A Comparison of MySQL and SQLite on Query Execution

Hallvard Westman (2873575), Daniel Wang (2808686), Griffith University

***Abstract*—Database management systems (DBMS) are essential for users to efficiently store, manage, and use data. In many cases, the variety of DBMS available on the market and the numerous features they offer may be challenging for users to decide on the most appropriate one. In this report, we will examine and compare the two leading open source implementations of relational database management systems, MySQL and SQLite. Features and usability of the two systems will be compared, and a final recommendation will be provided in the conclusion of this report.**

I. Introduction

IN the current environment, database management systems (DBMS) are complex and play an integral role in managing and manipulating data. There are many types of databases that are used across all fields of technology from and can contain varying volumes and types of data. According to Bai [1], there are many DBMS programs providing different features, but the primary goal of DBMS are the same in that they all ow users to create, manage, and query databases. Due to the abundance of database management systems available and the various features they offer, the selection of the most appropriate DBMS can be influential on the effectiveness of the database; Lara et al [2].

The leading open source implementations of relational database management systems available are MySQL and SQLite. Both of these programs provide the user with fundamental DBMS functions but differ in available amount of existing features and usability. MySQL provides standalone traditional client server architecture whereas SQLite is an embedded database without the need of configuration. Because MySQL is based on the client server architecture, it is able to provide more query functionality and support, as well as being more suited for larger databases. SQLite is designed to be a lightweight DBMS that is optimised to be embedded into other programs, which has led it to be adopted in many popular web browsers, mobile operating systems, and various web application frameworks.

In order to determine the usability of MySQL and SQLite, a benchmark will be done on the query evaluation for each database, as well as a comparison of the features and functionality of each. Through the analysis of MySQL and SQLite, its features and usability, we will be able to investigate the pros and cons of each database, and the situations in which one would be more efficient than the other.

II. Background

MySQL is a relational database management system (RDBMS) based on the client server architecture. According to DB-Engines Ranking [3], MySQL is the most widely used open-source database product on the market. Although owned by Oracle, it is still licensed as open source and backed by a community. Due to the architecture and design of MySQL, it is more suited for larger databases as it provides full support for queries and functionality such as server-side scripts and user permissions. MySQL is similar to other commercial DBMS products such as Oracle in that it runs a server and allows multiple users to access numerous databases, and it can also be run on cloud computing platforms.

The limitations of MySQL are similar to other large DBMS products in that its efficiency is limited by the hard disk performance; Vegh [4]. It also does not offer an OLAP layer; therefore, users must install additional OLAP software such as Mondrian in order to use OLAP functions. MySQL also only provides limited support for XML and XPath functionality, as further functionality is still under development; MySQL 5.6 Reference Manual [5].

The primary goal of SQLite is to be a simple, easy to use and administer SQL database that can be embedded into larger programs. SQLite is an embedded SQL database engine (RDBMS) and differs from other RDBMS products in that it does not have a server process. Databases are created as a single file, and queries are made directly by the application's engine. As stated on their website, SQLite is not a replacement for Oracle, but rather a replacement for the function call fopen(); SQLite [6]. The SQLite project is open source, and developed by a dedicated team that is funded by providing professional support of the SQLite product. The nature of SQLite allows it to be embedded into other programs, as such it has been implemented in a number of commercial products from web browsers (Mozilla Firefox, Google Chrome) to applications (Skype, Adobe), and mobile operating systems (Apple iOS, Android).

Due to the design of SQLite, it lacks many of the deeper functionality as other larger relational database management systems. SQLite does not provide full query functionality, such as writing to views, full trigger support, or is suitable for stored procedures. Because it does not have client-server architecture, some features such as replication, user

permissions, and server-side scripts are not applicable in SQLite. In this paper however we will focus on scenarios where both systems have the required features, and attempt to decide which one is the best for the task.

The following table is a comparison between some of the features available in MySQL and SQLite.

TABLE I
FEATURES OF MYSQL [7] AND SQLITE [8], [9], [10]

Feature	**MySQL**	**SQLite**
Server-side scripts	Allows server-side scripts	Does not allow server-side scripts
User permissions (GRANT/REVOKE)	Able	Unable, SQLite uses file system permissions
Partitioning	Horizontal partitioning in MySQL cluster	Cannot partition
Replication	Master-Master, Master-Slave, and MySQL Cluster	None
Server operating systems	Windows, Solaris, Linux, OS X, FreeBSD	Server-less
Concurrency	via table locks or row locks depending on storage engine (best suited for multi-user environment)	via file-system locks (best suited for non-multi-user environment)
Stored procedures & Joins	Fully compatible, along with other common RDBMS operations	Not suited for stored procedures and certain types of joins (RIGHT OUTER JOIN, FULL OUTER JOIN)
ALTER TABLE support	Full support	Only RENAME TABLE and ADD COLUMN (does not support DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT, etc.)
Trigger support	Full support	Only FOR EACH ROW but not FOR EACH STATEMENT
Writing to Views	Full implementation	May not execute DELETE, INSERT or UPDATE on views. Views are read-only
Embedding in hardware	Not suitable, as server component of database is still needed	Fully suitable

III. RELATED WORK

In order to compare two database systems, Boicea et al. [11] used a criteria consisting of query and insertion times, restrictions, and features of MongoDB and Oracle to present the differences between the two databases. The criteria used to compare the products are able to provide the user with an analysis into the features and limitations of each database which is helpful for the user to decide what they want. Ramanathan et al. [12] examined and compared cloud database products used by Amazon and Google, and applied a similar criteria in which features and characteristics of the database systems were examined. Similar comparisons of database products were also completed by Tudorica and Bucur [13] in which several NoSQL database systems were compared using a criterion to analyse the differences in features of NoSQL products. In each of the comparisons by Boicea et al. [11], Ramanathan et al. [12], and Tudorica and Bucur [13], all of the database system used were judged on a criteria based on the features and usability of the products to ultimately allow the user to better understand each database system and choose the most appropriate DBMS for their needs.

A series of performance tests of MySQL and SQLite completed by the SQLite vendor [14] provides a comparison into the speed and efficiency of both database systems. The evaluation of both database systems was performed on older implementations of each system, and thus the results are no longer relevant to the current versions available today.
MySQL and SQLite were each measured on the speed of query evaluation. Queries were executed on thousands of data entries and the amount of time elapsed determined the speed of processing. In the results, SQLite was shown to be faster and more efficient than MySQL for most common operations.

The limitations of the tests were that they were performed on older hardware and older versions of MySQL and SQLite. Since then, both MySQL and SQLite have been updated, new features and existing features have been upgraded; therefore, the previous tests are no longer an appropriate benchmark for current implementations.

IV. METHODOLOGY

In order to compare the differences between MySQL and SQLite, a series of tests has been reconstructed from a previous experiment that was executed by the SQLite vendor. The tests that are conducted will measure the speed and efficiency of each database system in the way they handle SQL queries.

A. Reconstruction

There are a total of 16 tests, each designed by the SQLite vendor to represent general queries that would be typical for both of the DBMS. These tests have not been published, so they have been reconstructed in a PHP script for this paper to generate SQL-syntax for the test execution. This includes statements such as insert, delete, select, update, create, and drop and aggregation. All of these statements are tested in different contexts such as in a single transaction, multiple transactions, on an index, after a delete and so on. All of the different test cases and contexts will be accounted for and explained in the Analysis section.

B. Test Environment

In this evaluation, the tests will be executed on fresh installations of the newest releases of the respective database systems, on current hardware specifications and a fresh Operating System (OS) installation.

The hardware to be used will be running a Virtual Machine (VM) given a single core of a 2.4GHz Intel processor and 2048MB of 1067MHz DDR RAM. HDD is a Corsair force 3 solid state disk, and the host OS for the VM is OSX Lion, on a Macbook 5.1 mid-2009. The VM operating system is Ubuntu 12.04.03 LTS Server. The specification of the host operating system is not considered to be relevant to the test outcome, because the tests will be compared on the same system. The VM OS is however chosen because it is a long term support OS, supported until 2017.

C. Benchmark Utility

In order to test the speed and efficiency of each DBMS, we will look at the elapsed time of execution, as well as CPU usage of each query. The tests will be executed using the time utility [15] that is natively installed on Ubuntu in order to maintain integrity. This way, none of the DBMS will influence how the tests are conducted, and the same tools are used on both DBMS's. The time command will be executed on the BASH command line of the server, with the specific query as parameter. All the queries will be executed sequentially, by the following bash script.

```
for i in {1..16}; do
    echo $(/usr/bin/time -o result/mysql/q$i.txt mysql BENCHMARK < query$i.sql)
done
```

This script iterates through all 16 SQL scripts and passes the SQL to the given DBMS. After the DBMS has executed the SQL, the command returns to the time utility which returns the following variables to a text file:

- System time
  - CPU time in seconds spent in kernel mode
- User time
  - CPU time in seconds spent in user mode
- Total elapsed time
  - The total elapsed time in seconds of the command, including CPU time as well as waiting for IO.
- CPU usage.
  - The amount of CPU used to processing the command in percentage.

These values will be presented in the benchmark, but in order to keep it simple when evaluation the products the Total elapsed time, and CPU usage will be used considering the total elapsed time, includes users and system time.

D. Evaluation

MySQL (5.5) and SQLite (3.8.0.2) will be used in the tests in order to evaluate the performance and usability of each system. MySQL (5.6) has been released, but has not yet been accepted to the LTS repository of Ubuntu, so it cannot be guaranteed support. For that reason we have chosen to implement the versions of MySQL and SQLite which is supported by Ubuntu 12.04.

V. Analysis

The results of the testing have been compiled into the following table.

TABLE II
TEST RESULT OF MYSQL AND SQLITE

Query	MySQL	SQLite
1: 1000 INSERTs	0.00 user 0.03 system 0:00.36 elapsed 12% CPU	0.00 user 0.50 system 0:01.44 elapsed 34% CPU
2: 25000 INSERTs in a transaction	0.28 user 0.53 system 0:07.62 elapsed 10% CPU	0.24 user 0.10 system 0:00.35 elapsed 98% CPU
3: 25000 INSERTs into an indexed table	0.23 user 0.57 system 0:08.32 elapsed 9% CPU	0.44 user 0.13 system 0:00.59 elapsed 98% CPU
4: 100 SELECTs without an index	0.00 user 0.01 system 0:01.58 elapsed 1% CPU	0.66 user 0.02 system 0:00.70 elapsed 98% CPU
5: 100 SELECTs on a string comparison	0.00 user 0.02 system 0:01.89 elapsed 1% CPU	2.10 user 0.03 system 0:02.14 elapsed 99% CPU
6: Creating an index	0.00 user 0.00 system 0:00.25 elapsed 4% CPU	0.10 user 0.06 system 0:00.16 elapsed 95% CPU
7: 5000 SELECTs with an index	0.10 user 0.08 system 0:00.85 elapsed 22% CPU	0.19 user 0.18 system 0:00.45 elapsed 82% CPU
8: 1000 UPDATEs without an index	0.01 user 0.03 system 0:00.89 elapsed 5% CPU	0.22 user 0.02 system 0:00.25 elapsed 97% CPU
9: 25000 UPDATEs with an index	0.27 user 0.53 system 0:04.08 elapsed 19% CPU	0.76 user 0.15 system 0:00.92 elapsed 99% CPU
10: 25000 text UPDATEs with an index	0.28 user 0.46 system 0:03.65 elapsed 20% CPU	0.61 user 0.11 system 0:00.73 elapsed 98% CPU
11: INSERTs from a SELECT	0.00 user 0.00 system 0:01.22 elapsed 0% CPU	0.28 user 0.15 system 0:00.45 elapsed 96% CPU

12: DELETE without an index	0.00 user 0.01 system 0:00.21 elapsed 5% CPU	0.14 user 0.28 system 0:00.45 elapsed 96% CPU
13: DELETE with an index	0.00 user 0.01 system 0:00.52 elapsed 2% CPU	0.16 user 0.14 system 0:00.30 elapsed 98% CPU
14: A big INSERT after a big DELETE	0.00 user 0.01 system 0:00.78 elapsed 1% CPU	0.27 user 0.06 system 0:00.34 elapsed 98% CPU
15: A big DELETE followed by many small INSERTs	0.11 user 0.24 system 0:01.18 elapsed 29% CPU	0.14 user 0.07 system 0:00.22 elapsed 97% CPU
16: DROP TABLE	0.00 user 0.00 system 0:00.09 elapsed 13% CPU	0.00 user 0.08 system 0:00.09 elapsed 91% CPU

As shown in the above results, SQLite is almost always faster than MySQL in processing queries, but requires more CPU resources in the process. In the cases where MySQL was faster such as 1000 inserts in multiple transactions, SQLite would have to close and reopen the database file for each insert, resulting in a slower execution time. This argument is supported by the second test where we execute 25000 inserts in a single transaction, which in SQLite is actually faster than the 1000 inserts of the first test with multiple transactions. In the second and third query, MySQL is drastically slower than SQLite, as much as 20 times slower.

The tests do however suggest that MySQL's execution of inserts is linear regardless of if the inserts are part of a single or of multiple transactions. When performing select statements in query 4 and 5, SQLite performs better on a straight up select without an index, but the results are almost the same when it comes to string comparisons. This indicates that comparisons of strings would need to be executed more or less the same regardless of DBMS, while a straightforward select with (query 4) and without (query 7) is slower on MySQL when it has to be treated by the server, and not just executed as a read such as in the case of SQLite. Creation of indexes (query 6) is marginally in favor of SQLite on execution time, but is significantly better on MySQL in regards to CPU usage. Updates with and without an index(query 8,9,10) is consistent for the two DBMS's, and is in favor for SQLite by 4 times the execution time of MySQL. This is the same result as the tests performed by the SQLite vendor, and it was not concluded as to why that was. In this test, it seems consistent with the results for the select statements in query 4 and 5, which is presumed to be due to the MySQL server architecture. Delete Statements shows different results on the two vendors if they include an index or not. SQLite will perform better deleting with an index, and MySQL performs better without. The margin of execution time is small, and therefore the reason is not transparent. In comparison with other single operations such as creating an index, SQLite would be faster, this is however not the case when an index is not in place. As shown in query 16, the single operation of drop table has the exact same execution time, preventing the conclusion of single operations to be faster on either DBMS. Composite transaction where a delete is followed by multiple inserts and is in SQLites favor, and is consistent with the same actions when performed isolated.

FIGURE I
MYSQL & SQLITE QUERY TIME

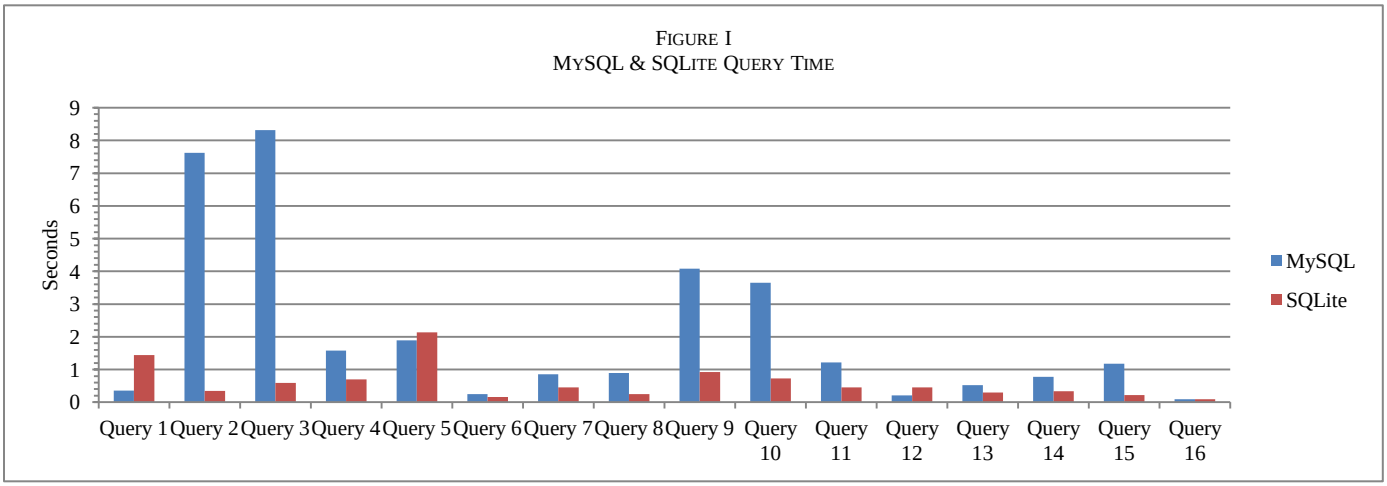
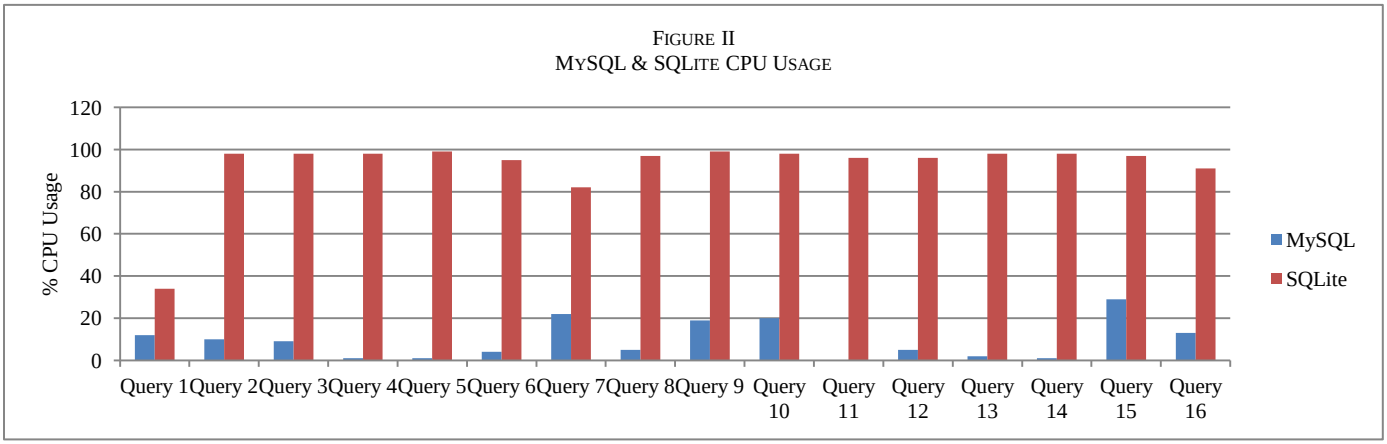


FIGURE II
MYSQL & SQLITE CPU USAGE



VI. Conclusion

The methods used in evaluating MySQL and SQLite successfully identified that MySQL is slower in processing queries, but requires less CPU resources in doing so. In order to process queries, SQLite requires more CPU usage and queries are not optimized for I/O which is not an issue in MYSQL. This means that when writing queries for SQLite, the user will have to account for I/O to get the best performance as it writes directly to the file. With MySQL, the user does not have to account for I/O when writing queries because the MySQL server architecture handles I/O by independently. This process leads to MySQL using less CPU resources than SQLite, but overall takes more time to execute queries. The result of the testing supports the idea that SQLite is not suitable for a client server database [8] whereas MySQL is more appropriate in a client server environment. The architecture of SQLite does not support high volume access and concurrency of multiple users due to its CPU usage, which is one of the major advantages of MySQL because MySQL runs the DBMS as a service.

We have concluded that SQLite is best used in situations where high CPU usage is acceptable, fast access to the database is required, low volume web sites and situations with low number of concurrent writers. MySQL is suggested to be the best option when there are multiple users need to write to the database, high volume environment (data and users), and situations where a client server database is needed.

References

[1] Bai, Y. (2010) Introduction to Databases, in Practical Database Programming with Visual C#.NET, John Wiley & Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/9780470567845.ch2

[2] Lara, J. M. G., Osma, B. G. and Noguer, B. G. d. A. (2006) Effects of database choice on international accounting research. Abacus, 42: 426–454. doi: 10.1111/j.1467-6281.2006.00209.x

[3] DB Engines Ranking (2013) September Complete Rankings [Online]. Available: http://db-engines.com/en/ranking

[4] Vegh, A. (2010) MySQL Database Server, in Web Development with the Mac®, Wiley Publishing, Inc., Indianapolis, IN, USA. doi: 10.1002/9781118255759.ch14

[5] MySQL 5.5 - 12.11. XML Functions [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/xml-functions.html

[6] SQLite - About SQLite [Online]. Available: http://www.sqlite.org/about.html

[7] MySQL 5.5 - MySQL 5.5 Reference Manual [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/index.html

[8] Appropriate Uses for SQLite [Online]. Available: http://sqlite.org/whentouse.html

[9] Implementation Limits for SQLite [Online]. Available: http://sqlite.org/limits.html

[10] Distinctive Features of SQLite [Online]. Available: http://sqlite.org/different.html

[11] Boicea, A., Radulescu, F., Agapin, L. I. (2012) MongoDB vs Oracle - Database Comparison doi: 10.1109/EIDWT.2012.32

[12] Ramanathan, S., Goel, S., Alagumalai, S. (2011) Comparison of Cloud database: Amazon's SimpleDB and Google's BigTable. doi: 10.1109/ReTIS.2011.6146861

[13] Tudorica, B.G., Bucur, C. (2011) A comparison between several NoSQL databases with comments and notes. doi: 10.1109/RoEduNet.2011.5993686

[14] SQLite Database Comparison: SQLite Database Speed Comparison [Online]. Available: http://sqlite.org/speed.html

[15] Time Utility [Online]. Available: http://man7.org/linux/man-pages/man1/time.1.html