





Performance evaluation of relational embedded databases: an empirical study

Evaluación del rendimiento de bases de datos embebida: un estudio empírico

Author:

 Hassan B. Hassan¹
 Qusay I. Sarhan²

SCIENTIFIC RESEARCH

How to cite this paper:

Hassan, B. H., and Sarhan, Q. I., Performance evaluation of relational embedded databases: an empirical study, Kurdistan, Irak. Innovaciencia. 2018; 6(1): 1-9.

<http://dx.doi.org/10.15649/2346075X.468>

Reception date:

Received: 22 September 2018

Accepted: 10 December 2018

Published: 28 December 2018

Keywords:

Embedded devices, Embedded databases, Performance evaluation, Database operational performance, Test methodology.

ABSTRACT

Introduction: With the rapid deployment of embedded databases across a wide range of embedded devices such as mobile devices, Internet of Things (IoT) devices, etc., the amount of data generated by such devices is also growing increasingly. For this reason, the performance is considered as a crucial criterion in the process of selecting the most suitable embedded database management system to be used to store/retrieve data of these devices. Currently, many embedded databases are available to be utilized in this context. **Materials and Methods:** In this paper, four popular open-source relational embedded databases; namely, H2, HSQLDB, Apache Derby, and SQLite have been compared experimentally with each other to evaluate their operational performance in terms of creating database tables, retrieving data, inserting data, updating data, deleting data. **Results and Discussion:** The experimental results of this paper have been illustrated in Table 4. **Conclusions:** The experimental results and analysis showed that HSQLDB outperformed other databases in most evaluation scenarios.

¹ Software Engineering and Embedded Systems (SEES) Research Group, University of Duhok, Duhok, Kurdistan Region, Iraq, Email: hassan.bapeer@uod.ac.

² Software Engineering and Embedded Systems (SEES) Research Group, Department of Computer Science, College of Science, University of Duhok, Duhok, Kurdistan Region, Iraq, Email: qusay.sarhan@uod.ac.

INTRODUCTION

Recent advances in data-driven software applications and systems that need to store their own data within the machines they reside in rather than remote machines/servers, have paved the way for different types of software applications and devices to use embedded databases. An embedded database is a small size database integrated with a software application that requires access to its own data. Thus, the database system is “hidden” from the application’s end-user and requires little or no maintenance activities (1). Embedded database systems can be used for various purposes. For example, they can be used for storing system logging activities, email archive activities, contacts information in mobile devices, and sensor data in constrained and IoT devices (17). To ensure database consistency, the data are stored and updated in short running transactions which increases write latency and amplification (2). This plays a determining factor in the performance of applications. As the data in embedded databases reside in the same machine that runs the application; the database engine is always running and accessible. Besides, it is not shared or networked (only one software application could access the embedded database file at any time). Compared to traditional databases (also called client-server databases) which are often used in web applications, web services, and distributed software applications (16), traditional databases are more complicated. For example, they require monitoring and managing of data sharing mechanisms, servers configuration, servers failure solution, and servers maintenance) (3). In this paper, four well-known open-source relational embedded databases; namely, H2 (4), HSQLDB (5), Apache Derby (6), and SQLite (7) have been compared experimentally with each other to evaluate their operational performance. Database developers and researchers are concerned about the factors affecting the database performance in real-world usage (8). In addition, database benchmarking helps organisations in selecting the most suitable

and compatible database for their businesses. Thus, many studies in literature such as (9-14) have introduced, explored, and evaluated the performance of various traditional and embedded databases alongside many directions. All the aforementioned studies were crucial in providing useful explanation of different types of databases, databases architectures, and database based applications. Also, they were extremely valuable in providing a theoretical background and a general evaluation metrics for this experimental study. However, no previous published study in literature has compared experimentally between the selected embedded databases in terms of data storing and processing capabilities. Thus, this was the rationale behind this experimental study to be conducted.

The rest of this paper is organized in sections as follows. Section 2 presents the evaluation methodology (evaluation conditions, evaluation scenarios, evaluation metrics, and software/hardware setups) used in this experimental study. Section 3 presents the experimental results of comparing the selected relational embedded databases with each other according to various test scenarios. Finally, some conclusions and future works have been given in Section 4.

EVALUATION METHODOLOGY

The evaluation methodology used in this study is adapted from (18) and it includes the evaluation conditions, evaluation scenarios, evaluation metrics, and software/hardware setups that have been applied and used to compare experimentally the performance of each embedded database in terms of data storing and processing capabilities.

1. Evaluation Conditions

The following evaluation conditions have been considered in this study:

- All selected embedded databases have been tested with their default settings and configurations.

- Every test scenario has been applied on each embedded database using Structured Query Language (SQL) statements⁽⁴⁵⁾ with the same test scenario and its related parameters.
- Each database table used in the evaluation process consists of 1000 records (rows) and 10 fields

(columns). The table with its specifications has been created using the SQL statement in Scenario 1.

- All test data in this paper are using identical data structure as shown in Table 1.

Table 1. Database structure

Attribute Name	Data type
efname	varchar(10)
elname	varchar(10)
email	varchar(40)
job	varchar(9)
mgr	int
hiredate	date
sal	numeric(7,2)
comm	numeric(7,2)
gender	varchar(10)
empno	int

- The hardware configuration of the test environment is not the most paramount in this paper, we tend to design some scenarios in certain environment to evaluate the performance of different embedded databases. From our experience in the field, all test scenarios have been selected as the most used scenarios in different types of software applications.
- Test applications (to store/receive data to/from embedded databases) have been programmed and executed on the same computer system to ensure using the same software and hardware specifications.
- Before starting the testing and measurements, all user applications (excepting applications used to test the databases) have been closed.
- During the whole period of testing and measurements, the used computer system disconnected from the Internet.
- Every evaluation scenario has been repeated 10 times and then measurements have been averaged to ensure more accuracy.
- Evaluation results have been recorded after initializing the database driver and establishing con-

nections to the selected embedded databases

2. Evaluation Scenarios

The operational performance of each relational embedded database is compared experimentally via different Evaluation tests taken from real-world database scenarios, as follows:

• Scenario 1

The test application creates a single database table using the following SQL statement:

```
CREATE TABLE empTable "+i+" (empno
INT PRIMARY KEY , efname VARCHAR(10) ,
elname VARCHAR(10) , email VARCHAR(40) ,
job VARCHAR(9) , mgr INT , hiredate DATE,
sal NUMERIC(7,2) , comm NUMERIC(7,2) , gender
VARCHAR(10))
```

• Scenario 2

The test application creates 50 database tables using the following SQL statement in a loop (it-

eration variable *i* goes from 1 to 1000):

```
CREATE TABLE empTable "+i+" (empno  
INT PRIMARY KEY , efname VARCHAR(40),  
elname VARCHAR(40), email VARCHAR(40) , job  
VARCHAR(9) , mgr INT , hiredate DATE , sal  
NUMERIC(7,2), comm NUMERIC(7,2), gender  
VARCHAR(40));
```

- **Scenario 3**

The test application deletes the database table with no inserted records using the following SQL statement:

```
DROP TABLE empTable;
```

- **Scenario 4**

The test application deletes the database table with 1000 records using the following SQL statement:

```
DROP TABLE empTable;
```

- **Scenario 5**

The test application reads all data in the database table using the following SQL statement:

```
SELECT * FROM empTable;
```

- **Scenario 6**

The test application reads only the first record from the database table using the following SQL statement:

```
SELECT * FROM empTable where empno=1;
```

- **Scenario 7**

The test application reads only the middle record from the database table using the following SQL statement:

```
SELECT * FROM empTable where emp-  
no=500;
```

- **Scenario 8**

The test application reads only the last record from the database table using the following SQL statement:

```
SELECT * FROM empTable where emp-  
no=1000;
```

- **Scenario 9**

The test application deletes all data in the database table using the following SQL statement:

```
DELETE * FROM empTable;
```

- **Scenario 10**

The test application deletes only the first record from the database table using the following SQL statement:

```
DELETE * FROM empTable where empno=1;
```

- **Scenario 11**

The test application deletes only the middle record from the database table using the following SQL statement:

```
DELETE * FROM empTable where emp-  
no=500;
```

- **Scenario 12**

The test application deletes only the last record from the database table using the following SQL statement:

```
DELETE * FROM empTable where emp-  
no=1000;
```

- **Scenario 13**

The test application updates all data in the database table using the following SQL statement:

```
UPDATE empTable SET efname = 'Frank' , el-  
name = 'Lowi' , email = 'frank.lowi@gmail.com'  
, job = 'DOCTOR' , mgr = 11, hiredate = '1998-
```

08-09' , sal = 88000 , comm = 900 , gender = 'female';

- **Scenario 14**

The test application updates only the first record in the database table using the following SQL statement:

```
UPDATE empTable SET efname = 'Frank' , elname = 'Lowi' , email = 'frank.lowi@gmail.com' , job = 'DOCTOR' , mgr = 11, hiredate = '1998-08-09' , sal = 88000 , comm = 900 , gender = 'female' where empno=1;
```

- **Scenario 15**

The test application updates only the middle record in the database table using the following SQL statement:

```
UPDATE empTable SET efname = 'Frank' , elname = 'Lowi' , email = 'frank.lowi@gmail.com' , job = 'DOCTOR' , mgr = 11, hiredate = '1998-08-09' , sal = 88000 , comm = 900 , gender = 'female' where empno=500;
```

- **Scenario 16**

The test application updates only the last record in the database table using the following SQL statement:

```
UPDATE empTable SET efname = 'Frank' , elname = 'Lowi' , email = 'frank.lowi@gmail.com' , job = 'DOCTOR' , mgr = 11, hiredate = '1998-08-09' , sal = 88000 , comm = 900 , gender = 'female' where empno=1000;
```

- **Scenario 17**

The test application inserts a single record into the beginning of the database table using the following SQL statement:

```
INSERT into empTable values ( 1 , 'GRANT' ,
```

```
'John' , 'grant.john@gmail.com' , 'ENGINEER' , 10 , '1987-01-01' , 72000 , 200 , 'male' );
```

- **Scenario 18**

The test application inserts 1000 records into the database table using the following SQL statement in a loop (iteration variable i goes from 1 to 1000):

```
INSERT into empTable values ( ++i , 'GRANT' , 'John' , 'grant.john@gmail.com' , 'ENGINEER' , 10 , '1987-01-01' , 72000 , 200 , 'male' );
```

- **Scenario 19**

The test application sums up all the (empno) in the database table using the following SQL statement:

```
SELECT SUM (empno) FROM empTable;
```

- **Scenario 20**

The test application calculates the average of the (empno) in the database table using the following SQL statement:

```
SELECT AVG (empno) FROM empTable;
```

- **Scenario 21**

The test application determines the maximum (empno) in the database table using the following SQL statement:

```
SELECT MAX (comm) FROM empTable;
```

- **Scenario 22**

The test application determines the minimum (empno) in the database table using the following SQL statement:

```
SELECT MIN (comm) FROM empTable;
```

- **Scenario 23**

The test application sorts all the 1000 records

in an descending order by the (empno) column, using the following SQL statement:
 SELECT * from empTable ORDER BY empno desc;

It is important to mention that all the test scenarios listed above have been chosen to cover different aspects of each database’s performance.

3. Evaluation Metrics

The time (in milliseconds) required to perform each

test scenario by each relational embedded database has been used as a metric to evaluate practically the operational performance of each database. Thus, any database performs a specific test scenario in a less time is considered as the best one in performance in that specific test scenario.

4. Software/hardware Setup

This study has been setup with software and hardware which their specifications are presented respectively in Table 2 and 3.

Table 2. Software specifications

	Software	Version
Java Test Applications	Java JDK	1.8.0_191
	NetBeans IDE	8.2
Embedded Databases	H2	1.4.197
	HSQLDB	2.4.1
	Apache Derby	10.10.2.0
	SQLite	3.16.1
Database Driver	JDBC	4.0
Operating System	Microsoft Windows	7 Home Premium (64-bit)

Table 3. Hardware specifications

	Hardware	Detail
Computer System	Laptop Model	HP
	CPU Type	Intel Core i3-2350M
	CPU Speed	2.3 GHz
	CPU Cores	NA
	RAM	4 GB
	Rating (Windows Experience Index)	4.5

EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, the obtained results of the experimental performance analysis of the embedded databases have been presented in Table 4.

Table 4. Databases evaluation results

Scenarios	H2 (ms)	HSQLDB (ms)	Apache Derby (ms)	SQLite (ms)
1	10.207	3.499	<i>638.156</i>	559.157
2	92.284	47.722	<i>25095.642</i>	18597.952
3	3.540	0.801	273.981	<i>449.902</i>
4	42.479	3.202	306.288	<i>363.981</i>
5	98.261	13.700	6.758	0.397
6	22.877	6.591	48.806	0.509
7	22.743	6.557	49.839	0.475
8	21.355	5.727	48.563	0.496
9	71.696	28.206	108.176	<i>363.609</i>
10	11.034	2.850	53.718	<i>376.589</i>
11	11.396	3.189	51.339	<i>348.899</i>
12	10.763	2.985	51.395	<i>377.192</i>
13	459.712	280.563	410.435	<i>639.056</i>
14	29.324	176.083	281.881	<i>521.942</i>
15	31.861	172.552	273.856	<i>547.213</i>
16	31.643	168.290	278.517	<i>560.448</i>
17	22.266	3.034	116.303	<i>384.963</i>
18	346.852	538.823	3840.694	<i>353484.962</i>
19	0.00679	0.00776	<i>0.00942</i>	0.00535
20	0.00566	0.0151	<i>0.0153</i>	0.0052
21	0.0059	0.0067	<i>0.0111</i>	0.0053
22	0.00661	0.00835	<i>0.00907</i>	0.0045
23	0.00718	0.0068	<i>0.022898</i>	0.00568

From Table 4 (plain bold font represents the best performance and italic bold font represents the worst performance), the following facts have been observed:

- In general, HSQLDB and H2 outperformed the other databases in most test scenarios. However, HSQLDB outperformed H2 as it never had a worst performance.
- In general, Apache Derby and SQLite were the worst ones in performance in most test scenarios compared to other databases.
- All databases excluding Apache Derby have the best performance in some scenarios.
- All databases excluding HSQLDB have the worst performance in some scenarios.
- Making a trade-off between best performance and worst performance results, HSQLDB can be considered as the most recommended embedded database with an acceptable level of performance.

CONCLUSIONS

This paper presented an experimental approach to evaluate the operational performance of four well-known open-source relational embedded databases; namely, H2, HSQLDB, Apache Derby, and SQLite in terms of creating database files, retrieving, inserting,

updating and deleting data. A well-defined test methodology has been proposed and used to achieve the goal mentioned before. Overall performance evaluation and analysis showed that HSQLDB database outperformed the others in most evaluation scenarios. In Table 3, the summary of this comparison study has been presented. This study is crucial to help software developers to select a relational embedded database with an acceptable level of data storing and processing capabilities. For the future, some works could be: (a) applying the evaluation approach used in this paper to evaluate other open-source embedded databases rather than the ones used in this study. (b) using more than 1000 records and more than 10 fields to do further performance evaluation of each of the selected embedded databases. (c) measuring the impact of using complex data types (e.g. images), joined tables (also called parent/child tables), nested queries (e.g. integrating INSERT and SELECT queries into a single nested query), etc. on the overall performance of each of the selected embedded database.

REFERENCES

1. Mingyao X, Xiongfei L. Embedded database query optimization algorithm based on particle swarm optimization. Proceedings of the 7th International Conference on Measuring Technology and Mechatronics Automation; 2015 June 13-14; Nanchang, China; IEEE; 2015. p. 429-432. <https://doi.org/10.1109/ICMTMA.2015.109>
2. Oh G, Kim S, Lee SW, Moon B. SQLite optimization with phase change memory for mobile applications. Proc. VLDB Endow. 2015; 8(12): 1454-65. <https://doi.org/10.14778/2824032.2824044>
3. Kang W, Son SH, Stankovic JA. Design, implementation, and evaluation of a QoS-aware real-time embedded database. IEEE Transactions on Computers. 2012; 61(1): 45-59. <https://doi.org/10.1109/TC.2010.240>
4. H2 Database Engine (redirect) [Internet]. H2database.com. [cited 1 October 2018]. Available from: <https://www.h2database.com>
5. HSQLDB [Internet]. Hsqlldb.org. [cited 1 October 2018]. Available from: <http://hsqldb.org/>
6. Apache Derby [Internet]. Db.apache.org. [cited 1 October 2018]. Available from: <https://db.apache.org/derby/>
7. SQLite Home Page [Internet]. Sqlite.org. [cited 1 October 2018]. Available from: <https://sqlite.org/index.html>
8. Ray S, Simion B, Brown AD. Jackpine: A benchmark to evaluate spatial database performance. Proceedings of the 27th International Conference on Data Engineering; 2011 April 11-16; Hannover, Germany; IEEE; 2011. p. 1139-1150. <https://doi.org/10.1109/ICDE.2011.5767929>
9. Kabakus AT, Kara R. A performance evaluation of in-memory databases. Journal of King Saud University-Computer and Information Sciences. 2017; 29(4): 520-5. <https://doi.org/10.1016/j.jksuci.2016.06.007>
10. Song W, Tao T, Gao T. Performance optimization for flash memory database in mobile embedded system. Proceedings of the 2nd International Workshop on Education Technology and Computer Science; 2010 March 6-7; Wuhan, China; IEEE; 2010. p. 35-39. <https://doi.org/10.1109/ETCS.2010.109>
11. Olson MA. Selecting and implementing an embedded database system. Computer. 2000; 33(9): 27-34. <https://doi.org/10.1109/2.868694>
12. Li Y, Manoharan S. A performance comparison of SQL and NoSQL databases. Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM); 2013 August 27-29; Victoria, BC, Canada; IEEE; 2013. p. 15-19. <https://doi.org/10.1109/PACRIM.2013.6625441>
13. Patchigolla VN, Springer J, Lutes K. Embedded database management performance. Proceedings of the 8th International Conference on Information Technology: New Generations; 2011 April 11-13; Las Vegas, NV, USA; IEEE; 2011. p. 998-1001. <https://doi.org/10.1109/ITNG.2011.171>
14. Tonghui Q, Yang Q, Limin C, Zili S, Xiaowu C, Dehua L. The design of embedded database management system for mobile computing. Proceedings of the International Conference on Computer Science and Information Processing (CSIP); 2012 August 24-26; Xi'an, Shaanxi, China; IEEE; 2012. p. 1454-1457.

15. Batra R. A Primer on SQL. 3rd ed. Leanpub; 2015.
16. Sarhan QI, Gawdan IS. Web Applications and Web Services: A Comparative Study. Science Journal of University of Zakho. 2018; 6(1): 35-41. <https://doi.org/10.25271/2018.6.1.375>
17. Sarhan QI. Internet of things: a survey of challenges and issues. International Journal of Inter-
net of Things and Cyber-Assurance. 2018; 1(1): 40-75. <https://doi.org/10.1504/IJITCA.2018.090162>
18. Sarhan QI, Gawdan IS. Java Message Service Based Performance Comparison of Apache ActiveMQ and Apache Apollo Brokers. Science Journal of University of Zakho. 2017; 5(4): 307-12. <https://doi.org/10.25271/2017.5.4.376>