QTJDB - Quick and Tiny Java Database Documentation

Lukasz Grzegorz Maciak

December 13, 2004

1 Project Description

QTJDB is a simple database application. It uses a flat file database, and employs a client server type architecture. I allows for more than one user to connect to the server and manipulate the file at the same time. The GUI client allows the users to display all records, update chosen record, delete a record and insert a new one.

2 Project Goals

To implement a small database application with following functionality:

- Client-Server Architecture supporting concurent connections from more than one client.
- Flat file database structure for data storage (ideally with variable lenght records and variable length fields).
- Data manipulation (reading in sequence, lookup by primary key, insertion, deletion, updates).
- Graphical User Interface (JTable representation of data on screen).
- Dynamic on-screen data manipulation (ie. changes to the JTable are reflected in the data file).
- Maintaining Data Constraints (on primary key especially).

3 Design

Please consult the attached class diagram

4 Client Server Architecture

The client-server arhitecture is done using java Socket class. The server spawns a new thread every time it detects a client connection. This way concurrent

operations are possible. I attempted to use synchronized methods to prevent stalling or freezing, but concurent writes might still be buggy.

At this stage QTJDB client does not maintain a persistent connection with the server. Connection is extablished when needed, and closed after the required data exchage.Therefore client-server transaction usually takes very small ammount of time, minimizing the chance for concurent write to occur. Most of the time the server communicates clients one at a time. Due to time constraints the persistent connection mode was abandoned, in favor of this temporary-need-based connection scheme.

This trait also prevents an effective user-login scheme (as login information would have to be verified many times during one user session - and that is not a desirable behavior). Therefore development of the login module was deffered indefinitely.

Communication between the server and the client is accomplished using 1 byte signals defined in QTJDBLib.java. Usually a client sends a signal, and waits for a response from the server. This response is either data, or a response signal such as ACK or GO AHEAD (prompting client to send a data stream).

All data is sent in form of byte streams (arrays). This allows to treat a record as a functional data unit that is transmitted and then parsed on the client side. The choice to send data this way was largely dictated by the data file structure (variable length records delimited by special characters encoded as character strings).

5 Database Structure

QTJDB can work with any human readable text file which follows the guidelines bellow:

- File is formated with Unix Style Line Endings (Apple or Windows line endings might work, but are not recomended).
- First Record (record 0) contains the Column Names and no other data.
- The first field of a record is the primary key (all values must be unique).
- The primary key is a character string representing numeric value which can be parsed into an int using Integer.parseInt method.
- All record fields (except for the last one) are delimited by tab characters.
- Last field of the record is delimited by a newline, marking the end of the record.

No other requirements are necessary. All the fields except the primary key are treated as character strings for all intents and purposes. The fields and records are variable length. In essence, QTJDB was designed to work with a human readable text file.

Deletions are accomplished by marking the first byte of the record with a tombstone. The tombstone is a byte value or 0. Some editors (notably kwrite) loose the ability to render the file when they encounter raw byte values. Others (vi) display them as strings of strange characters. The human readability of the file therefore deteriorates with many deletions taking place.

Insertions to the file attempt to reclaim the deleted space preventing fragmentation. However, space is reused only if the new record is a perfect fit. Managing partial fragments left over after inserting smaller records proved to be troublesome and greately increased the complexity of the algorithm. If no perfect fit is found, the record is appended to the end of the file.

6 Data Indexing

QTJDB maintains two index files. The first file holds the adresses of all the records with the respective Primary Key values. The second index holds addresses and respective sizes of all the data blocks marked for deletion. Index files are read into memory at the beggining of each transaction, and then written back into the file after it is finished. The first byte of each index file is the validation byte. It is set to 0 at each read, and set to 1 at each write. Therefore if the program crashes before being able to write to the index file, it will be recreated from the data file.

7 Data Constraints

QTJDB is designed to work with a wide variety of data files and therefore the only supported constraint is the primary key constraint. Insetion or update of a record with a duplicate id value will be rejected. All other fields are treated as basic character strings. Since the column datatypes might be diferent from one fule to another QTJDB does not enforce any constraints on them. So even if the file seems to have numeric field, or date fields - any input will be accepted.

This behavior is intended, as it adds flexibility to the db application. More rigorous constrait mechanism can be built in by expanding the record 0 to include datatype metadata. This was not implemented in the current version due to time constraints.

8 GUI

The GUI allows the user to perform all the essential operations on the data file. Please note that the client connects to the server only when it is triggered by a user action, and therefore it wont know that the server is not running until such action is taken. The server can be shut down remotely by using the GUI but naturally there is no way to start the server from the GUI (as it will most likely be on a remote machine).

The updates and insertions are done dynamically by catching events on the JTable which displays the data. For insertion all other buttons are locked till the user finishes to input data into all the fields, and hits the insert button again.

Update is done immediately after the user finishes to edit the current cell.

The GUI utilizes dialogs to communicate errors to the users or provide other usefull information.

9 Instalation and Usage

Please consult the README file. Please note that the client must be started from the command line with two parameters representing the name or IP address of the machine hosting the server, and the port on which to initiate communication.

The server can be shut down gracefully by sending a signal from the client. But, the server is robust enough so that it can be killed from the OS level without any negative effects as long as no transactions are taking place at the time.

It is possible for the server to stall waiting for client input if the client crashes in mid tansaction, or the transaction is interupted prematurely. In such case the only way to shut down the server is to do it from the OS level. To shut down the server simply close all the client connections and kill the server process.

In all the other cases it should be possible to shut it down using the client.

10 Development Notes

Please note that the client server-side algorithms are still very much work in progress. Ideally I would like to have persistent connections which open as soon as the client starts, and close only when the client session is terminated. Ideally these sessions would be protected by a user-login scheme and the data communication would be encrypted using some type of fully reversible crypto-hashing algorithm. In addition I would like to have a more strict constraint mechanism - wich would store metadata about the field datatypes, and validate the user input. These fetures however are well beyond the scope of this project.