STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Number 2, 2014

TRANSPARENT INTERCONNECTING PROXY FOR RDBMSS

RADU DRAGOŞ, DIANA HALIŢĂ, AND ANDREEA PUŞCAŞ

Abstract. The constraints of using a particular RDBMS change over time and a developer might be facing the difficult decision to migrate to another RDBMS. Usually, this is a difficult and time consuming task, during which databases must be converted to the new format and the client software must also be changed to connect to the new database server. If the source code for the client application is not available, this task cannot be performed.

This paper advances an original idea of implementing a proxy system that allows a database client software (MySQL) to transparently connect to a different database server (SQLite), without any modification in the original application.

1. INTRODUCTION

With the development of the Internet and cloud computing, it is really necessary to be able to store and process big data effectively. The idea for efficient management of high-performance tasks leads to new challenges for traditional relational database.

This paper presents a study which highlights a comparison and a correlation between two of the most popular relational database management systems (RDBMSs): MySQL and SQLite.

1.1. **Motivation.** Given the two RDBMSs it has been raised the issue of interoperability of the two systems, and even the possibility of migrating from one system to another. Therefore, we have considered the need of implementing an application, a proxy system that interconnects the above mentioned technologies. This system will act as a transparent bridge for the application

Received by the editors: September 5, 2014.

2010 *Mathematics Subject Classification.* 68P15.

1998 *CR Categories and Descriptors.* D.2.12 [**Software Engineering**]: Interoperability – *Data mapping*; H.2.3 [**Database Management**]: Languages – *Query languages*; H.2.4 [**Database Management**]: Systems – *Query processing*.

Key words and phrases. sql proxy system, SQLite, MySQL, migrating between RDBMSs.

which tries to connect to a MySQL server, but the requests are instead captured by the proxy, performed on SQLite databases and served back as MySql responses.

1.2. **Objectives.** Such a proxy could be used in unexpected situations, where a certain application does not have any more access to a MySQL server. This problems can be now solved, by replacing MySQL with SQLite, which does not require installation and is easier to maintain.

The purpose of the paper is to present the implementation of a portable PHP application (a client-server proxy) that can take requests from MySQL clients, but operations performed on SQLite database. This application can be used either from command line interpreter or it can be developed as an web-based application as well.

2. BACKGROUND AND RELATED WORK

Data collections are usually organized by using different RDBMSs, like Oracle, MySQL, Microsoft SQL Server or PostgreSQL. The main reason of using SQL is to access the data stored in a RDBMS. However, each RDBMS is unique in its own way because of its data types and querying methods. There are a lot of differences between them, and these differences are making migration amongst different RDBMS very difficult.

MySQL [4] is a relational database management system and it is a key part of LAMP (Linux, Apache, MySQL, PHP) open source enterprise software. MySQL is well appreciated, being the best and the most used database in the world for online applications. It is also available and is affordable for everyone, it is easy to use, it is continuously improved while remaining fast, safe and reliable and more important it does not have bugs.

2.1. **SQLite.** SQLite [1, 2] is an open source software library that implements an SQL database engine by itself, without a server, zero configuration and transactional [8]. SQLite is different from most other SQL database engines as it was designed to be simple to administer, to use, to be embedded in a larger program, to maintain and to set up. SQLite is often characterized as small and fast, safe and simple.

There are presented below a few of the main features of SQLite which are different from many others SQL database engines:

- **zero configuration**: SQLite does not require installation prior its use;
- **serverless**: with SQLite, the process that wants to access the database reads directly from files stored on the disc;

- **one file database**: a SQLite database is a single file on disk that can be placed anywhere in the hierarchy of directories. If SQLite can read the file on disk then can read any database. If the file on the disk and its directory are writable, then SQLite may change any database. Database files can be easily copied to USB memory sticks or sent by e-mail for sharing;
- **compact**: when optimized for size, the whole SQLite library with all options enabled is less than 225 KB;
- **dynamic data type allocation**: SQLite uses the concepts of storage classes and column affinity type which allows for storing different data types in the same column. In this way, the data type property is the value itself, not the column where the value is stored;
- **variable amount of space for table records**: SQLite uses only the amount of disk space required to store the information in a line; this makes databases run faster, since there is less information to be moved to and from disk;

2.1.1. *Why choose SQLite?*

- SQLite is the ideal solution as a database for web sites which have a small to medium traffic, situation encountered in almost 99% of cases;
- as SQLite databases requires almost no management, SQLite is a good choice for devices or services that must run unmanaged or without human support. SQLite is suitable for use in mobile phones, PDAs, hand-held devices;
- SQLite has been used successfully as a file on disk or desktop applications such as financial analysis tool, programs to keep records and so on.
- from pedagogical perspectives, SQLite is a database engine suitable for use in teaching SQL;

2.1.2. *Why choose other RDBMSs?*

- when we are dealing with client-server applications we should take into consideration a client-server database engine instead of SQLite;
- when we are holding very large data sets;
- when concurrency is needed;

2.2. **Migrating to a new RDBMS.** When migrating to a new RDBMS, an application may require full and direct access to the database. So, the migrating process will imply several modifications in order to communicate with the new RDBMS. This will lead to a large volume of work, especially if the application is using a lot of features specific to the old RDBMS that are not available in the new one. [6]

The solutions which were proposed untill now were implying building a web service which can access the data set. The data access service exposes the atomic data from the database in an independent system and auto descriptive format, by using XML and offers methods for data filtering and sorting [6]. In [5] there are proposed three different approaches related to database conversions: handling data using XML/OO interfaces, connect existing RDBMS to a conceptually different database system or migrating an RDBMS into a target database. They also emphasize the need of two distinct steps, such as: database integration and database migration.

3. CHALLENGES

In order to implement a database server that understands MySQL queries, we had to study and understand the internals of MySQL RDBMS. Therefore, we will present as follows the essentials from the client-server MySQLs protocol specifications.

MySQL client-server protocol has several phases. The first phase is the connection phase. The MySQL server listens for connections on a TCP / IP port or on a local socket. After a MySQL client connects to the server, it follows the authentication handshake mechanism between them. If the connection is successful, the session begins. The client sends an SQL command and the server responds with a set of data or an appropriate message to the type of command that was sent. When the client has finished the execution, it sends a special command that tells the server that it's over, and the session is terminated.

The basic unit of communication at the application level is the packet. All amount of information which is exchanged between client and server are encapsulated in packets which can have 16 MBytes maximum size. A query can be contained by a packet, but the servers response may be included in one or more packets. In terms of compression, there are two types of packets: compressed and uncompressed. The decision on which one will be used for the session is taken during the handshake phase and depends on the capabilities and settings of both client and server.

A compressed packet consists in a compressed header and the effective content, which may be either compressed or uncompressed. A compressed packet will have an additional field of 3 bytes, that contains, among others, the length of the compressed packet body part that follow. An uncompressed package will have the effective content immediately after the header.

All the packets are divided into two categories: commands sent by the client and answers returned by the server. The generic packets which contains servers response are: OK package, error package or end of file (end data flow)

packet. SQL queries are responded to with text based responses encapsulated in one or more MySQL packets.

4. RDBMS PROXY IMPLEMENTATION

We have implemented a transparent proxy system for interconnecting MySQL client applications to SQLite databases. We are talking about an application which was developed using a MySQL database, but for certain reasons it does not have any more access to MySQL server. Our final goal was to make the application able to use a SQLite database. SQLite is represented by a simple file which lies on the server. One big advantage of SQLite is that it does not need any installation or configuration and it does not require any dedicated service database management.

This transition can be made without any change to the application source code, and more interesting, without it being aware of it; so the application will run in the same manner as if running MySQL server.

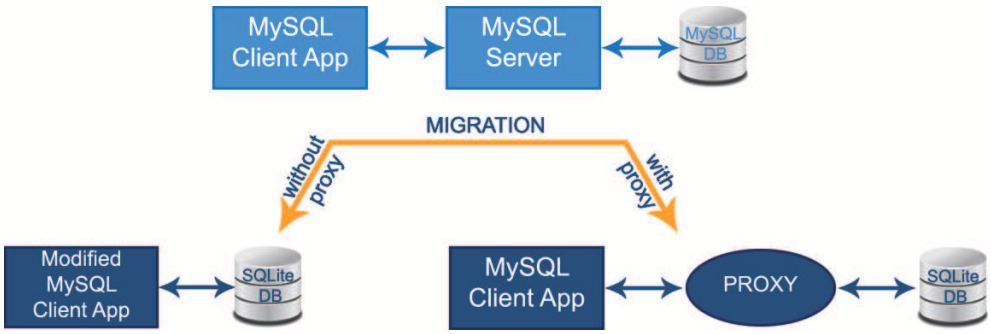

FIGURE 1. RDBMS migration

4.1. **Proxy design.** In order to make the proxy compatible with a MySQL client, the exchange of messages must comply with MySQL clientserver protocol specifications [7], as presented in Figure 2 and described below.

MySQL protocol requires a handshake exchange of messages between the client and the server. Upon a TCP connection was established, the server presents to the client a welcome message and waits for an authentication message. If the authentication attempt is successful, the server responds with an OK packet. In our prototype implementation we accept any combination of user / password. An authenticated client will then issue a SELECT_VERSION query to which the proxy responds with its version and capabilities.

Then, the proxy will enter a query-response loop. Once a client was authenticated, and aware of servers capabilities, it can issue MySQL queries and wait for responses from server. A query of type COM_QUIT will close the connection and end the communication between the client and the proxy.

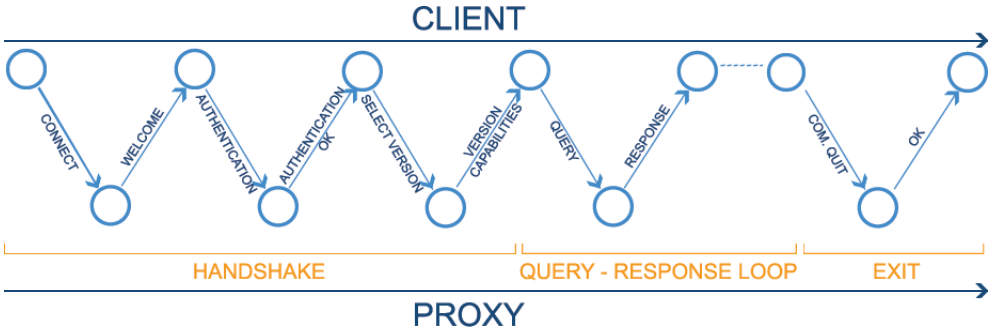

FIGURE 2. Proxy Logical Design

4.2. **MySQL to SQLite translation.** The role of the proxy system is to accept MySQL requests, to forward the corresponding SQLite requests, and upon receiving SQLite responses to convert them to MySQL responses that can be understanded by the client. In order to achieve this goal, our proxy must be able to encapsulate/decapsulate MySQL packets, as defined by the MySQL clientserver protocol.

Some MySQL client commands do not require a text contained response from the server, but a confirmation of successfulunsuccessful execution. Hence, for commands such as "create database", "drop database" and "use database", the proxy will execute the corresponding SQLite commands and reply with an OK_packet. Such a packet is encapsulated in PHP using the pack() function as presented below:

pack("C*",0x07,0x00,0x00,0x01,0x00,0x01,0x00,0x02,0x00,0x00,0x00);

In conformity with the format of an OK_packet the hexadecimal values of the given parameters convey the following semnification:

- the first 3 bytes (0x07,0x00,0x00) define number of bytes in the packet (in Little Endian representation).
- 0x01 is the packet number in the current packet sequence.
- 0x00 is the reserved value for an (0x07,0x00,0x00).
- 0x01 is the number of rows affected by the command.
- the remaining bytes represent state flags sent from server to client in conformity with the MySQL clientserver protocol.

Table 1 depicts (bolded) the encapsulation of an OK_packet captured with the tcpdump tool.

11:16:28.370669 IP 193.231.20.76.3306 > 193.231.20.76.44605: Flags [P.], seq 1:12, ack 22, win 342, options [nop,nop,TS val 315951189 ecr 315951178], length 11
0x0000: 0000 0000 0000 0000 0000 0000 0800 4500 ....E.
0x0010: 003f 23d7 4000 4006 6a7b c1e7 144c c1e7 .?#.@.@.j{...L..
0x0020: 144c 0cea ae3d cb66 756f 0593 ac8a 8018 .L...=.fuo......
0x0030: 0156 ac98 0000 0101 080a 12d5 0855 12d5 .V...........U..
0x0040: 084a **0700 0001 0001 0002 0000 00** .J**.....**

Table 1: OK_packet

Another example is the encapsulation of a "select" query, presented in table 2.

14:06:48.021527 IP 193.231.20.76.46178 > 193.231.20.76.3302: Flags [P.], seq 260:292, ack 1308, win 392, options [nop,nop,TS val 340106102 ecr 340090493], length 32
0x0000: 0000 0000 0000 0000 0000 0000 0800 4508 ....E.
0x0010: 0054 2353 4000 4006 6ae2 c1e7 144c c1e7 .T#S@.@.j....L..
0x0020: 144c b462 0ce6 76bb 5740 53a9 ce25 8018 .L.b..v.W@S..%..
0x0030: 0188 acad 0000 0101 080a 1445 9b76 1445 ....E.v.E
0x0040: 5e7d **1c00 0000 0373 656c 6563 7420 2a20** ^**}.....select.*.**
0x0050: 6672 6f6d 2061 6e64 7265 6561 5461 626c **from.andreeaTabl**
0x0060: 6531 **e1**

Table 2: Select Query

4.3. **Command execution examples.** In order to validate the proposed concept we developed the PHP code for decapsulating MySQL requests, executing (using the Linux system() call) SQLite commands, interpreting the result and encapsulating it as MySQL text responses.

In table 3 is an example of an MySQL client connecting to our proxy, issuing MySQL queries and getting the expected responses. The client is unaware that it is talking to a PHP program and not with the real MySQL server.

4.4. **Usage scenario.** Here we describe a scenario for using this proxy for a web based database application.

Considering a web page that does no longer have access to the MySQL server, the required steps for using SQLite as database storage are presented below:

- convert the MySQL database to SQLite;
- copy the SQLite database onto the server;
- copy the SQLite binaries onto the server;
- copy and execute the PHP proxy from command line or a web page.

If the proxy can not be started on the standard MySQL port (3306), then the web page must use a different port for connecting to the database server. No other modifications are required to the client (web) application.

```
ap@athena:~/public_html/radu$ mysql -A –host 193.231.20.76
Welcome to the MySQL monitor. Commands end with ; or '\ g'.
Your MySQL connection id is 11
Server version: Andreea MySQL Community Server (GPL)
Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\ h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
+————————-+
| Database    |
+————————-+
| a2          |
| andreea     |
| database    |
| mydatabase  |
| radu        |
| test        |
+————————-+
6 rows in set (0.01 sec)
```

```
mysql> create database new;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> show databases;
+————————-+
| Database    |
+————————-+
| a2          |
| andreea     |
| database    |
| mydatabase  |
| new         |
| radu        |
| test        |
+————————-+
7 rows in set (0.01 sec)
```

```
mysql> drop database new;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> show databases;
+-----------+
| Database  |
+-----------+
| a2        |
| andreea   |
| database  |
| mydatabase|
| radu      |
| test      |
+-----------+
6 rows in set (0.01 sec)
```

```
mysql> use mydatabase;
Database changed
```

```
mysql> show tables;
+-----------------+
| information_schema |
+-----------------+
| myfirsttable    |
+-----------------+
1 row in set (0.02 sec)
```

```
mysql> select * from myfirsttable;
+---+-----------+
| id | nume     |
+---+-----------+
| 1 | andreea   |
| 3 | radu      |
| 4 | diana     |
+---+-----------+
3 rows in set (0.03 sec)
```

```
mysql> quit
Bye
```

Table 3: An example of a MySQL client connecting to our proxy.

5. CONCLUSIONS AND FUTURE WORK

This paper overviewed some issues related to migrating one application from one RDBMS to another. Also, we described the implementation of a prototype that validates the concept of interconnecting MySQL and SQLite databases through a proxy server written in PHP. The proposed system allows database access operations without any modification to the client application which had previously used a database server, but it does not longer have access to the MySQL server.

We also highlighted the usefulness of such a proxy, especially because there are no changes which should be made to the application, so that the application may not even be aware that it is running without database server.

The described application is just a proof of concept. In order to make it complete and ready for deployment, one may consider implementing, as future work, all MySQL specification commands, packets compression and encryption, along with all related security issues. The proxy could also be enhanced to process multiple concurrent MySQL connections.

References

[1] G. Allen, M. Owens, The Definitive Guide to SQLite (2nd ed.), Apress, 2010.
[2] S. Haldar, Inside SQLite, O'Reilly Media, 2007
[3] S. Jeon, J. Bang, K. Byun, S. Lee, A recovery method of deleted record for SQLite database, Personal and Ubiquitous Computing, 2012, Volume 16, Issue 6, pp 707-715.
[4] M. Kofler, MySql, Apress, 2001
[5] A. Maatuk, V. Ali, N. Rossiter, Relational Database Migration: A Perspective, Springer-Verlag Berlin Heidelberg, DEXA 2008, LNCS 5181, pp. 676-683.
[6] V.F. Pais, V. Stancalie, Using web services for remote data access and distributed applications, Fusion Engineering and Design, 2006, Volume 81, Issues 15-17, pp 2013-2017.
[7] `http://dev.mysql.com/doc/internals/en/client-server-protocol.html`
[8] `http://www.SQLite.com/`

Babeş-Bolyai University, Department of Mathematics and Computer Science, 1 M. Kogălniceanu St, 400084 Cluj-Napoca, ROmania
E-mail address: `radu.dragos@ubbcluj.ro`
E-mail address: `diana.halita@ubbcluj.ro`
E-mail address: `andreea_puscas06@yahoo.com`