Sync Algorithm

Preliminaries:

1. There is one Mater DB (MDB), several Devices (D_i), each having a Local Database (LDB_i).
2. We can never trust the local clock of the Device, and would have to completely rely on THE MDB's clock to assign proper Timestamp (if any).

Specifications of MDB:

1. It has a table containing the UID field, All the Info Fields (Irrelevant to the syncing algorithm) and a few Sync Algorithm aiding fields for **each record in the table**. All records are *Timestamp* sorted. The table has an index on the UID field.

2. It has a few **Master Attributes** that are the same for every record of the table, i.e. the attribute is of the table and not of any single/multiple records in particular.

3. The **Sync Algorithm aiding fields** in the MDB's table are as follows:
   a. **Timestamp** (DATETIME data type as in SQL LITE): This keeps an entry about when that particular record was last time updated (added / deleted / edited).

   b. **Record_Exists** (BOOLEAN data type): If any particular record is deleted, it would not be removed from the MDB, but just simply this Field would be set to FALSE (0), so as to be able to report this to other devices that this entry no longer exists and has been removed.

4. The **<u>Master Attributes</u>** of the MDB are as follows:
    a. **Last_Update_Timestamp** (DATETIME data type as in SQL LITE): This attribute would facilitate any Device to know if it needs to sync or not. It would be equal to the maximum (most recent) value in the *TIMESTAMP* field of the MDB's table.

Specifications of LDB$_i$:

1. It also has a table containing the UID field, All the Info Fields (Irrelevant to the syncing algorithm) and a few Sync Algorithm aiding fields for **each record in the table**. All records are *Sync_Timestamp* sorted. The table has an index on the UID field.

2. It also has a few **Master Attributes** (**LDB$_i$ specific**) that are the same for every record of the table, i.e. the attribute is of the table and not of any single/multiple records in particular.

3. The **Sync Algorithm aiding fields** in the LDB$_i$'s table are as follows:
    a. **Sync_Timestamp** (DATETIME data type as in SQL LITE): This value is from the server's clock and after sync it is updated to the same value as the record's value in the MDB.
    b. **Edit_Timestamp** (DATETIME data type): This would contain the time when that particular record was last edited. It would be equal to the *Last_Sync_Timestamp* + *TIME_OFFSET*.
    Whenever a device syncs this value is set to zero. Once this field is SET for any record, its *SYNC_TIMESTAMP* field becomes of no consequence.

    Note: The amount of time elapsed would have to be taken care of, if we are dealing with the Devices clock changing at multiple times.

    c. **Record_Exists** (BOOLEAN data type): If some particular record is deleted this field is simply set to FALSE (0). And the record entry is deleted from the LDB$_i$ only when at some sync the *MDB->UID$_k$ ->Record_Exists* and *LDB$_i$->UID$_k$->Record_Exists* are set to FALSE.

4. The **Master Attributes** for each LDB$_i$ are as follows:
    a. **Last_Sync_Timestamp** (DATETIME data type as in SQL LITE): This would help the Device to know whether it has to sync with the MDB or not. It would be equal to the largest (most recent) value of the *SYNC_TIMESTAMP* field of LDB$_i$'s table.

b. **Last_Edit_Timestamp** (DATETIME data type as in SQL LITE): This also would help the Device to know whether it has to sync with the MDB or not. It would be equal to the largest value of the EDIT_*TIMESTAMP* field of LDB_i's table. Whenever a device syncs this value is set to zero.
c. **Time_Offset** (DATETIME data type as in SQL LITE): This would contain the time offset between the MDB's clock and the Device's clock, to facilitate the calculation of the *EDIT_TIMESTAMP* value in each record.

Algorithm Specifications:

1. The MDB may be interacted with by a device only at the time of sync and would require a semaphore to be acquired before proceeding.

2. The database can be only updated locally, and then be synced to reflect the changes in the MDB.

3. Whenever a Device (D_i) syncs with the MDB and a conflict for some UID_k occurs, then the most recent value (according to the timestamps associated with the UID_k in the MDB and the LDB_i) of the record would be considered.

4. The Algorithm assumes that at all times the most recent data is the correct one.

5. The algorithm offers no rollback option, but if required it can be tweaked further to add the functionality.

6. There are in total 3 ways that the Database can be updated locally:
    a. **Add**: A new record can be added to the LDB_i if only UID_k isn't present in the table already or if present, *Record_Exists* field is set to FALSE, i.e. it had been deleted earlier.
    The *LDB_i->UID_k->Edit_Timestamp* field would be accordingly updated.

    b. **Delete**: An already existent record (UID_k) can be deleted from the LDB_i, doing so only requires the *Record_Exists* field to be set to FALSE. And the record would be truly removed from the LDB_i, only when the *MDB->UID_k ->Record_Exists* field is also set to False and the *LDB_i->UID_k->Edit_Timestamp* field would be accordingly updated.

c. **Edit**: An already existent record UID_k can be edited by a Device (D_i) and updated to a new value. The *LDB_i->UID_k->Edit_Timestamp* field would be accordingly updated.

7. Every time a Device (D_i) syncs with the MDB, its action can be strongly divided into three subgroups:
   a. Records which are **more recent in the LDB_i (if any)**, and have to updated in the MDB.
   b. Records which are **more recent in the MDB (if any)**, and have to be update in the LDB_i.
   c. For **all other records(if any)**, nothing has to be done.

The Algorithm:

The Process has to main categories:

- Updating LDB_i on device D_i (A record is added/deleted/ or edited on Device (D_i)).
- Syncing LDB_i with MDB.

Updating LDB_i:

1. **Addition:**
   a. Once the UID is fetched into the application on Device D_i, it queries for it in the LDB_i. If no such record is found or the record is found but the *Record_Exists* field is set to FALSE, then the Application would give the user an option to add the entry.
   b. Upon addition, the corresponding fields of Information would be filled in by the user.
   c. The Device would then, set the value of the *EDIT_TIMESTAMP* field to (*LAST_SYNC_TIMESTAMP* + *TIME_OFFSET).*
   d. The *SYNC_TIMESTAMP* field is of **No consequence** now, but is set same as the *EDIT_TIMESTAMP* field.
   e. The *RECORD_EXISTS* field is set to TRUE (1).
   f. The *LAST_EDIT_TIMESTAMP* is updated to the same value as the *EDIT_TIMESTAMP* field of the record.

2. **Deletion**:
   a. Once the UID is fetched into the application on Device D_i, it queries for it in the LDB_i. If a record is found and the *Record_Exists* field is set to TRUE, then the Application would give the user an option to Delete or Edit the record.
   b. Upon deletion, the *RECORD_EXISTS* field would be set to FALSE.
   c. The Device would then, set the value of the *EDIT_TIMESTAMP* field to (*LAST_SYNC_TIMESTAMP* + *TIME_OFFSET).*

d. The *SYNC_TIMESTAMP* field is of **No consequence** now, but is set same as the *EDIT_TIMESTAMP* field.
e. The *LAST_EDIT_TIMESTAMP* is updated to the same value as the *EDIT_TIMESTAMP* field of the record.

3. **Editing:**
   a. Once the UID is fetched into the application on Device D_i, it queries for it in the LDB_i. If a record is found and the *Record_Exists* field is set to TRUE, then the Application would give the user an option to Delete or Edit the record.
   b. Upon selecting to Edit, the user would edit the Information fields corresponding to the UID.
   c. The Device would then, set the value of the *EDIT_TIMESTAMP* field to (*LAST_SYNC_TIMESTAMP* + *TIME_OFFSET).*
   d. The *SYNC_TIMESTAMP* field is of **No consequence** now, but is set same as the *EDIT_TIMESTAMP* field.
   e. The *RECORD_EXISTS* field is unchanged and still set to TRUE (1).
   f. The *LAST_EDIT_TIMESTAMP* is updated to the same value as the *EDIT_TIMESTAMP* field of the record.

Syncing LDB$_i$ with MDB:

In every interaction with the MDB a temporary copy of the MDB would be created on the Server, a semaphore would be requested by the Device, which would be granted by the server only if the MDB is not currently in use by another Device.

If the network fails in between, or somehow the sync is not successful then the MDB (forcefully) resets the Semaphore to be free, and the server trashes the temporary MDB copy.

Else if, the transaction is successful then the server keeps the temporary copy of the MDB, and trashes the previous one, and then acknowledges the device's request to release the Semaphore, and finally frees the semaphore.

1. A Device D$_i$ tries to sync with the MDB.
    a. D$_i$ requests Server to access the MDB, by requesting for a semaphore.
    b. If the semaphore is free to be acquired, the server blocks it. Else it denies the device access.
    c. If the semaphore was free, then the server creates a temporary copy of the MDB (tempMDB).
    d. The server grants the device the semaphore to access the tempMDB.

2. The Device has acquired the Semaphore:
    a. The device sends the *LAST_SYNC_TIMESTAMP* to the server.
    b. The server compares the value to its own *LAST_UPDATE_TIMESTAMP.*
    c. If the server's timestamp is more recent it sends all the records having *TIMESTAMP > LAST_UPDATE_TIMESTAMP,* and a copy of the server's current clock time.
    The device receives the data (tempDATA table) and sends acknowledgement to server.

d. Else the Server sends its own LAST_UPDATE_TIMESTAMP, to the device for update of the MDB and also the current clock time.
   The device again acknowledges this.
e. Device sets the *TIME_OFFSET* current clock time – Device's clock time.

3. If the server's timestamp was more recent then the device has received tempDATA table, containing all changes made in the MDB since the last time it synced.

   a. If the LAST_EDIT_TIME is set to zero, then the device sends a message to the server reporting that no changes have been made. The device doesn't care if it receives an acknowledgement from the server about the message. It jumps to step d.
   b. The device queries the LDB_i for each UID_k in the received tempDATA.
   c. For each record in the tempDATA where the UID already has an entry in the LDB, and the *EDIT_TIMESTAMP* field of the LDB's record is set to a non-zero value, and if *EDIT_TIMESTAMP > TIMESTAMP* on the tempDATA record. The Device saves the value of the UID in a list.

      The device before updating the LDB scans the complete tempDATA and prepares this list.

      i. Once the list is prepared, the device copies the records (whose UID is on the list) into a tempDATA_Update table and sends it to the MDB along with the value of the server's clock time received.
      ii. If the device receives an acknowledgement from the server about the data been received, it would proceed, else it would halt the sync and report error.

iii. If acknowledgement is received, then the device would set the *SYNC_TIMESTAMP* on each of those records to the server's clock time received and the *EDIT_TIMESTAMP* field to 0.
iv. For each of these records, if the RECORD_EXISTS is set to TRUE, it would be left unchanged; else in case it is FALSE the record would be deleted from the LDB.
v. Then the device would proceed to update the rest of the LDB.

d. The device again queries the LDB_i for each UID_k in the received tempDATA.
e. If an entry is not found and the *RECORD_EXISTS* field on the tempDATA is TRUE, device adds the record in LDB_i:
    i. The UID and all the info fields are copied onto the LDB.
    ii. The *EDIT_TIMESTAMP* is set to 0.
    iii. The *SYNC_TIMESTAMP* is set to the server's clock time.
    iv. *RECORD_EXISTS* is set to TRUE.

f. If an entry is not found and the *RECORD_EXISTS* field on the tempDATA is FALSE, device ignores it.
g. If an entry is found, if *EDIT_TIMESTAMP* field is 0 or if *EDIT_TIMESTAMP <= TIMESTAMP* on the tempDATA record and *RECORD_EXISTS* is TRUE:
    i. The UID and all the info fields are copied onto the record.
    ii. The *EDIT_TIMESTAMP* is set to 0.
    iii. The *SYNC_TIMESTAMP* is set to the server's clock time.
    iv. *RECORD_EXISTS* is set to TRUE.

h. If an entry is found, if *EDIT_TIMESTAMP* field is set to either 0 or if *EDIT_TIMESTAMP <= TIMESTAMP* on the tempDATA record and the *RECORD_EXISTS* field is set to FALSE. The device removes the entry from the LDB.

        i. The device sets it LAST_SYNC_TIMESTAMP to the current clock time received and the LAST_EDIT_TIME to 0.
        j. The device requests to release the semaphore.

4. If the server's timestamp wasn't more recent then the device has received the server's LAST_UPDATE_TIMESTAMP and the current clock time.
    a. If the LAST_EDIT_TIMESTAMP is set to 0, then the device sends a message to the server reporting that no changes have been made. If the device doesn't care if it receives an acknowledgement from the server about the message.

    b. Else the Device prepares a tempDATA_Update table containing all the records in which the EDIT_TIMESTAMP is set to a non-zero value. This table is sent to the server along with that a copy of the server's clock time received is sent back.
    If the device receives an acknowledgement from the server about the data been received, it would proceed, else it would halt the sync and report error.

    c. If acknowledgement is received, then the device requests to release the semaphore.

3. The server has received a tempDATA_UPDATE table or a message saying that no updates from device side, and a request to release the semaphore.
    a. If No updates from device side message is received, the MDB simply destroys the tempMDB copy and grants the request freeing the Semaphore.
    b. Else for each UID in the tempDATA_UPDATE:
        i. The server copies all the info fields from the tempDATA_UPDATE table onto the tempMDB.
        ii. The TIMESTAMP field is set to the current clock time received and the RECORD_EXISTS field is copied from the tempDATA_UPDATE table.

iii. The server then updates the LAST_UPDATE_TIMESTAMP to the clock time received.
iv. The server now grants the device's request and frees the semaphore.

SYNC is now complete.