Works

A division with zero should not allowed according to the Standard and it should always raise an error. Nevertheless, MySQL does not raise an error and the result of the division with zero is NULL.

### 5.2.12 Dif 12

**Q12:**

```

SELECT  r31.b AS A1
FROM    r3 AS r31
WHERE    r31.a >= r31.a
GROUP BY r31.a

```

According to the Standard, only columns that appear in the GROUP BY clause can be selected. The intuition behind this issue is based on the fact that if non-grouped and non-aggregate fields are selected, then the DBMSs cannot know which field to return. The Q12 project the attribute r31.b which is not appear in the the GROUP BY clause. PostgreSQL, Microsoft SQL Server, Oracle and IBM DB2 raise an error as the attribute does not appear in the GROUP BY clause. Nevertheless, MySQL executes the query without to raise an error.

### 5.2.13 Dif 13

**Q13:**



Table 5.12: Difference 12

DBMS	Result Message
Mysql	Works
PostgreSQL	[42803] ERROR: column "r31.b" must appear in the GROUP BY clause or be used in an aggregate function Position: 9
MS Server	[S0001][8120] Column 'r3.B' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
Oracle	[42000][979] ORA-00979: not a GROUP BY expression
IBM DB2	[42803][-119] An expression starting with "B" specified in a SELECT clause, HAVING clause, or ORDER BY clause is not specified in the GROUP BY clause or it is in a SELECT clause, HAVING clause, or ORDER BY clause with a column function and no GROUP BY clause is specified

```

SELECT  NULL+NULL AS  ART1
FROM  r4 AS  r41, r5 AS  r51
WHERE   r41.a >= r41.a OR ( r41.a >= 4 )
GROUP BY r41.a
HAVING MIN(r41.a) < 7506

```

Table 5.13: Difference 13

DBMS	Result Message
Mysql	Works
PostgreSQL	[42725] ERROR: operator is not unique: unknown + unknown Hint: Could not choose a best candidate operator. You might need to add explicit type casts
MS Server	Works
Oracle	Works
IBM DB2	Works

The Standard does not make a distinction between NULL value of the boolean data type and the truth value Unknown and based on the Standard an addition between

two NULLs should result to a NULL. The evaluation of the NULL+NULL in the Q13 should result to Unknown value. Nevertheless, PostgreSQL raises an error when executes this query, in the contrary with the rest DBMSs which are executed the query without any error.

### 5.2.14 Dif 14

**Q14:**

```
SELECT  'SQL' || 'STANDARD'
FROM R1
```

Table 5.14: Difference 14

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	[S0001][102] Incorrect syntax near '—'.
Oracle	Works
IBM DB2	Works

According to the Standard, concatenation is a mandatory feature and it should be supported by all DBMSs with a double-pipe mark —. The purpose is to concatenate two or more strings into one. Thus, executing Q14 is expected the result to be SQL-STANDARD. Nevertheless, MySQL supports the double pipe operator, but it treats the double-pipe — as a logical OR and thus the query returns 0. MySQL uses the CONCAT() function as concatenation and it takes as parameters one or more strings. Microsoft SQL Server uses the + operator in order to perform the concatenation. Oracle and PostgreSQL support the double-pipe — concatenation operator.

### 5.2.15 Dif 15

**Q15:**

```
SELECT  *
FROM   r1 AS R1 , r1 AS R1
```

Q15 should raise an error because it give the same alias on two tables. Thus, it will be unambiguous to which table it is referred. Nevertheless, the query is executed on IBM DB2 database, where on the rest DBMSs raise an error.

Table 5.15: Difference 15

DBMS	Result Message
Mysql	[42000][1066] Not unique table/alias: 'R1'
PostgreSQL	[42712] ERROR: table name "r1" specified more than once
MS Server	[S0001][1011] The correlation name 'R1' is specified multiple times in a FROM clause.
Oracle	[42000][933] ORA-00933: SQL command not properly ended
IBM DB2	Works

### 5.2.16 Dif 16

#### Q16:

```
SELECT TRIM ( ' _____SQLSTANDARD_____ ' )
FROM R1 ;
```

Table 5.16: Difference 16

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	[S00010][195] 'TRIM' is not a recognized built-in function name.
Oracle	Works
IBM DB2	Works

According to Standard trim function is a mandatory feature and it should be implemented by all DBMSs. The usage of trim feature is that it returns the string which is given as argument with leading and/or trailing pad character. This function is supported by most of DBMSs, except Microsoft SQL Server.

### 5.2.17 Dif 17

**Q17:**

```
SELECT  2 * 5 AS ART
WHERE   ( 1 = 1 )
WHERE  r1.c3 LIKE 'standard%'
```

Table 5.17: Difference 17

DBMS	Result Message
Mysql	[42000][1064] You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'WHERE (1 = 1 )
PostgreSQL	Works
MS Server	Works
Oracle	[42000][923] ORA-00923: FROM keyword not found where expected
IBM DB2	[42601][-104] Expected tokens may include: "FROM"

The basic structure of an SQL query is SELECT...FROM...WHERE. The Q17 omits the FROM clause. SQL query without FROM clause can be executed only on PostgreSQL and Microsoft SQL Server. On the contrary, Mysql, Oracle and IBM DB2 raises an error.

**5.2.18 Dif 18****Q18:**

```

SELECT  SUBSTRING ('Standard', 1, 4)
FROM    R1

```

Table 5.18: My caption

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	[42000][904] ORA-00904: "SUBSTRING": invalid identifier
IBM DB2	Works

The substring function is a mandatory feature according to the Standard and as a result all DBMSs should implement this feature. The usage of substring is to return a part of the string which given as argument. Mysql, PostgreSQL, IBM DB2 and Microsoft Sql Server support this function. Oracle database use Substr instead of Substring which has a similar behaviour. The prototype of oracles function is as follow: *substr(column<sub>n</sub>ame, start<sub>p</sub>os, no<sub>o</sub>f characters)*.

**5.2.19 Dif 19****Q19:**

```

SELECT  "SQLSTANDARD"
FROM    R1

```

Table 5.19: Difference 19

DBMS	Result Message
Mysql	Works
PostgreSQL	[42703] ERROR: column "SQLSTANDARD" does not exist Position: 8
MS Server	[S0001][207] Invalid column name 'SQLSTANDARD'.
Oracle	[42000][904] ORA-00904: "SQLSTANDARD": invalid identifier
IBM DB2	[56098][-727] An error occurred during implicit system action type "2"

According to the SQL standard encompass by . It is demonstrated in Q19 that MySQL allow a string to be encompassed by both `'` and `"` which make the SQL code less portable as the rest DBMSs raise an error if it is used `"` instead if `'`.

### 5.2.20 Dif 20

Q20:

```
SELECT *
FROM r1 AS R1
WHERE R1.c3 =
```

Table 5.20: Difference 20

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	Works
IBM DB2	Works

The Q20 is executed on all DBMSs without to raise an error, though the semantic of this query differs. As it was mentioned in the background chapter, and more precisely in the missing value, Oracle database treats the empty string as NULL, on the contrary with the rest DBMSs, which treats it as a normal String. Thus, all the comparisons in

the WHERE clause involving NULL are evaluated to Unknown which implies that the result will be empty as none of the rows will be satisfied. On the other hand, if there is at least a row which contains an empty string by executing the above query will be returned in the result. All the DBMSs except Oracle Database return in the result all the rows that contains an empty string in the column c3 and Oracle Database return an empty result.

### 5.2.21 Dif 21

**Q21:**

```
SELECT *
FROM R1
WHERE r1.c3 LIKE 'standard%'
```

Table 5.21: Difference 21

DBMS	Result Message
Mysql	Works
PostgreSQL	Works
MS Server	Works
Oracle	Works
IBM DB2	Works

Q21 is executed on all DBMSs without to raise an error but it differs semantically. This query will not return the same results on all DBMSs. More precisely, all the tested systems contain a database that stores a field with the word STANDARD (in capital letters) in the attribute c3 of the relation r1. By executing this query, it can be observed that some of the DBMSs are case sensitive with the LIKE operator. PostgreSQL and Oracle are case-sensitive, resulting to return an empty set. On the contrary with the rest systems, which returns one row which is the row that contains the word STANDARD in the attribute c3.



# Chapter 6

## Conclusions

### 6.1 Conclusions

In this project an entire framework is implemented composed by the random query generator tool and the comparison tool which are used to evaluate the SQL-compliance of five DBMSs. Also, we verified the correctness of the implemented tools by conducting the experiments and we demonstrated from the experiment evaluation chapter that the implemented tools are competent to reveal crucial differences and issues among current systems. Without a similar framework, it would be almost impossible to detect some of the differences and issues by generating queries manually, or by testing all DBMSs empirically. Furthermore, as described in the related work, there is no similar framework, except of some documentations provided by the vendors of such systems and some other studies which presented some issues according to the Standard without having a systematic tool.

In addition, a summarized table is provided exposing all the issues and incompatibilities between the most popular DBMSs which are of major importance for vendors, users, programmers and researchers of such systems. We believe that this project has contributed in the database management systems research area by introducing a new way of testing the SQL-Compliance of any DBMS and reporting any issues and incompatibilities that may arise. In addition, the implemented tools can be major importance for future vendors or researchers.

Furthermore, demonstrating and analyzing these incompatibilities makes users and programmers and researchers aware for these issues. We verified our assumptions that some parts of the Standard are implemented differently but it was somewhat surprising that so many differences have emerged. Lastly, the implemented framework is portable

and can be extended efficiently. For example, although experimental evidences are provided for both numeric and alphanumeric data types, the random generator tool is implemented in such a way that can track any data types such as Date. In that way, it can be extended efficiently to generate queries with attributes of date as data type.

## 6.2 Summary of the findings

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

Table 6.1: My caption

Operation	Mysql	PostgreSQL	Microsoft SQL Server	Oracle	IBM DB2
INTERSECT ALL	X	✓	X	X	✓
INTERSECT	✓	✓	✓	X	✓
AS in FROM Clause	✓	✓	✓	X	✓
EXCEPT ALL	X	✓	X	X (MINUS ALL is not supported)	✓
EXCEPT	X	✓	✓	X	✓
GROUP BY contains columns not in SELECT list	✓	X	X	X	X
Arithmetic operations in WHERE Clause	✓	X	X	X	✓
Support keyword True in WHERE clause	✓	✓	X	X	✓
Support of % operator	✓	✓	✓	✓ (It uses mod function)	✓
Division by zero	✓	X	X	X	X
Row comparison	✓	✓	X	✓	✓
Identical names in the FROM Clause	X	X	X	X	✓
SQL Query without FROM Clause	X	✓	✓	X	X
operator in SELECT Clause	✓ (But It uses    as logical OR), (Uses CONCAT function )	✓	✓ ( It uses + )	✓	✓
TRIM function in SELECT Clause	✓	✓	X	✓	✓
operator in SELECT Clause	✓ (But it uses as logical OR)	✓	✓	✓	✓
SUBSTRING function in SELECT Clause	✓	✓	✓	X (It uses SUBSTR function instead)	✓
Enclose Strings with instead of	✓	X	X	X	X
LIKE Operator	✓	✓ (Case- sensitive)	✓	✓ (Case- sensitive)	✓

### 6.3 Suggestions for future work

Several issues arose when DBMSs are tested. The experiments are conducted in various databases that contained integers and strings. We expect that more issues can arise by generating also databases containing dates but by doing so, the generator tool should be extended in order to support this new data type. This should be an easy extension as there is the provision for supporting any data type. Yet another future extension could be to include a new DBMS for evaluation of its SQL-compliance. This extension also should not need a lot of effort as the architecture is implemented in such a way that a new system can be easily added.

# 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.