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 proper DBMS software. In the past few decades, the amount of data has increased exponentially, causing a rapid increase in the demand for systems that can store, organise and manipulate such data. With this requirements in mind, DBMSs such as Oracle DB, Microsoft SQL Server, MySQL and PostgreSQL were created, each implementing its own architecture. However, since these DBMSs have been implemented with a significant amount of differences, a Standard was set by the American National Standards Institute (ANSI) in order to provide a common interface for all DBMSs and ensure compatibility for the SQL language adopted in all implementations. Even though, the SQL Standard has been introduced and adopted by most DBMS vendors, there are some differences that still exist, probably due to the difficulty of interpreting and implementing all the parts of the Standard, 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 incompatibilities 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 investigates five different DBMSs 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 Standard 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 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 the SQL Standard 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

List of Figures

List of Tables

Chapter 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 has been widely used by 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. 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.

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. However, 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

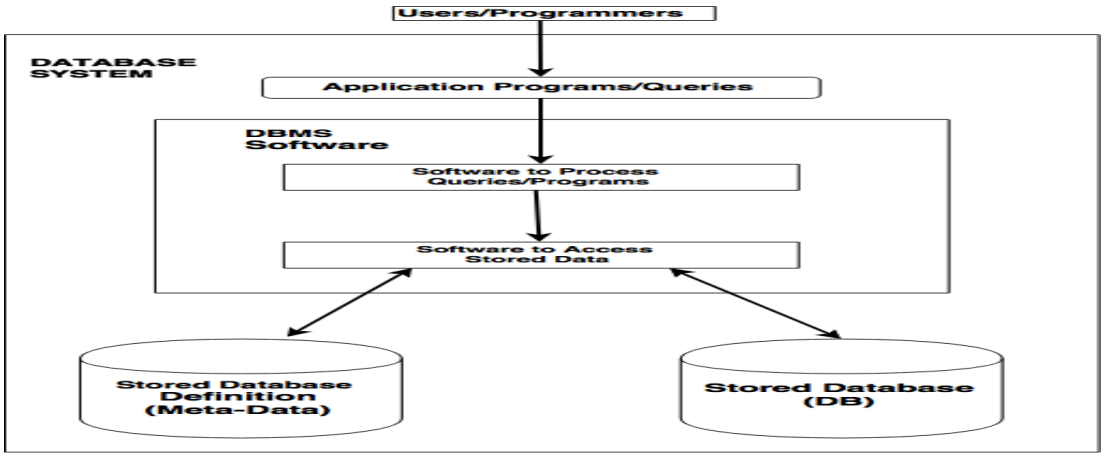

Figure 1.1: Abstract DBMS architecture

been emerged over the years, however, 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.

Such systems have been extensively studied and tested for their correctness and efficiency. Software testing is an essential approach for testing and evaluating the quality of such systems [28]. However, implementing a software that can be used to assess complex systems, it usually requires sophisticated and large-scale implementation. On the contrary, automate the process of testing in the only way to have a systematic and effective way of testing such complex systems. Since DBMSs are the most popular systems for storing and manipulate data, many studies have conducted focusing mainly for evaluating their performance. For example, TPC-H is a popular benchmark for evaluating vendors DBMSs implementations [25]. However, this benchmark is designed for analysing performance and it generates a relatively small number of SQL queries.

1.1 Motivation

The SQL Standard is established a long time ago, and aims to provide a common interface among modern DBMSs [2, 11]. 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 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 in exactly the same way by all vendors, almost impossible. Currently, there is no well-known framework that can be used to test the SQL-Compliance among these systems and be capable to detect any incompatibilities or semantic issues. Hence, this project aims to build a complete framework for systematically discovering and highlighting both existing and new differences that may arise between popular DBMSs. Afterwards, experimental evidences with a comprehensive explanation with respects to the SQL standard are provided. 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, programmers and vendors aware about the current problems.

1.2 Related Work

As DBMSs are necessary for many fields some work has been conducted for evaluating whether different vendors have implemented various points of the Standard in the same way. In fact, some parts of the Standard is interpreted differently and SQL code is not portable among these systems. Albeit some differences are demonstrated [1, 14, 19], there is no well-known tool for evaluating these systems and automatically identify any issues and incompatibilities. As a matter of fact, queries that executed without raising any error but interpreted differently by DBMSs can cause different results but without a tool to check the results is almost impossible to be exposed. Thus, the current studies

show results which have been detected by working with these systems. Having a tool for regularly checking the systems is imperative.

1.3 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 framework and an explanation for each individual tool that the framework 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) [8, 16]. 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) [2, 11, 21, 22]. 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) [17, 12,23]. 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

In this section a high-level description of the basic SQL commands is given, as defined in the Standard. Due to the fact that these commands are used to generate random queries, it is imperative to give a short presentation of each command and its purpose.

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 followed: all the rows of the tables list in the FROM Clause are evaluated. Each row that satisfies 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 be 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, 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. In particular, it lists the number of students in every country where there are more than three students in this 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. However, 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 from where there both Students and professors come.

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 extracts 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 defines in which order the expression

should be evaluated. The sequence of operators is provided as follows, classified according to their priority, starting from those with the highest 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 are enclosed by a parenthesis are evaluated first.

2.3.1 Missing values

This section aims to provide a basic background regarding null values and present the problems that can be arised from the presence of nulls in a database. In the section of experiments, it is illustrated that many problems can be appeared. SQL uses null marker for missing or unknown values in a database and for that reason null is a reserved word [24]. It is worthy mentioning that null should not be confused with zero value or an empty string. However, Oracles database treats the empty string as null[12], and for that reason new issues are arised. 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 nulls in a database is the fundamental source of issues and incompatibilities among current DBMS [26, 27]. In addition, 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, an expression can be evaluated only to 2 values such as 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 evaluated to Unknown
```

```
NULL = NULL is evaluated to Unknown
```

```
NULL <= 3  is evaluated 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 thus 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 = NULL should be evaluate to true. However, 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

A few concrete examples are provided below which demonstrate that indeed some aspects of the Standard is implemented differently by each vendor.

Example:

The following SQL query does not return identical results on both PostgreSQL and Oracle [14].

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].

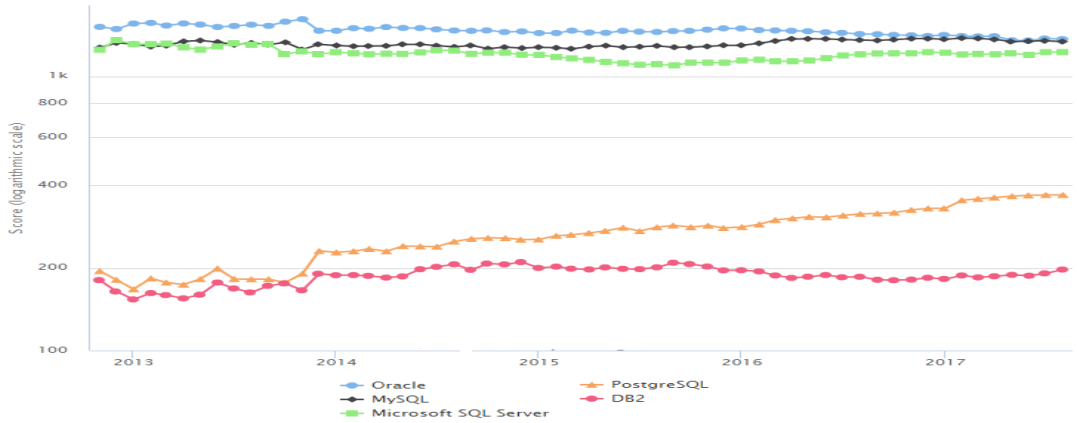

Figure 2.1: Popularity of modern DBMSs

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 small differences might be critical.

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 if freely available and 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

As it was mentioned in the first chapter, the key idea of the current project is to investigate whether current DBMSs comply according to the Standard and highlight the main issues and differences among popular systems. In particular, five popular systems such as MySQL, IBM DB2, Microsoft SQL Server, PostgreSQL and Oracle Database are examined. Hence, this project providing a complete framework that can be used to systematically test the SQL-Compliance as these systems have been evolved rapidly.

The framework consists of three different tools, and more precisely, two different tools have been implemented (1) the Random SQL Query Generator Tool, and (2) the Comparison Tool. The third one, which is used to produce random realistic data is an open-source tool called Datafiller. Different data can be generated using Datafiller tool based on a schema which is provided as a parameter. In addition, complex data can be generated by specifying additional parameters to the tool.

The purpose of the random query generator tool is to rapidly generate a very large number of SQL queries based on the SQL statements as defined in the SQL Standard. In that way, an examination of whether answers to SQL queries are the same independently of which DBMS they are executed on or not can be performed. Therefore, the comparison tool is used to identify if the results are identical among DBMSs. A detailed explanation of the internal implementation and structure of those tools is given in the following Chapter. The implementation of the tool is modular and allows the addition of new DBMSs seamlessly without altering the existing structure of the tool. The comparison of the DBMSs results will be performed using a main-memory data structure for achieving efficiency. Differences among current DBMSs are automati-

cally 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

This chapter describes the framework that is implemented for assessing the SQL-compliance of current DBMSs.

The framework is divided into three different tools as introduced in the previous chapter. The first two tools are implemented in the Java programming language and the third one is implemented in the Python programming language which is an open-source project that can generate random realistic data. This project uses Java programming language since it is very popular and it can be used to build cross-platform systems. The complete framework consists of twenty java classes.

4.1.1 Random SQL generator engine

An important component of the framework is the random SQL query generator tool which is used to generate random SQL queries for assessing modern DBMSs. The SQL language consists of 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 such as MAX, COUNT or AVG. These SQL commands need to be properly generated and syntactically correct in order to avoid syntax or semantic errors.. As a result, for generating meaningful and syntactically valid queries, the current implementation of the

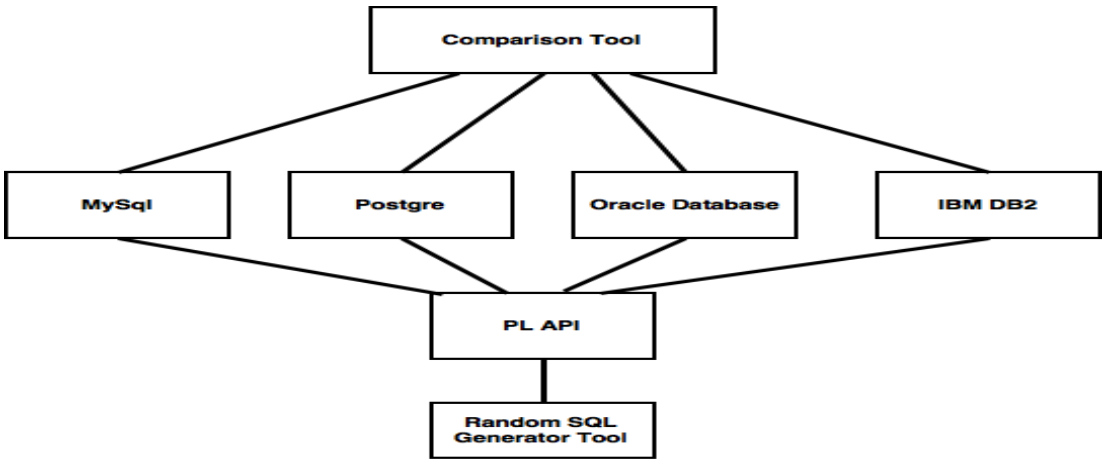

Figure 4.1: Random SQL Generator Architecture

generator tool must take into consideration table and attribute Naming, data type compatibility and projection.

In addition, the tool has been designed in a modular way for reusability and therefore new systems can easily extend it without the need of changing the whole structure of the tool.

Moreover, an important decision that is taken is with regards to the internal implementation of the generator tool, in order to make it feasible to generate more valid and syntactically correct SQL queries. Hence, an internal representation is implemented with each of its classes responsible for generating a different SQL clause that contributes to the overall query. For example, one of the java classes generates the SELECT clause. Having different classes for each SQL statement 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 parts 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 the attribute names and data types need to be tracked first. If SQL strings were to be used, it would not be possible to check if a variable mentioned in the WHERE clause is also mentioned in the FROM clause or if that variable comes as a parameter from an external query. Thus, there is a need to track attributes for each clause and in order to achieve that, LinkedList and HashMap data-structures were used.

4.1.2 Configuration file

The generator tool also includes a configuration file for partially controlling the random SQL queries and a detailed explanation is given as follows: The configuration file is used to provide information to the generator tool. Therefore, many parameters can be specified from the configuration file. Below, 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 SQLEngine
    Maximum number of tables in the FROM STATEMENT
maxTablesFrom =3
    Maximum number of attributes in the SELECT STATEMENT
maxAttrSel =5
    Maximun 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 the provision of relations and attributes to the tool. An initial approach was to use them as parameters in the configuration file. Although, this approach works acceptably, the resulting tool lacks portability, especially when a DBMS has a large number of tables and columns. Hence, It will be time-consuming to give these parameters via the configuration file. Thus, an efficient approach is to retrieve the whole schema from DBMSs automatically. As a result, the implemented 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. The current architecture has the capability to automatically retrieve the whole database schema from any DBMS by just providing the credentials in the configuration file such as user, pass and dbName parameters.

- **MaxTablesFrom** = parameter is used to set an upper bound of the number of tables that a 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 set to the total number of tables.

- **MaxAttrSel** = parameter indicates an upper bound of projections in the SELECT clause. 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 considerations should be taken into account. For example, we should track attributes for outer queries, because 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 comparisons in the SELECT clause.

- **Dinstinct** = parameter represents the probability of the DISTINCT command appearing in the SQL statement.

- **StringInSel** = parameter indicates the probability of projecting an attribute of type String or having String functions in the SELECT clause.

- **StringInWhere** = parameter represents the probability of having String comparison in WHERE clause.

It can be seen from the configuration file that many parameters of the random query generator can be controlled but, on the other hand, it does not imply that the diversity of SQL queries that can be generated is restricted. For example, an upper bound of

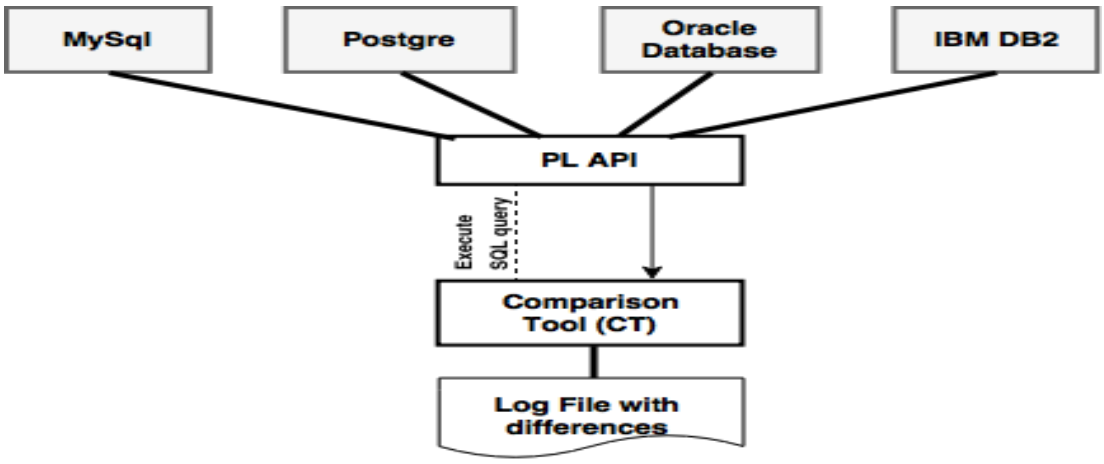

Figure 4.2: Simple Architecture of Comparison Tool

tables that appear in the FROM clause can be set. In that way, having an enormous table size from cartesian product can be avoided.

In addition, it is not so common to have constant comparisons in an SQL query even though SQL Standard supports it. Thus, SQL queries which have constant comparisons are generated but only with a limited number based on the probability which is given in the configuration file.

SQL Query generated by the random query generator

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

4.2 Comparison Tool

The generated SQL queries are evaluated on five different DBMSs as they were mentioned in the methodology chapter. The comparison tool (CT) takes into consideration all methods of accessing different databases so that the process of executing and comparing identical queries among all databases can be automated.

The CT is illustrated below and a detailed explanation of its internal implementation is provided.

The tool is implemented in Java programming language and is one of the main components of the complete framework. Specifically, the CT is used to compare the results of each randomly generated query. As each DBMS uses various algorithms to evaluate each query, sometimes the results are identical but the order or format differs.

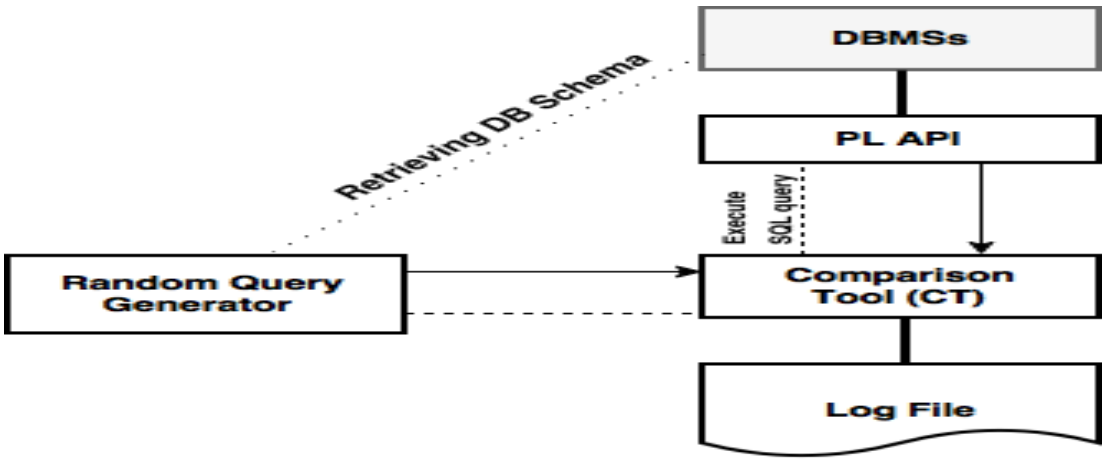

Figure 4.3: Architecture of Comparison Tool

As a consequence, for efficient comparison of the query results between DBMSs, the following approach is used: Initially, the query results are stored in a LinkedList, so that an efficient in-memory sorting algorithm can be performed, called Quicksort. In that way, the results among the DBMSs should be the same and a row by row comparison is performed to verify that they are identical. If a difference is found or a query raises an error, on one of the tested DBMSs, then a detailed explanation of the difference or the error message is recorded in the log file.

An important consideration for implementing this tool is its ease of reuse and extensibility by other systems or software. As each system needs its own credentials such as username, password and database name, a configuration file is created for providing these information to the CT. By doing so, the CT becomes portable and adaptable and a new system can be easily added. As shown in the experiments chapter, our approach can detect many important differences and issues.

4.3 Random generated data

Datafiller is a well-known open-source project that provides the capability of generating random data [3]. For our project, Datafiller will be used for generating a diversity of database instances in order to evaluate all major DBMSs. More precisely, the Datafiller script generates random data, based on a an SQL 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 each field should be unique, foreign key or primary key. Another important parameter is the df: null=rate%

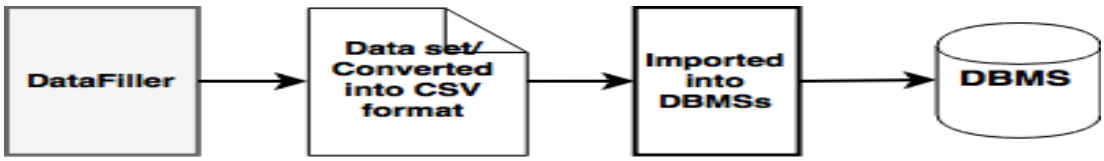

Figure 4.4: Method of importing csv files

which indicates the nullable rates. It necessary to test the behaviour of current DBMS in databases with nulls in order to evaluate whether all DBMSs behave in a similar fashion regarding null values.

Additionally, more complex parameters can be provided as well, such as the number of tuples per table using –size SIZE parameter. It is worth mentioning that these parameters should be defined within the schema script and should start with – df. Furthermore, we can generate more realistic data by providing some information in the 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 next to 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 and therefore, for our project we need to import the random generated data into five DBMSs. For this reason, we transform 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 is followed in order to provide the experimental evidences. Subsequently, the main findings are presented with appropriate explanation according to the Standard.

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 had 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.

A very large number of SQL queries are generated using the random query generator tool. More precisely, more than 100,000 queries are generated. In addition, different schemas are used varying from two to ten relations and two to ten attributes. Since the data types of attributes may be important, the following data types are used: TEXT, CHAR, VARCHAR, INTEGER, SMALLINT, BIGINT. Furthermore, we generated data with different NULL rate such as 5%, 10%, 20% and 40% for identify if differences can arise depending on the number of NULLs that a database contains. Furthermore, for each schema a database instance is generated using the random data generator Datafiller. As an important consideration that needs to be checked with these experiments is whether various features of the Standard are implemented identically, thus the instances size was 100 rows per table. It will be aimless to use larger size of instances since issues and differences of the Standard can be still disclose with a smaller size. In that way, the throughput of generated queries is increased and more queries can be evaluated in less time.

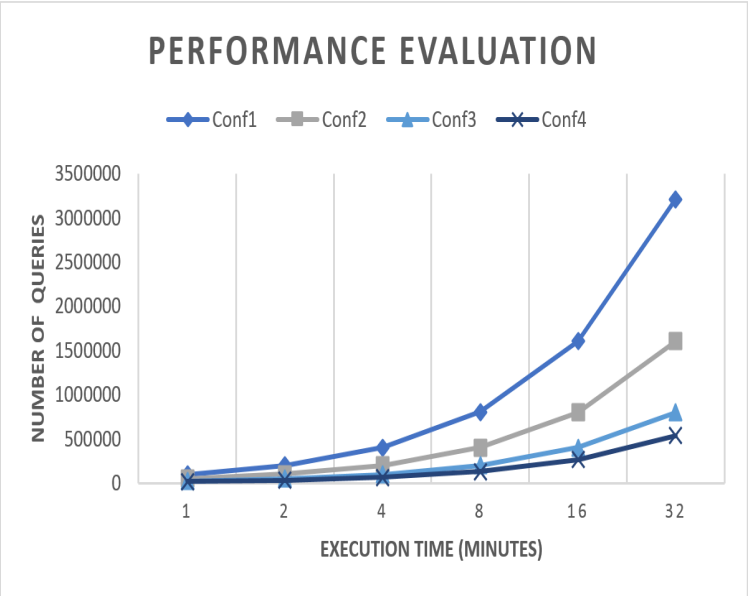

Figure 5.1: Performance Evaluation

5.2 Experiment Results

The implemented framework is used to conduct the experiments. In that way, the current implementation is checked and tested whether is capable to detect differences and issues between the five DBMSs. The results which are illustrated below have the following format: Firstly, the SQL query that causes an error or a semantic issue is presented (As may some features of the Standard are not implemented at all, or they are implemented differently). Then, a table for each query is provided which demonstrates if any DBMS raises an error, otherwise, the keyword Executed meaning that the SQL query is executed. Afterwards, a comprehensive explanation is given for each query by explaining about the source of the problem and an explanation according to the SQL standard is provided.

Difference 1:

Q1:

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

According to the Standard each expression in the WHERE clause is evaluated to

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

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.

Difference 2:
Q2:

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

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.

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

Difference 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, PostgresSQL raises an error when executes this query, in the contrary with the rest DBMSs which are executed the query without any error.

Difference 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.

Difference 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

Difference 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.

Difference 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.

Difference 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.

Difference 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.

Difference 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.

Difference 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.

Difference 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.

Difference 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, PostgresSQL raises an error when executes this query, in the contrary with the rest DBMSs which are executed the query without any error.

Difference 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.

Difference 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

Difference 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.

Difference 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.

Difference 18:

Q18:

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

Table 5.18: Difference 18

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: $substr(column_name, start_pos, no_of_characters)$.

Difference 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 .

Difference 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.

Difference 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: Summarize results

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

Table 6.2: Summarize results

Operation	**Mysql**	**PostgreSQL**	**Microsoft SQL Server**	**Oracle**	**IBM DB2**
|| 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 in implemented in such a way that a new system can be easily added.

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.