CS 522—Fall 2020
Mobile Systems and Applications
Assignment Three—Databases

In this assignment, you will extend the chat server app from the previous assignment. The previous app just saves messages received in an array in the activity UI. If the user navigates away from the activity, and then returns to it, there is a good chance that the messages already received will have been lost, due to the way that Android manages resources in the activity life cycle. Another motivation is that we are so far violating one of the most basic tenets of Android programming, that no blocking operations should be performed on the main UI thread. In preparation for moving receipt of messages to background processing, we will in this assignment persist both messages received, and information about people that have sent us messages, to a database.

You should follow these guidelines for your implementation.

First, define a subpackage of your application, called `contracts`. Define classes in this package called `MessageContract` and `PeerContract`, that define (as final static `String` constants) the names of the columns in your database. For each operation, define a type-specific read operation that reads the value of the column from a cursor, and a type-specific write operation that adds the value in a column to a `ContentValues` object. For example:

```
public static final String MESSAGE_TEXT = “message-text”;

public static String getMessageText(Cursor cursor) {
   return cursor.getString(cursor.getColumnIndexOrThrow(MESSAGE_TEXT));
}

public static void putMessageText(ContentValues values, String text) {
   values.put(MESSAGE_TEXT, text);
}
```

Second, extend the `Message` and `Peer` entity classes that you defined in the previous assignment, with a constructor that initializes the fields from an input cursor, and a method that writes the fields of the entity to a `ContentValues` object:

```
Public interface Persistable {
   public void writeToProvider(ContentValues out);
}

public class Message implements Parcelable, Persistable {
   ...
   public Book(Cursor cursor) {
      this.messageText = MessageContract.getMessageText(cursor);
      ...
   }
   ...
```

```
    public void writeToProvider(ContentValues values) {
        MessageContract.putMessageText(values);
        ...
    }
}
```

Third, define a subpackage called `databases`. In this class, define the database adapter class `ChatDbAdapter`. This class defines:

1. Useful string literals such as `DATABASE_CREATE` (the SQL command to create the database), `DATABASE_NAME` (the name of the file containing the database), `MESSAGE_TABLE` (the name of the table containing message information) and `PEER_TABLE` (the name of the table containing peer information), and `DATABASE_VERSION` (an integer that is used for versioning your database).
2. A private static inner class called `DatabaseHelper` that extends `SQLiteOpenHelper` with your logic for creating the database and upgrading where necessary. Your adapter class, on instantiation, instantiates this helper class in order to obtain a reference to the database.
3. Useful application-specific operations for accessing the database:

```
public void open() throws SQLException;
public Cursor fetchAllMessages();
public Cursor fetchAllPeers();
public Peer fetchPeer(long rowId);
public Cursor fetchMessagesFromPeer(Peer peer);
public void persist(Message message) throws SQLException;
public void persist(Peer peer) throws SQLException;
public void close();
```

Fourth, in your main activity where you display the messages received in a `ListView`, use the `SimpleCursorAdapter` class to connect the cursor resulting from a query to the list view. There are two constructors for `SimpleCursorAdapter`. It is okay **for this assignment only** to use the first, deprecated constructor, that performs cursor operations (such as requerying the database if the activity is restarted) on the main thread:

```
public SimpleCursorAdapter(Context context, int layout, Cursor c,
                           String[] from, int [] to);
```

You should call `startManagingCursor` in your activity to manage the cursor you get back from a query, through the activity life-cycle (closing and requerying). Again, this method is deprecated, and **you should only use it for this assignment**.

You can use the predefined layout resource `android.R.layout.simple_list_item_2` as the layout resource for each row in your list view. This defines a layout of two text views, one above the other, for each row. Display the title and authors of each book:

```
String[] from = new String[] { MessageContract.SENDER,
                               MessageContract.MESSAGE_TEXT };
int[] to = new int[] { android.R.id.text1,
                       android.R.id.text2 };
```

When you receive a message, extract the name of the sender and where it came from (part of the UDP packet header information), and update the database record for this peer, or insert a record if this is the first time we have heard from this peer. A message record contains a foreign key reference to the peer record for the sender:

```
CREATE TABLE Peers (
   _id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   timestamp LONG NOT NULL,
   address TEXT NOT NULL,
   ...
);
CREATE TABLE Messages (
   _id INTEGER PRIMARY KEY,
   ...
   peer_fk INTEGER NOT NULL,
   FOREIGN KEY peer_fk REFERENCES Peers(_id) ON DELETE CASCADE
);
CREATE INDEX MessagesPeerIndex ON Messages(peer_fk);
CREATE INDEX PeerNameIndex ON Peers(name);
```

Note: You should **always** define table and column (and index) names just once as string literals, and then define a string expression that builds a SQL command such as above to create the database. You must define the secondary index on author foreign keys yourself, otherwise SQLite will perform linear searches of the database to enforce such constraints. By default, foreign key constraint enforcement is not enabled in SQLite. You can enable it *on each connection to the database* by executing:

```
db.execSQL("PRAGMA foreign_keys=ON;");
```

If you redundantly store a peer's user name along with the foreign key in a message record, you do not need to do a join when querying for a list of messages and their senders. The index on peer names is required for searching to see if a peer is already in the database, when a message from that peer is received.

As in the previous assignment, provide another UI (accessible using the options menu from the action bar) for displaying a list of the peers. This UI just lists peers by name in a list view. Rather than passing the list of peers as a parcelable list from the main activity, you should in this assignment just load the list of peers from the database (instantiating the database adapter in the activity for viewing chat peers).

If the user selects one of these peers, then provide in a third UI information about that peer (their currently known IP address, and the last time that a message was received from that user). Also display a list of messages just from this peer. You should pass the primary key for the peer (or the peer entity) to this subactivity.

Submitting Your Assignment

Once you have your code working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
2. In that directory you should provide the Android Studio projects for your apps.

In addition, record short mpeg or Quicktime videos of your apps working. Make sure that your name appears at the beginning of the videos. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*. The videos should demonstrate running the app, exiting the app, then rerunning the app and demonstrating that the saved state from the previous run (messages for the chat app) are still present when the app is restarted. Make sure that you send messages from several peers to the chat server.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have two Android projects, for the apps you have built. You should also provide videos demonstrating the working of your assignments, as well as a completed rubric.