Disconnected Recordsets

and

their Implementation in Java

Vasile Braileanu
Teacher Coordinator: Adina Cocu

Abstract

This paper describes the disconnected recordset approach for database access. Also, it describes a Java library written by author from scratch, who implements some of the interfaces needed for disconnected recorsdsets in Java. Similar technology is available for almost all databases. Although a database is designed to be connected all time with the user, producing the data which user needs to consume, sometimes is most efficient to disconnect temporarily the user data from database and use data disconnected from database. Later on, data changes can be reflected in database implementing specific synchronization schemas in database server part.

Introduction

Today's businesses and activities are strong tied with computer databases, which is one of the most studied and used concept in computer technologies. From their beginning as flat text files up to modern databases, they all share the same need: to store, analyze and modify in shortest time all information stored. Major software companies have developed their own databases, like Microsoft, Oracle, IBM to mention a few. Free non commercial databases are available too. Also, linked with databases, there were developed a lot of software libraries for administering and accessing the databases in different programming languages and operating systems.

In this paper an aspect of database synchronization and a possible solution is described, used sometimes to solve some common issues. The same concept, of data disconnected from source is used in human history to access other information, like newspapers, books, sound and video records. Sometimes this method tends to be slow, but the concept exists and it is used because of his simplicity. On the other part, on a day to day basis there are heavily used systems that works connected with the user, like telephony, radio and television.

One of the important aspects of data producing and consuming is the connection type between data producer and consumer. Considering the time of connection, a consumer can access the database permanently (asynchronously) when needed but also it can “disconnect” from the data source and access the information “offline”. Later on, the information updated in the disconnected database can be synchronized with the main database. There are scenarios when this later approach is feasible and needed.

In section one, Database Synchronization describes the concept of data synchronization and some of the problems which can occur on using a permanent asynchronous connection between database and user.

Section two, Implementing the solution is reflects the steps which conducted to the solution implemented in Java, and the recordset storage file format.

Section three explains the implementation technologies used, the script used in server side and similar implementations.

Section four is the case study of a database who cannot be accessed remotely all the time.

Section five, Performance and Issues highline some of the gains and the loses of the disconnected recordsets.

1 Database Synchronization

Databases are one of the most important objects used everyday. Even if we don't use databases directly, they are everywhere. Today, any real world application uses at least one form of data persistence. The Relational Database Management System (RDBMS) is the most used persistence storage mechanism and uses SQL for query and data manipulation.

In current database implementations, a connection must be established with the database server. Exceptions are the embedded databases that are linked directly with the applications and are installed with them. Also, to use the database we send SQL or native commands to the database asynchronously. That means we need the connection to manipulate the database.

In practice, sometimes there is a need to read, update and insert a new record even if a link with the database cannot be established.

On the other part, anyone who uses databases will probably have to work with a particular recordset over a period of time. However, it would not be practical or desirable to maintain an open connection to the database all time, because this would use server connection resources.

Here are few simple examples:

- The database at an ISP cannot be accessed remotely because of the database access rights. But, the database server can be accessed from web pages inside web server. That means a database connection can be made locally in server. The database can be accessed on regular basis writing server side software that accesses the database. The server side part can receive the commands and execute, later sending results back to the caller. This can be easily done sending GET and POST messages to a web page that is backed server side.
- At a supermarket sales point, there are cash registers. Cash registers are connected with a database all the time. What happens if the server is offline? Cash registers cannot send their data in this scenario. We need a cached record for the payments at the cash register. This cached record can be stored in cash register computer then updated in database when the server is online.
- In a remote area, a sales point is connected through a wireless connection to the Internet and from there to the database. In case the wireless connection is not available, the database cannot be updated. The records are stored locally until the wireless connection will be available in order to update the database.
- Accessing the database with many connections, especially when we need to query it will slow down the database server. In some applications will be more efficient to store locally the dataset needed. For example a localized application, where all the strings for a language must be fetched from database, the translation time will be considerably reduced if there is only a database query which maps the strings for a particular language.
- Finally, creating a database in a small device, maybe mobile phone with a RDBMS will be cumbersome (and maybe impossible) if the database is small and there is no disk space in the device.

All these scenarios are somehow common in real world. If we want to share information, let say an article, we can print it. From printed form, it can be read by many but when concurrency is high the information cannot be accessed easily.

That is exactly the same as database server bandwidth available. Of course, today databases are state of the art programs, as we expect from them, but the above scenarios are not uncommon. We can share the bandwidth but at one point the database server can crash because of many connections (or at least will slow down). Some of the database operations can be done in client side if we want to.

2 Implementing the Solution

Considering the scenarios described above, we can imagine a result set stored locally in a file. Also, because this result set must be accessed through a connection, it will be useful to be represented in a form of a real recordset identical with the database recordset so there is no need to implement the whole database access programs again.

Because in Java is implemented JDBC, it will be useful to access the recordset in identical form with the other database recordsets, thus creating a recordset from this file in same manner.

In Java, the user can take advantage of already available Java packages to create recordsets disconnected form the main database. There are also other databases that offer natively disconnected recordsets.

2.1 Introducing the disconnected recordset notion

These specific recordsets are, as the name suggests, recordsets that have been disconnected temporarily from the server, thus allowing the user to work off-line and move freely between records.

If a disconnected recordset is created with write permissions, the user can also add, modify or delete records. These changes will be cached locally, mainly in files and they will not affect the main database. Later on a connection can be re-established to the database, which can then be updated with the changes.

One possible problem which can arise is the possibility of records conflict with other recordsets, online or offline. That could happen also in non transactional databases.

One requirement of disconnected recordsets is that they must be maintained by the database client, rather than the database server.

2.2 File Format

Data can be stored in files in any format, but XML is one of the most heavily used format today and many of the actual technologies uses it to store information. In practice, XML will be a common choice. Today, many databases can export tables and other data as XML files.

Considering XML penetration in network transmissions (even Apache helicopters are using XML streams for data communications between helicopter and ground), XML it is a smart choice.

Being portable and simple, XML is used both in files and streams. The disadvantage of space is somehow tempered by the simplicity of format and human readability.

2.3 Format Manipulation

Because XML is a very used format and the algorithms for manipulating it are very simple, today there are libraries for XML manipulation in all programming languages.

In Java we have multiple choices for parsing and generation. A standard library in Java, which is included in JDK is SAX. SAX model is based on events generation, like other implementations in other languages. In implementation described in this paper, SAX was preferred because the structure of the XML was very simple and because is faster than DOM. Also, DOM uses a lot of memory which can be a limitation for storing data organized as tables like we need in a recordset. DOM uses a tree for supporting the XML content and that tree resides in memory. Using SAX accessing the information is faster, reading the whole XML file in one step then preserving it in memory. For simple and small tables this will be a simple task.

DOM, on the other part is more useful when we need a complete structure reflecting the elements inside XML.

2.4 Programming Language

What programming language is the most used today? The simple answer is Java. There are a lot of reasons why many programmers chose Java. To mention a few, is portable, offers a superior speed than scripting languages being compiled for JVM, forces exception treatment and has access protection to code. A lot of code was written in Java, both for desktop and server and there is a tremendous quantity of documentation and samples. Also, the JDK is free to use, being shipped with many free available IDEs.

A natural choice, Java was chosen to implement a disconnected recordset with XML input and output.

3 Implementation

Disconnected recordsets are not new. Few years ago, Sun Microsystems included in their SQL libraries for Java some implementations that support them. At the top of them is a class, WebRowSetImpl who implements WebRowSet interface. It is available from JDBC 3.0, which was introduced in JDK 5.0. Someone who uses this class will notice that WebRowSetImpl is proprietary and it could be removed from next versions. That not happened yet but happened in the future. WebRowSet implementations can offer serialization if needed, so they can be sent through networks or stored on disk.

We could also choose ADO 2.0 and stick with Microsoft technology to create disconnected recordsets or any other available technology if we want to.

As a sample code, a simple WebRowSet implementation was written, who contains base functionality for the recordset, including common data insert, update and delete, input from and output to XML. Due to the complexity of the code the current implementation does not store the data streams and national strings or blob and clob fields. Future implementations could solve this. Also, for a better functionality it will be useful to write specific implementations for databases. Some of the databases use specific date format which cannot be easily detected.

A simple PHP script was also written for MySQL database to provide a very simple way to receive and send database commands and data through GET and POST methods, generating simple XML pages conforming to Java WebRowSet XML schema.

4 Case Study - A Database Server cannot be accessed remotely

As a case study, it was considered a MySQL database who cannot be accessed remotely because the lack of rights. The database was located in a remote server who can be accessed only server side. It is the standard case of web server serving pages at a web services provider.

For testing purposes, a small Java program sent an SQL command through POST, querying the database and returning the recordset. Server side script, written in PHP, returned in HTTP response the WebRowSet XML schema (webrowset.xsd) conformant XML. WebRowSet filled the records and modified some of them, saving another XML file. This last recordset, stored in XML can be accessed like other recordsets from other databases, even to be sent back to the server if there is a service that can analyze and modify the real table in database.

Testing procedure:

1. Sample query (sent to PHP script on remote server): ”select * from tranzactii”
2. XML recordset has been sent back to Java program (according to standard schema)
3. WebRowSet implementation written in Java executed some operations on recordset and returned the new XML as text file.

Commands executed:

```
WebRowSetVB wrs = new WebRowSetVB();//implemented disconnected recordset
Reader buffer = new BufferedReader(reader);
wrs.readXml(buffer);

//test delete
wrs.absolute(2);
wrs.deleteRow();

//test insert
wrs.moveToInsertRow();
wrs.updateInt(1, 10000001);
wrs.updateString(3, "test.....test");
wrs.insertRow();

//test modify
wrs.absolute(1);
wrs.updateInt(1, 20);
wrs.updateString(3, "modify test");
wrs.updateRow();

//write result
FileWriter fstream = new FileWriter("test.xml");
BufferedWriter out = new BufferedWriter(fstream);
out.printtln(wrs.getXMLString());
out.close();
```

XML response after recordset commands:

```
...
<modifyRow>
        <columnValue>1</columnValue>
        <updateValue>20</updateValue>
        <columnValue>1</columnValue>
        <columnValue>2536 3040 3020 1034</columnValue>
        <updateValue>modify test</updateValue>
        <columnValue>1</columnValue>
        <columnValue>Mon Dec 20 06:48:00 EET 2010</columnValue>
        <columnValue>0.6</columnValue>
</modifyRow>
<deleteRow>
        <columnValue>2</columnValue>
        <columnValue>1</columnValue>
        <columnValue>2536 3040 3020 1034</columnValue>
        <columnValue>1</columnValue>
        <columnValue>Mon Dec 20 06:50:49 EET 2010</columnValue>
        <columnValue>0.6</columnValue>
</deleteRow>
...
<insertRow>
        <columnValue>10000001</columnValue>
        <columnValue></columnValue>
        <columnValue>test.....test</columnValue>
        <columnValue></columnValue>
        <columnValue></columnValue>
        <columnValue></columnValue>
</insertRow>
...
```

5 Performance and Issues

5.1. Benefits

- Considering the application area and memory involved, implemented WebRowSet contains a vector of records in memory. All data is stored in memory, which makes sense because the recordset is not large.
- XML format is very simple to manipulate, consisting of text fields.
- It uses standard schema and other applications can use the XML format generated. Using a standard schema facilitates the transfer between generated format and other formats.
- Less database traffic, only the selected information is sent through network.

5.2 Liabilities

- XML file format is large and this can be an important factor on distributed applications. Data sent through network can be sent also in a binary form or at least compressed on server and decompressed on client and vice versa.
- As we expect, storing in memory all values from recordset limits the application area, and the supplementary data coming from field properties and result set properties (metadata) will slow down the application. Because the memory in client is not as large as the server's, and usually the server only caches a small chunk of recordsets or none the application area is limited to those applications which stores in memory only a small part of the database. Similar server caching schemas can be used to reduce the memory used.
- Accessing data is slower than in database server. Database servers are powerful computers comparing with client machines, data is in binary format and is cached inside server memory.

Conclusion

Described implementation is similar in functionality with WebRowSetImpl from Sun, implementing the same interface. The common XML content used as output and input format simplifies the using of the disconnected recordsets. Because disconnected recordsets need to be synchronized on the server part and the implementation depends on the database, implementation use some PHP server side code for connecting a MySQL database. Also a mapping between MySQL data types and JDBC data types was created to permit the creation of the recordset. In the future versions the implementation could be completed with data streams to permit the storage of objects in serialized form.

Acknowledgement: I wish to thank Lect. Dr. Eng. Adina COCU for her patience and guidance in writing this paper and the programming code associated with it. Also I wish to thank her for his Java course which we taken in 2010 at the University "Dunarea de Jos". I wish to thank for programming course to Asist. Drd. Eng. Diana STEFANESCU (C/C++), and Associate Prof. Dr. Emilia PECHEANU (Linux).

References

[1] Java API
[2] Deepak Vohra, *Oracle Web RowSet,* http://www.packtpub.com, 2009

[3] Infinite Software Solutions Inc., *Tutorials – Using disconnected recordsets,* http://www.devguru.com
[4] Sharad Acharya, *Making the Most of JDBC with WebRowSet,* http://onjava.com, 2006
[5] McObject LLC, *McObject's eXtremeDB High Availability Embedded DatabaseGains Role In Boeing AH- 64D Apache Longbow Helicopter,* http://www.in-memory-database.net, 2004

VASILE BRAILEANU
Low Danube University
Faculty of Computer Science
111 Domneasca St. , Galati
ROMANIA
E-mail: vasilebraileanu@yahoo.com