Abstract: J2ME is a development platform for mobile devices and has been introduced by Sun Micro-Systems Inc. In 1999 for programming limited devices such as phones, PDAs and other small devices. But this architecture does not support APIs for data persistence management and relational database because of its limitations. This paper presents a base for local relational database and data persistence management for J2ME based applications and can be used in any database aware application in J2ME platform. In this paper and implementation of this database system mobile device and J2ME limitations have been considered. Also, the B+tree indexing structure has been implemented in this project that allows us fast insertion, deletion, range queries, search, backup and restore mechanism for RMS based databases.

Keywords: J2ME; B+tree; mobile devices; data management; storage; relational; database.

1 Introduction

The mobile devices are going to be more and more popular today and the need for data oriented software for them is growing very fast. So the need for a fast and acceptable manner of storing and retrieving data has been more obvious in the past few years.

The mobile applications should be interactive in a way that they could respond to user actions. For an application that uses database and storing and retrieving data, this is an important issue. since storing and retrieving data if not implemented in a good way, can be very lengthy and time consuming this definitely will not let mobile platform to succeed. Also, a mobile application Should be available in off-line mode as it is in on-line. This issue is important if either the cost of the data transmission over the network is high or the network is not always available. Also, the network speed is also an issue. Hence a data persistence man-
agement system is needed in order to store and retrieve information and data in the mobile device [3].

We have chosen J2ME as our development platform because the number of mobile devices that support this platform has been increasing in the past few years. The JAVA virtual machine that is implemented in the mobile devices is called KVM (K for Kilobyte).

J2ME virtual machine is nearly installed on every mobile devices that has been manufactured nowadays. Thus, for devices that does not come with KVM by default, the custom releases by third party manufacturers are available (like IBM J9 for Windows Mobile and Palm-OS and JBED and Net-Mite for Android).

But J2ME does not support JDBC, because the classes that implement JDBC are very heavy and complex for such limited devices. In J2ME there are some options available that are JSR75 for file connection API and Record Store Management (RMS). JR75 needs special permissions to access disk directly and is not a good option for database implementation. So the later is used in this project, that is , this system is built on top of the RMS and uses RMS low level API’s to store and retrieve data in the background. We know that there is not a unique solution for database management that supports every kind of mobile devices. That is, all the designed systems are specifically designed for one type of the system (for example Symbian, Android, Palm-OS and Windows Mobile database management mechanism are different).

So some solutions for this lack of existence of unique DBMS has been developed. Among them, IBM ToolBox for J2ME [76] for remote database access, JT-Open [2], Point-Base Micro by Data-Mirror [3], CodeBase Micro [4] are more important.

This implementation here uses the B+tree indexing structure, an indexing structure that with hashing mechanism are the most powerful indexing mechanisms today and most of powerful database management systems such as SQL Server and Oracle use this type of indexes in their databases. In the next section the database structure and implementation will be described and section 3 will introduce an application for this database system. Section 4 suggests other implementation ideas and will discuss future works.

2 System Internals

Till now, some information about the platform and limitations has been discussed. In this section we are going to study the internal structure of this system.

2.1 Overall system analysis

Record Store management is a mechanism for storing and retrieving data in a J2ME application without having special permissions. This API is a low level API and the programmer should manipulate bytes directly. RMS supports some basic operations and its structure is similar to Simple File. The programmer should pack bytes directly and then write them to database [1]. There is not any good disk data structure for data storage like BTree, B+tree, Hashing, etc.

| Simple SQL             |
|------------------------|
| RMS B+tree Database    |
| RMS                    |
| J2ME Midlet            |

Table 1: System Architecture

This basic API support operations like adding record, update a record by ID, access a record by ID, record filters and record enumerations. But these operations and their performance are not good enough for database applications like accounting systems or dictionaries or something like them. Also, creating applications that need relational databases is nearly impossible with these simple operations [1]. The main goal here is to extend RMS capabilities for search, backup, restore and high speed query of the stored data. As mentioned earlier one of the most important data structures for database system indexing is B+Tree. This indexing method allows fast searches, high speed range queries, addition and deletion of data with minimal disk access. This system also supports object serialization that is not supported directly in J2ME.

2.2 Add Mechanism

Two types of addition are supported in the system. The difference between them is number of records that are being added to the database at one time. First one is add record method that inserts a record into database and updates the B+tree index accordingly.
This one is the fundamental method for adding data in most applications. The other one is used when there are multiple records waiting to be added to the database and also they share the insertion key. For example consider an accounting table for factor items that we are buying and suppose that the key is factor number.

In this situation factor items share a common key between them, so instead of accessing disk multiple times, addition could be done via 2, 3 or more disk accesses, of course, this will reduce disk accessed from 20 to 2 or even smaller with respect to database properties. This will save the program some time that in the mobile platform, this time saving is considerable. Of course another type of addition is also supported that this will be discussed in next sections. This mechanism is called Bulk Loading and this will be used mainly for Backup and restore of the saved data [2].

### 2.3 Search and Range Query Mechanism

The system supports both types of exact search and range query and uses B+tree properties. Since all data are indexed search and range queries are performed directly on the index. This will speed up this type of actions. We also know that B+tree structure has very high performance in performing range queries. Search and create result set methods are implemented to perform these operations and the programmer should interpret the results. Test results on random data are presented here [2].

### 2.4 Deletion Mechanism

Two types of deletion is common in database programming world. One of them is physical deletion and the other one is marking [4]. In this system the later is used. That is, when a record is deleted, it will not be deleted physically. This method is used by many DBMS systems nowadays. Of course this method has some space overhead, but considering size of memory and space this days, this is not a big deal for database management systems. On the other hand, the database will be compacted in backup and restore operations. That is, this marked records will be deleted physically when performing these operations. Unfortunately J2ME does not support services for running them in background, hence we do not have any real time mechanism for compacting data whenever the program is not active [2].

### 2.5 Updating Records

Updating records is a bit tricky. Update can be performed on a record and changes different parts of a record. If changes affect the key part of a record, the location for the record in the B+tree index should be updated. Otherwise, updating non key part of the record is sufficient.

### 2.6 Backup and Restore

If we have a large collection of records, and we want to create a B+tree on some field, doing so by repeatedly inserting records is very slow. Bulk loading is a technique to overcome this shortcoming. Bulk loading can be done much more efficiently. This technique is used on sorted data entries and usually is applied on a large number of records. In this system we gain some improvements on number of disk accesses while implementing bulk loading technique. These improvements are achieved via caching higher B+tree items for inner nodes in main memory and write them to disk at once instead of writing them one by one. For this purpose a queue has been used to store a list of upper levels for as many levels as exists and writing process uses this queue for minimizing the number of disk accesses.

One of the applications of this method is importing data from other database systems and restoring data from system backups. Furthermore compacting database for physical deletion of records that are marked for deletion is possible via this method. Also, this method for back up and restoring data is the best method for this purpose, because accessing RMS file directly and manipulating it in J2ME world is impossible. So, using this mechanism will minimize number of disk accesses and hence will minimize elapsed time for these operations [2].

### 3 Case Study: J2ME High Speed Accounting Software

For testing system, a complete accounting software had been developed. The complete software had been ported from Windows platform to J2ME and their operations are identical. All operations are done in real time. First step is creating the database schema. For defining schema and opening a table, field sizes for the records of the table are needed. So in the constructor
of the table field sizes are passed to table creator. After this step, database is ready for all kind of actions. After defining all the tables and creating the schemas, tables will remain open until the programmer closes them. This is done for speeding up the program by reducing number of opening and closing operations on tables. In this software a simple relational database model had been used. Of course, the relational model still needs more work. Taking reports is done via carrying out joins on tables manually.

All listings in this software are sorted. This is one of the interesting properties of B+tree indexes as mentioned earlier. The database system in this software has special properties. A background queue works simultaneously with the main program to perform lengthy database operations in background to keep the user interface responding. This background thread starts with main program and usually is used on operations which does not affect next possible actions or actions whose probability are low. The system with real world data and about 30 tables had been tested and the program speed and performance was good. This database system can be used on any database application like dictionaries, accounting systems, etc.

4 Discussion and Future Works

This system is still young and many new features can be added to it. An example is Simple SQL language manager for insert, search, query, update, delete and other operations like creating joins. Also, the future of database indexing technology is possibly UBTree. However, because of limitations of mobile devices and J2ME, porting SQL completely to this platform and implementing a complete UBTree system is nearly impossible. Thus, network management and accessing remote databases is an important matter for a DBMS. Network manager for a database system based on J2ME should be created carefully, because mobile network and communication system is also limited compared to computers. Indexing on multiple fields separately is also an idea for next works. Although data exchange via Blue-tooth is another idea that is important for mobile devices.

Notes

[1] IBM toolbox for J2ME. Available at http://www.ibm.com/.
[2] T.Jopen Available at: http://sourceforge.net.
[3] PointBase Micro Available at:http://www.pointbase.com/.
[4] Oracle Lite. Available at http://www.oracle.com/.

References

[1] J. Keogh, J2ME: The Complete Reference, McGraw-Hill/Osborne, 2003.
[2] R. Ramakrishnam., Database Management Systems, McGraw-Hill.
[3] V. Kumar, Mobile Database Systems, John Wiley and Sons,Inc. Hoboken, New Jersey., 2006.
[4] Siddhartha Sen and Robert E. Tarjan, Deletion Without Re-balancing in Multiway Search Trees (2010), 125–128.
[5] R. Bayer and E. M. McCreight, Organization and maintenance of large ordered indexes, Acta Informatica 1 (1972), no. 3, 173-189.
[6] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, Data Structures and Algorithms, Addison-Wesley, Reading, MA, 1987.
[7] D. Comer, The ubiquitous B-trees, ACM Computing Surveys 11 (1979), no. 2, 121–137.
[8] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, Rockville, MD, 1993.
[9] J. Jannink, Implementing deletion in B+-trees, ACM SIGMOD Record 24 (1995), no. 1, 33–38.
[10] D. Knuth, The Art of Computer Programming: Sorting and Searching, Vol. 3, Addison-Wesley, Reading, MA, 1973.
[11] J.D. Ullman, Principles of Database and Knowledge-Based Systems, Computer Science Press, Rockville, MD, 1988.
[12] G. Wiederhold, Database Design, McGraw-Hill, New York, 1983.