AN EFFICIENT DATABASE SYNCHRONIZATION FOR MOBILE DEVICES USING SAMD ALGORITHM

Dr. Venkatesh. J [1], Aarthy. C[2]

[1] School of Management Studies, Anna University Chennai, Coimbatore Regional Office, Coimbatore, India. venkijv@gmail.com
[2] Full Time Ph.D. Scholar, School of Management Studies, Anna University Chennai, Coimbatore Regional Office, Coimbatore, India. mailtoaarthy@gmail.com

ABSTRACT: A Database Synchronization Algorithm for Mobile Devices recommends the SAMD (Synchronization Algorithms based on Message Digest) algorithm based on message digest in order to aid data synchronization between a server-side database and a mobile database. The SAMD algorithm makes the data at the server-side database and the mobile database uses message digest tables to compare two data's in order to select the rows required for synchronization. If the two data's are different, the synchronization progresses according to synchronization policy. The SAMD algorithm does not use techniques that are reliant on specific database vendors: neither does it use triggers, stored procedures or timestamps. The SAMD uses only the typical SQL functions for the synchronization. Therefore the SAMD algorithm can be used in any mixtures of server-side database and mobile database because of its objectivity of database vendor.
[Venkatesh. J. **AN EFFICIENT DATABASE SYNCHRONIZATION FOR MOBILE DEVICES USING SAMD ALGORITHM.** *Life Sci J* 2012;9(3):2470-2476] (ISSN:1097-8135). http://www.lifesciencesite.com. 357

Key Words: Synchronization, adaptability, mobile database, client, server.

INTRODUCTION:

Recent advances in mobile technology and equipment have led to the emergence of a new computing environment and a variety of small sized mobile devices such as PDAs (personal digital assistants), smart mobile phones, HPCs (handheld PCs) and Pocket PCs have been popularized. As various network technologies are increasingly being associated with such mobile devices, the processing of business information can be available using mobile devices. As a result, business models that rely on mobile technologies are appeared. Mobile devices do not have much computing power and rely on batteries. Additionally, constant access to network is difficult due to narrow bandwidth. Therefore, it is not easy to process a large size of stored data and maintain a continuous connection with the server-side database. For these reasons, mobile devices have mobile databases in order to achieve stable data processing. Mobile devices download replications of limited data from a connected server-side database using a synchronization device that has a stable wire communication function. Mobile devices process various tasks using the data downloaded in an off-line state. The work on the network disconnected condition is a crucial point for mobility support.

In a disconnected environment, there are inevitable inconsistencies between the server-side database and the mobile database. Synchronization techniques can solve the data inconsistencies and guarantee the integrity of the data. Consequently, synchronization is an essential subject in mobile device computing environments. Commercial DBMS venders offer various solutions to data synchronization in a mobile environment. However, these solutions are not independent of the server-side database because they use database dependent information such as metadata or use specific functions of server-side database such as trigger and time stamp. In other words, the mobile database vender should be equivalent to the server-side database vender. The solution of operating a separate synchronization server in the middle tier is independent of the server-side database but dedicated to the mobile database. That is, the synchronization solution and the mobile database should be the identical vender product. Because of these restrictions, the extensibility, adaptability and flexibility of mobile business systems are markedly decrease. This problem must be solved in order to build efficient mobile business systems because upcoming mobile environments will have heterogeneous characteristics in which diverse mobile devices, mobile databases, and RDBMS exist.

1.2 Synchronization:

Good Synchronization is a file synchronization tool that enables user to backup user files and/or synchronize them with another location such as a different drive, folder, USB drive, network location or Amazon S3 storage account. The program supports one-way and two-way synchronizations, it can propagate deletions and check for conflicts, which can be reviewed and approved before executing the synchronization. User can also execute

jobs automatically when a removable device is connected, periodically, or schedule via Task Scheduler. Other features include file filters and exclusions, support for time shifts, chained syncing, backup of deleted files and more.

1.3 Smart Synchronization Mechanism:

The synchronization method for all applications based on Smart Synchronization has the following features:

Many of the features in the synchronization components of Smart Synchronization are intended in such a way that they promise in-order message processing and one-time message delivery at all times. The message delivery assurance is the accountability of the synchronization layer. The architectural diagram for smart synchronization is represented in Fig: 1. each message from a device is measured in Smart Synchronization with an eminence. The administrator can view each of the messages in different states.

Smart Synchronization is a context that is particularly intended to offer the development tools and synchronization features essential for offline applications. Smart Synchronization methods are, therefore, usually accomplished in asynchronous mode in the SAP MI Server Component. Synchronous mode for synchronization is also available. The basic synchronization mechanisms are indistinguishable for both synchronous and asynchronous mode. Technically, the same messaging protocol is used for both modes. The variance lies in the timing for expecting response from the server. In the SAP MI Client Component, there is no application-specific coding that directly interacts with the synchronization process. Application-specific coding is only necessary when dealing with local data using the MI Client APIs. For application data that is stored locally in the persistence layer, the underlying components of the MI Client APIs keep the device data synchronized with the back-end. Error and conflict handling are exceptions. To handle synchronization errors using application-specific logics, Smart Synchronization offers features to access details error and conflict information.

2. SYSTEM ANALYSIS:

2.1. Existing System:

The existing system faces certain issues in algorithm like clock synchronization, event based, non-block synchronization and default synchronization algorithm which are discussed as follows.

Clock synchronization:

Clock synchronization is a hindrance from computer science and engineering which deals with the indication that internal clocks of several computers may vary. Even when primarily set precisely, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. There are several hitches that occur as a consequence of rate variances and several solutions, some being more suitable than others in certain situations. In serial communication, some people use the term "clock synchronization" just to deliberate getting one metronome like clock signal to pulsation at the same frequency as another one frequency synchronization and phase synchronization. Such "clock synchronization" is used in synchronization in telecommunications and instinctive baud rate detection.

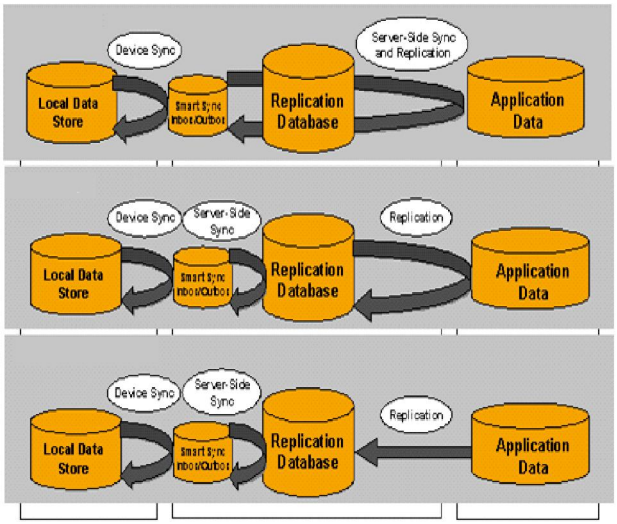


Fig: 1 Architecture of Smart Synchronization

Problems:

In addition to the erroneousness of the time itself, there are difficulties related with clock skew that take on more intricacy in a distributed system in which several computers will need to comprehend the same global time. For instance, in UNIX systems the make command is used to assemble new or revised code without the need to recompile unchanged code. The make command customs the clock of the machine it runs on to regulate which source files need to be recompiled. If the sources exist in on a separate file server and the two machines have unsynchronized clocks, the make program might not produce the correct results.

Event-based synchronization:

A synchronization method and apparatus describes event objects to permit synchronization of execution units (e.g., threads). In one procedure, the synchronization method and apparatus is used in aggregation with a UNIX operating system. By describing event objects on which threads or other execution objects can wait upon, multiple threads can

wait on one event, or otherwise, one thread can wait on multiple events. Besides, using the event-based synchronization method and device, it is conceivable to specify behavior, mainly when one thread or other execution object waits on multiple events. For example, the performance indicated can be that a condition is gratified if any of the events occur, if all of the events occur, or some other logical combination of events occurs.

Non-Block Synchronization Algorithm:

Java provides supports for additional atomic operations. This allows to develop algorithm which are non-blocking algorithm, e.g. which do not require synchronization, but are based on low-level atomic hardware primitives such as compare-and-swap (CAS). A compare-and-swap operation checks if the variable has a certain value and if it has this value it will perform this operation. Non-blocking algorithm are usually much faster than blocking algorithms as the synchronization of threads appears on a much finer level (hardware)

Default Synchronization Algorithm:

The default synchronization algorithm starts when an attention identifier (AID) key is pressed. An attention identifier (AID) key is any key that generates a presentation space update. Primarily, the state of the terminal is UNINITIALIZED. The procedure waits for a period of time for updates to the presentation space. User can modify the wait time in the Timeout field in the preferences window. The nonappearance wait time is 1200 milliseconds. If Timeout is set to 1200 milliseconds, and an update arises during the last 600 milliseconds, the process waits for an additional 600 milliseconds for extra updates. During this extra wait period, added update occurs during the last 300 milliseconds, the algorithm waits again for another 600 milliseconds for further updates. This continues until no updates are received during the last half of the last additional time period. At this point, the state of the terminal is either LOADED (keyboard locked) or READY (keyboard unlocked), reliant upon the OIA status. Thus to conclude the existing system faces few disputes like slow indexing, poor extensibility, flexibility, low synchronization, reliant on database vendors, less secured, high cost.

2.2 Proposed System:

To overwhelm the above hitches we have hosted the algorithm SAMD (Synchronization Algorithms based on Message Digest). SAMD resolves synchronization glitches using only standard SQL queries as trained by the ISO. This is monitored by a possible synchronization of any data combination irrespective of the kind of server-side database or mobile database. The SAMD thus would offer extensibility, compliance and flexibility. The SAMD makes the data's at the table of the server-side database and the mobile database using a message digest algorithm; then the data's, and the message digest values are saved in the message digest tables on both sides.

3. SYSTEM IMPLEMENTATION:

The Fig.2 represents a synchronization framework using a synchronization server in a mobile business environment. The whole framework consists of a server-side database, synchronization server (AnySyn) and multiple mobile devices with internal mobile databases. The server-side database maintains all of the data required for business, and the mobile database downloads copies of data the user needs from the server-side database. The synchronization server is located between the two databases to synchronize the data and manage additional information required for synchronization. The AnySyn synchronization server performs synchronization based on the SAMD algorithm. The synchronization policy is established in AnySyn, and the load caused by accessing the server-side database is minimized by operating a connection pool. Every mobile device uses a separate toolkit to access the AnySyn server over a wired network to perform synchronization.

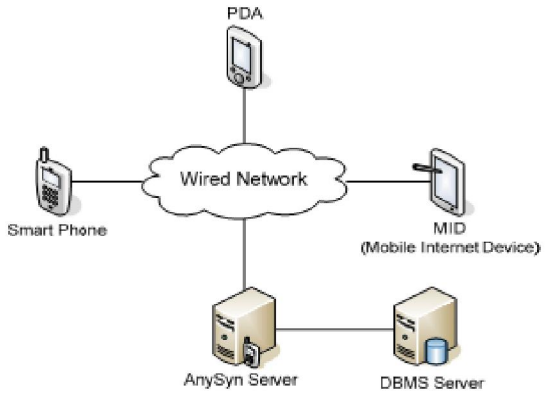


Fig .2 SAMD Synchronization Framework

Row Inconsistency:

An inconsistency refers to a state in which the published data in the server-side database and the subscribed data in the mobile database carry different values due to a change at either side. The two databases add, delete and modify data independently, which makes inconsistency inevitable. The Table.1 displays every case for an inconsistency for a single row.

Among the 16 cases indicated in Table.1, Cases 6, 7, 8, 10 and 14 include the ADD operation, which cannot occur for a single row. For example, in Case 7 the row added at the server side is different from the row modified at the client; therefore, it cannot be

considered an inconsistency. Case 7 is equivalent to Case 3 and Case 5 occurring independently. Similar reasoning can be made for Cases 6, 8, 10 and 14, so SAMD does not consider the five cases.

C	Mobile DB	DB Server	C	Mobile DB	DB Server	C	Mobile DB	DB Server	C	Mobile DB	DB Server
1	UC	UC	5	UC	ADD	9	UC	MOD	13	UC	DEL
2	ADD	UC	6	ADD	ADD	10	ADD	MOD	14	ADD	DEL
3	MOD	UC	7	MOD	ADD	11	MOD	MOD	15	MOD	DEL
4	DEL	UC	8	DEL	ADD	12	DEL	MOD	16	DEL	DEL

(C:Case, UC:Unchanged, ADD:Added, MOD:Modified, DEL:Deleted)

Table 1: Inconsistency Table
(Source: Mi-Young Choi et.al)

Message Digest Algorithm:

Message digest algorithm consists of a unidirectional hash function that maps a message of a random length to a fixed-length hash value. Message digest h is created by the hash function H, which can be expressed in equation (1):

$$h = H(M) \rightarrow (1)$$

M is a message of a random length and H (M) is a fixed-length message digest. Even a single bit changed in the message causes a change of message digest value.

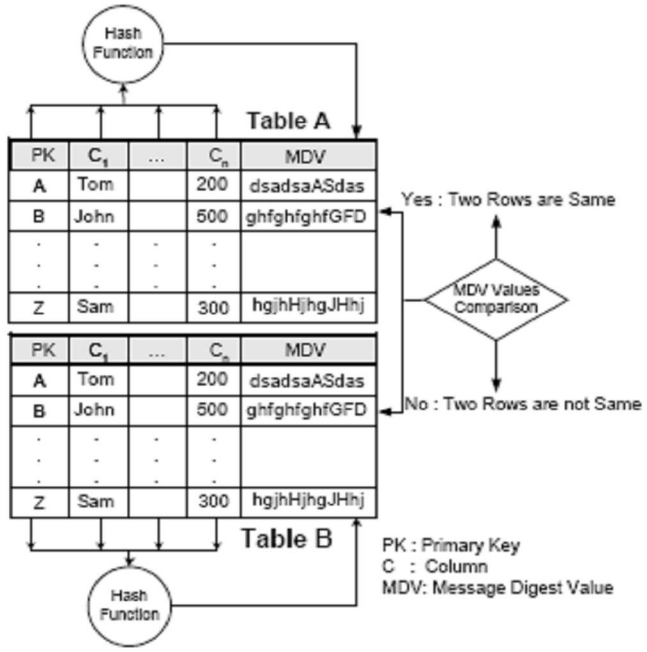


Fig. 3. Message Digest Mechanism
(Source: Mi-Young Choi et.al)

The Fig. 3 demonstrates how this message digest mechanism can be applied to a relational database to examine data identity between rows of two tables. Data in two rows are identical if two rows in Tables A and B have identical message digest values. If the two values are different, it means that the two rows have one or more different column values. Accordingly, this method can be useful in detecting inconsistency between two rows once a row with an inconsistency is detected; the row is copied using the primary key in the direction of synchronization according to the synchronization policy. This synchronization algorithm identifies a modified row without relying on the database's internal functions, logs or metadata to enable synchronization that is independent of the database vender.

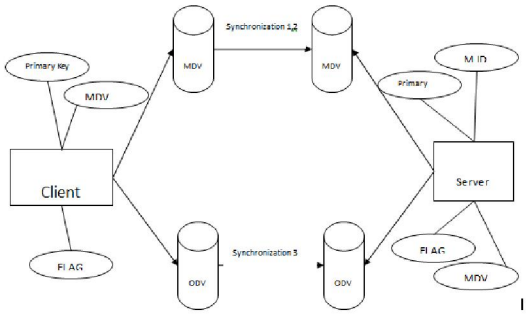


Fig. 4 Algorithm Design

SAMD synchronization algorithm satisfies the following objectives.

1) Independence of vendors.

2) Synchronization using only standard SQL statements.

3) Disallows schematic modification of data table of the server-side database.

4) Disallows adding restrictions in implementing applications.

5. SAMD Algorithm Explanation:

SAMD Synchronization Algorithm displays (Fig.4) the table schema of the server-side database and the mobile database where the SAMD synchronization algorithm is applied. Both databases have a data table (DSDT: Database Server Data Table, MCDT: Mobile Client Data Table) and a message digest table (DSMDT: Database Server Message Digest Table, MCMDT: Mobile Client Message Digest Table). The data table contains the business data, and the message digest table stores the message digest value from the data table. The message digest table consists of a PK column of data table, message digest value (MDV) column, flag (F) column and mobile device ID (Mid) column. The flag column signals an inconsistency that has occurred in

the corresponding column; therefore, the flag column is used to identify a row that requires synchronization. The mobile device ID is a unique number of the mobile device, so this column is used to identify a mobile device that requires synchronization. In SAMD Synchronization Algorithm, if a row's PK value is A1, this value is identical to the two message digest values and there is no need for synchronization. However, if a row has a PK value of C1, the value of MDV in MCMDT is different from the value of MDV in DSMDT and the MCMDT flag value is 1.

Consequently, synchronization is necessary. The synchronization process is performed for each row to resolve all of the inconsistencies mentioned. For instance, if there is a discrepancy in row C1, synchronization proceeds from the mobile database to the server-side database and DSDT's PK C1 row is swapped with the MCDT's C1 row. Thus, the two are same synchronization algorithms applied to diverse tables. Here, the message digest values that are created with each row value of the data table, and the message digest values of the message digest table, are associated. If the values are alike, there has been no modification in the data and synchronization is not required. If the values are different, it means that the data table value has been changed, in which case the message digest table has to be updated with new message digest values and the flag has to be set to 1. Flag value is used to identify a row that needs synchronization. The server-side database has one DSMDT for every DSDT. Although the size of the MCMDT is smaller than that of the DSMDT, there is an MCMDT for every mobile device that has a unique ID. It is very inefficient to perform Synchronization 2 for every row of the DSDT every time there is a synchronization request from a mobile device. Therefore when the mobile device requests synchronization, the mobile device ID value is sent to the server-side database and then the SAMD algorithms select the row from DSMDT whose value of mid column is the same as the mobile device ID value and Synchronization 2 is only applied to the selected rows. For example, a mobile device whose mobile device ID value is 'md1' requests synchronization, the rows whose value of mid column is 'md1' are selected and then only used in Synchronization 2. After SAMD algorithms analyze the type of inconsistency using the flag values of both messages digest tables, primary key, which is used to identify the row. Therefore, Synchronization 3 is performed between two data tables for each inconsistent type. Upon completion of synchronization, the flag of the synchronized row is set to 0 in the message digest table. Most mobile devices have limited resources, and the load on the device should be minimized during the synchronization process.

Accordingly, all message digest tables are located in the server-side database to economize storage space of the mobile device, as shown in Fig., while there is the load caused by accessing the network in Synchronization 1 but the data size of MCDT is smaller than the server. Furthermore, the MCDT data necessary for Synchronization 1 is sent to the server-side database in a single transmission over a wired network using an SQL query capable of batch processing. After this point, there is no load on the mobile device, which reduces the load caused by network access in the Synchronization 1 stage. The following snap shots explain in detail about the process undergone and the output obtained in each synchronization stages.

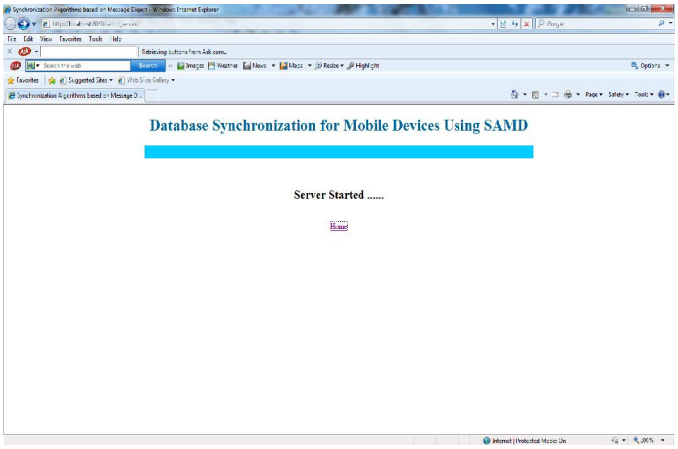


Fig. 6.SAMD and Server Starting Page

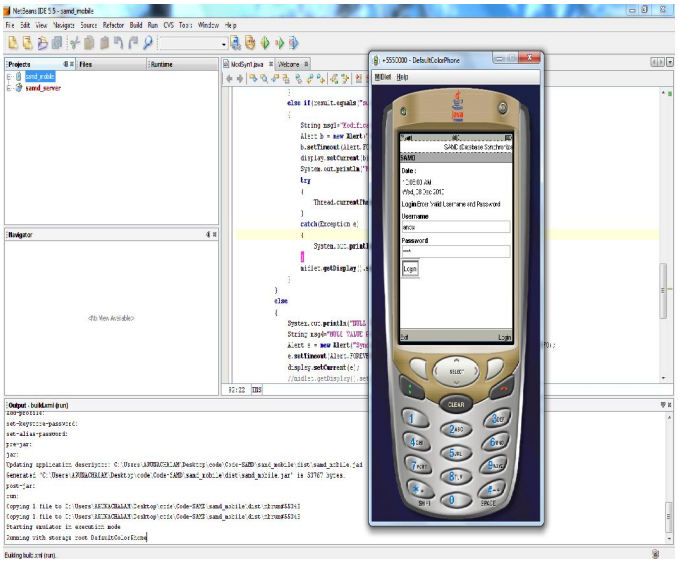


Fig.7 Login Page

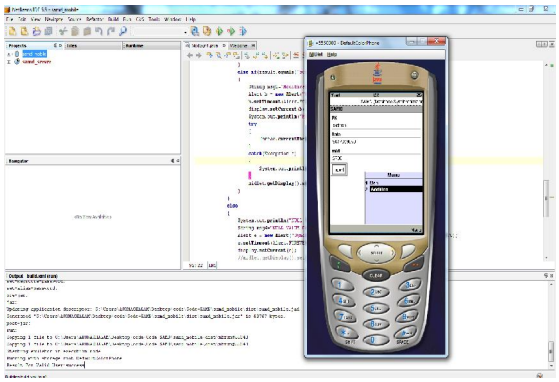


Fig.8 Insert Page

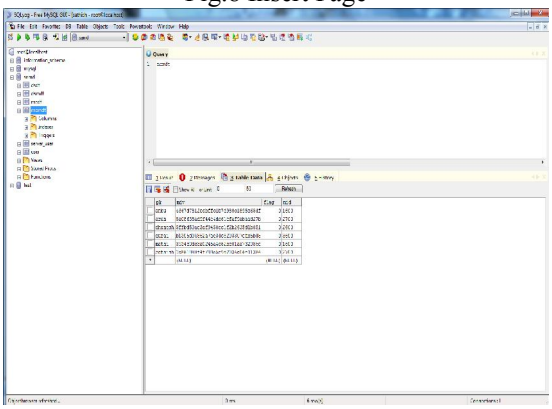
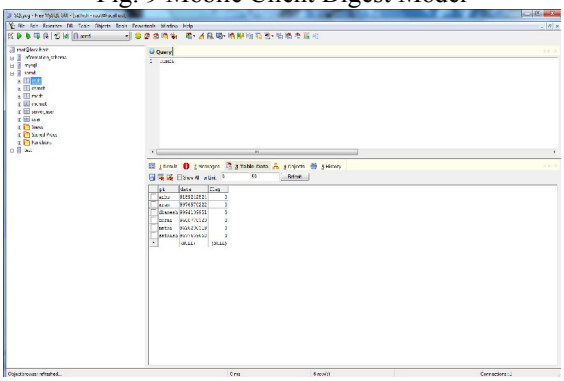


Fig. 9 Mobile Client Digest Model



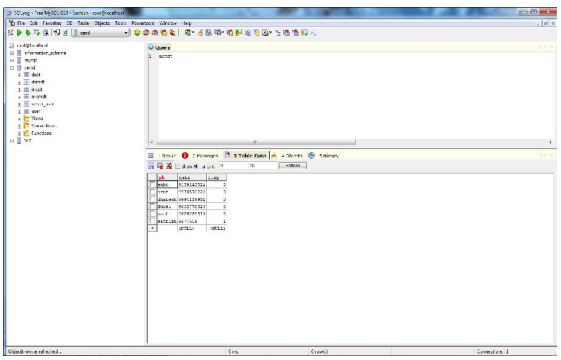


Fig. 10 Comparison of Database Server Message Digest Table and Mobile Client Message Digest Table

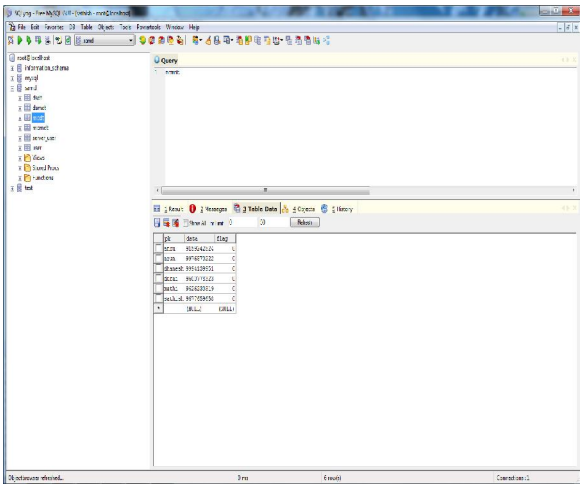


Fig. 11 Mobile Client Message Digest Table after Modification

CONCLUSION:

This work suggests an SAMD synchronization algorithm based on message digest for synchronizing between server-side databases and mobile databases. The SAMD algorithm is performed with only SQL functions of relational databases, so that it is not dedicated to particular vendors and is available for use in combination with any server-side databases and mobile databases. Therefore, extensibility, compliance and flexibility are assured when a mobile business system is ratified. This feature is vital in order to build effective mobile business systems since the upcoming mobile business situation has varied features in which diverse mobile devices, mobile databases and RDBMS exist.

FUTURE ENHANCEMENTS:

The work can be enhanced by integration of heterogeneous application to mobile devices. The system can use more volatile and robust Dynamic database storage improvement in mobile clients. The system can be further established for simultaneous and multiple applications processing.

REFERENCES:

1.Gye-jeong kim, seung-cheon baek, hyun-sook lee, han-deok lee,moon jeun, Joe (2006), "LGeDBMS: A small DBMS for embedded system with flash memory", 32nd international conference on very large data bases, pp.1255-1258,.

2. Joshua savil, (2008), "Moblink Synchronization Profiles', A white paper from Sybase iAnywhere, October 17.

3. Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik, (2009),

"A synchronization algorithm for mobile devices for ubiquitous computing"
4. Mi-seon choi,young-kuk kim, juno chang (2008), "Transaction-centric split synchronization mechanism for mobile E-business applications", April 20th.
5. Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik (2010), "A Database Synchronization Algorithm for Mobile Devices", Vol. 56, No. 2, May 2010.
6. Santashilpal Chaudhuri, Amit kumar saha, David B.Johnson (2007), "Adaptive clock synchronization in sensor networks", March 31st.
7. Xianzhong tian;younggang miao; Tongsen HU; Bojie fan; Jian pian; wei xu (2009), "Maximum likelihood estimate based on time synchronization algorithm for wireless sensor networks".
8. Ziad itani, Hassan diab, Hassan Artail (2005), "optimistic pull based replication for mobile devices".

9/7/2012