CS371m - Mobile Computing

Persistence - SQLite
In case you have not taken 347: Data Management or worked with databases as part of a job, internship, or project:
Databases

• RDBMS
  – relational data base management system

• Relational databases introduced by E. F. Codd in the 1970s
  – Turing Award Winner

• Relational Database
  – data stored in tables
  – relationships among data stored in tables
  – data can be accessed and viewed in different ways
Example Database

- Wines

<table>
<thead>
<tr>
<th>Winery ID</th>
<th>Winery name</th>
<th>Address</th>
<th>Region ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Moss Brothers</td>
<td>Smith Rd.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Hardy Brothers</td>
<td>Jones St.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Penfolds</td>
<td>Arthurton Rd.</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Lindemans</td>
<td>Smith Ave.</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Orlando</td>
<td>Jones St.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region ID</th>
<th>Region name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>2</td>
<td>Yarra Valley</td>
<td>Victoria</td>
</tr>
<tr>
<td>3</td>
<td>Margaret River</td>
<td>Western Australia</td>
</tr>
</tbody>
</table>
Relational Data

- Data in different tables can be related—hence, *relational database*
**Keys**

- Each table has a key
- Column used to uniquely identify each row

---

<table>
<thead>
<tr>
<th>Winery ID</th>
<th>Winery name</th>
<th>Address</th>
<th>Region ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Moss Brothers</td>
<td>Smith Rd.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Hardy Brothers</td>
<td>Jones St.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Penfolds</td>
<td>Arthurton Rd.</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Lindemans</td>
<td>Smith Ave.</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Orlando</td>
<td>Jones St.</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region ID</th>
<th>Region name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>2</td>
<td>Yarra Valley</td>
<td>Victoria</td>
</tr>
<tr>
<td>3</td>
<td>Margaret River</td>
<td>Western Australia</td>
</tr>
</tbody>
</table>
SQL and SQLite

• Structured Query Language
• a programming language to manage data in a RDBMS
• SQLite implements most, but not all of SQL
  – http://www.sqlite.org/
Aside - Database Admins

• full time jobs
• One example:
• ERCOT = Electric Reliability Council of Texas
SQLite and Android

• Databases created with applications are accessible by name to all classes in application, but no outside applications

• Creating database:
  – create subclass of `SQLiteOpenHelper` and override `onCreate()` method
  – execute SQLite command to create tables in database
  – `onUpgrade()` method for later versions of app and database already present
SQL and Databases

- SQL is a language used to manipulate and manage information in a relational database management system (RDBMS)

- SQL Commands:
  - **CREATE TABLE** - creates a new database table
  - **ALTER TABLE** - alters a database table
  - **DROP TABLE** - deletes a database table
  - **CREATE INDEX** - creates an index (search key)
  - **DROP INDEX** - deletes an index
SQL Commands

• SELECT - get data from a database table
• UPDATE - change data in a database table
• DELETE - remove data from a database table
• INSERT INTO - insert new data in a database table
ANDROID AND SQLITE
Android and SQLite

- SQLite "baked into" Android.
- Device will have SQLite and apps can create and use databases.
- Not necessary to add third party library or jar to your app.
- Many developers use a third party library to ease the syntax burden of using SQLite directly in their code.
Android and SQLite

- SQLiteDatabase class
- methods to programmatically interact with SQLite database
- SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.
- database restricted to application — unless create content provider

Android and SQLite

• Build database on the fly in application
• example (movie ratings) has no built in data to start with
• possible to create database ahead of time and include in apk
• move from apk to Android database on first use

http://stackoverflow.com/questions/5627037/how-can-i-embed-an-sqlite-database-into-an-application
Creating Database

• Example: Movie Rating App
• Stores user ratings
• Not a complex example
• Database only has one table
  – overkill in this scenario
• Adapted from Deitel Address Book Application
Classes

**MovieRaterActivity**
Starting Activity
Displays List of RatedMovies

click on Movie Title
menu - Add Rating

**ViewRating**
Show Rating and Information

**AddEditRating**
Add or Edit Rating

menu - Edit Rating

menu - Delete Rating

**DatabaseConnector**
Interact With Database

Row remove from database
MovieRaterActivity

- ScrollView
- Queries data base for all names / titles
- Clicking on Title brings up that rating in ViewRating
Menu for MovieRaterActivity

- Only one menu option / action bar item
- button to Add Rating
- Brings up AddEditRating Activity
ViewRating

• Pulls all data from database for row based on name / title
• Use of a RatingBar
• ViewRating has its own Action Bar items
ViewRating Menu

• Edit Rating starts AddEditRating activity and populates fields with these values (place in Extras)

• Delete Rating brings up confirmation Dialog

Edit Rating  Delete Rating

MovieRater
AddEditRating

• Add Rating
  – fields are blank

• Consider adding a button for date picker instead of typing data

• Must enter title / name

• other fields can be blank
AddEditRating

• When title clicked in main Activity, MovieRaterActivity
• Make changes and click save
DatabaseConnector Class

• Start of class

```java
public class DatabaseConnector {

    private static final String DATABASE_NAME = "MovieRatings";
    private SQLiteDatabase database;
    private DatabaseOpenHelper databaseOpenHelper;

    public DatabaseConnector(Context context) {
        databaseOpenHelper =
            new DatabaseOpenHelper(context, DATABASE_NAME, null, 1);
    }
}
```
DatabaseConnector Class

```java
public void open() throws SQLException {
    // create or open a database for reading/writing
    database = databaseOpenHelper.getWritableDatabase();
}

public void close() {
    if (database != null)
        database.close();
}
```
Creating Database

- Via an inner class that extends SQLiteOpenHelper
- Used to create database first time app run on a device
- Can also be used to update database if you update your app and alter the structure of the database

```java
private class DatabaseOpenHelper extends SQLiteOpenHelper {

    public DatabaseOpenHelper(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
}
```
Creating Database

- The money method in DatabaseOpenHelper

```java
// creates the ratings table when the database is created
@Override
public void onCreate(SQLiteDatabase db) {
    // query to create a new table named ratings
    String createQuery = "CREATE TABLE ratings" +
                         "(_id INTEGER PRIMARY KEY autoincrement, " +
                         "name TEXT, " +
                         "genre TEXT, " +
                         "dateSeen TEXT, " +
                         "tag1 TEXT, " +
                         "tag2 TEXT, " +
                         "rating INTEGER);";

    db.execSQL(createQuery);
}
```
Creating Database

• String is a SQLite command
• ratings is name of table
• table has seven columns
  – _id, name, genre, dateSeen, tag1, tag2, rating
• storage classes for columns:
  – TEXT, INTEGER, REAL
  – also NULL and BLOB (Binary Large OBject)
• _id is used as primary key for rows
Updating Database

• Quite likely you change the set up of your database over time
  – add tables, add columns, remove tables or columns, reorganize
  – referred to as the *schema* of the database

• `onUpgrade` method for class that extends `SQLiteOpenHelper`
  – for converting database on device (from previous version of your app) to scheme used by newer version of app
  – not trivial!
Aside - Contract Class

• If you plan to use the database in multiple activities and components of your app
  – consider creating a *Contract Class*

• A class with constants that define table names and columns
  – instead of hard coding in multiple places

• Recall the built in ContactsContract and CalendarContract classes
Databases on Device

- can pull database and view
- data/data/app package/database
- sqlitebrowser is a decent tool
sqlite browser

• Entire Database:
  – Recall, we created a single table
sqlite browser

- ratings table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>genre</th>
<th>dateSeen</th>
<th>tag1</th>
<th>tag2</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Star Wars</td>
<td>SCI FI</td>
<td>06/20/1976</td>
<td>Great</td>
<td>Trilogy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Despicable Me 2</td>
<td>Comedy</td>
<td>8/10/2013</td>
<td>Minions</td>
<td>Computer Animation</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>The Emperor’s New</td>
<td>Comedy</td>
<td>4/23/1999</td>
<td>Animation</td>
<td>Disney</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Saving Private Ryan</td>
<td>War</td>
<td>7/27/1998</td>
<td>Gory</td>
<td>Tom Hanks</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>The Magnificent Seven</td>
<td>Western</td>
<td>12/12/1976</td>
<td>Classic</td>
<td>Adaptation</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Wall-E</td>
<td>Animation</td>
<td>6/23/2009</td>
<td>Pixar</td>
<td>Robots</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>Aliens</td>
<td>Science Fiction</td>
<td>6/23/1986</td>
<td>Sequel</td>
<td>Thriller</td>
<td>9</td>
</tr>
</tbody>
</table>
sqlite Manager for Firefox

• Alternative to sqlite Viewer
Inserting Data

- **ContentValues**: object with key/value pairs that are used when inserting/updating databases
- Each **ContentValue** object corresponds to one row in a table
- `_id` being added and incremented automatically
Inserting Data

- In AddEditRating
- When save button clicked

```java
private void saveRating() {
    // get DatabaseConnector to interact with the SQLite da
    DatabaseConnector databaseConnector = new DatabaseConne

    if (getIntent().getExtras() == null) {
        // insert the rating information into the database
        databaseConnector.insertRating(
            title.getText().toString(),
            (int) rating.getRating(),
            genre.getText().toString(),
            dateSeen.getText().toString(),
            tag1.getText().toString(),
            tag2.getText().toString);
    }
    else {
        // ...
    }
```
Inserting Data

• In DatabaseConnector

```java
// inserts a new rating into the database
public void insertRating(String title, int rating,
            String genre, String dateSeen, String tag1, String tag2) {
    ContentValues newRating = new ContentValues();
    newRating.put("name", title);
    newRating.put("rating", rating);
    newRating.put("genre", genre);
    newRating.put("dateSeen", dateSeen);
    newRating.put("tag1", tag1);
    newRating.put("tag2", tag2);

    open();
    database.insert("ratings", null, newRating);
    close();
}
```

nullColumnHack, for inserting empty row
More on insert

• The second parameter
• nullColumnHack
  – that's the parameter identifier
• "optional; may be null. SQL doesn't allow inserting a completely empty row without naming at least one column name. If your provided values (second parameter) is empty, no column names are known and an empty row can't be inserted. If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your values is empty."

http://tinyurl.com/kpl3ow7
Updating Data

• In AddEditRating
• When save button clicked
• notice id added

```java
else {
    databaseConnector.updateRating(rowID,
    title.getText().toString(),
    (int) rating.getRating(),
    genre.getText().toString(),
    dateSeen.getText().toString(),
    tag1.getText().toString(),
    tag2.getText().toString());
}
```
Updating Data

• In DatabaseConnector

```java
// updates a rating in the database
public void updateRating(long id, String name, int rating,
                        String genre, String dateSeen, String tag1, String tag2) {

    ContentValues editRating = new ContentValues();
    editRating.put("name", name);
    editRating.put("rating", rating);
    editRating.put("genre", genre);
    editRating.put("dateSeen", dateSeen);
    editRating.put("tag1", tag1);
    editRating.put("tag2", tag2);

    open();
    database.update("ratings", editRating, "_id=" + id, null);
    close();
}
```
Query Data

• Getting a single row by _id
  – in order to populate ViewRating

```java
// get a Cursor containing all information about the movie specified
// by the given id
public Cursor getOneRating(long id) {
    return database.query(
        "ratings", null, "_id" + id, null, null, null, null, null);

    // public Cursor query (String table, String[] columns,
    // String selection, String[] selectionArgs, String groupBy,
    // String having, String orderBy

```
Query Data

• Get all rows
• To populate the ListView in the MovieRaterActivity
• only getting _id and name columns

```java
public Cursor getAllRatings() {
    return database.query("ratings", new String[] {"_id", "name"},
                        null, null, null, null, "name");
    // query(String table,
    // String[] columns, String selection, String[] selectionArgs,
    // String groupBy, String having, String orderBy)
```
Cursors

• When you execute a query on a database in Android ...

• you get a Cursor back

• "Cursor provided random [access] read-write access to the result of a query"

• Commonly used in other database implementations / models
Cursor

• find out number of rows in result with getCount()

• iterate over rows
  – moveToFirst(), moveToNext()

• determine column names with getColumnNames()

• get values for current row
Cursor

• To use all the data ...
• wrap the Cursor in a SimpleCursorAdapter
• pass the Adapter to a ListView or other view to handle lots of data
• NOTE: result must contain an integer column named _ID that is unique for the result set
  – used as id for row in ListView
• Recall:

```java
public Cursor getAllRatings() {
    return database.query("ratings", new String[] {"_id", "name"},
                          null, null, null, null, "name");
    // query(String table,
    // String[] columns, String selection, String[] selectionArgs,
    // String groupBy, String having, String orderBy)
```
MovieRaterActivity

- Rating Adapter is a CursorAdapter from onCreate method

```java
// map each ratings's name to a TextView
// in the ListView layout
String[] from = new String[] { "name" };
int[] to = new int[] { R.id.ratingTextView }; 
ratingAdapter = new SimpleCursorAdapter(
    MovieRaterActivity.this,
    R.layout.rating_list_item, null,
    from, to);
// public SimpleCursorAdapter (Context context,
// int layout, Cursor c,
// String[] from, int[] to)

setListAdapter(ratingAdapter);
```
Updating Cursor

• Cursor initially null
• separate task to create cursor and update adapter

```java
@Override
protected void onResume() {
    super.onResume();

    // create new GetRatingsTask and execute it
    new GetRatingsTask().execute((Object[]) null);
}
```
Async Task Again

```java
// performs database query outside GUI thread
private class GetRatingsTask extends AsyncTask<Object, Object, Object>
    DatabaseConnector databaseConnector = new DatabaseConnector(MovieRaterActivity.this);

// perform the database access
@Override
protected Cursor doInBackground(Object... params) {
    databaseConnector.open();

    return databaseConnector.getAllRatings();
}

// use the Cursor returned from the doInBackground method
@Override
protected void onPostExecute(Cursor result) {
    ratingAdapter.changeCursor(result);
    databaseConnector.close();
}
} // end class GetContactsTask
```
Clicking on Item in List

• _id not displayed but still part of entry in list -> use _id to get back to database row

```java
// event listener that responds to the user touching a contact's name
// in the ListView
OnClickItemClickListener viewRatingListener = new OnItemClickListener() {
    @Override
    public void onClick(AdapterView<?> parent, View view, int position,
                        long id) {

        Log.d("MoiveRater", "position: " + position + ", id: " + id);
// create an Intent to launch the ViewRating Activity
        Intent viewContact =
            new Intent(MovieRaterActivity.this, ViewRating.class);

// pass the selected contact's row ID as an extra with the Intent
        viewContact.putExtra("ROW_ID", id);
        startActivityForResult(viewContact);
    }

};
```
Deleting Data

• Menu Option in ViewRating

```java
// delete the rating specified by the given id
public void deleteRating(long id) {
    open();
    database.delete("ratings", "_id=" + id, null);
    close();
}
```
Other Cursor Options

• moveToPrevious
• getCount
• getColumnIndexOrThrow
• getColumnName
• getColumnNames
• moveToPosition
• getPosition
Possible Upgrades

• Add functionality to
  – show all movies that share a particular genre
  – movies from a date range
  – shared tags
  – table for the genres (predefined)

• Simply more complex data base queries
ALTERNATIVES TO SQLITE
MOVING HIGHER UP THE FOOD CHAIN
Alternatives to sqlite

• When using SQLite you may feel like you are "Down in the weeds"

• Various alternatives to work higher up the food chain
  – in other words at a higher level of abstraction

• Object Relational Mappers - ORM

• Higher level wrappers for dealing with sql commands and sqlite databases

• Many ORMs exist
ORM Example - Sugar ORM

• Syntactic Sugar?
  – what does that mean?
• Install package
• Add to manifest file
• Classes you want stored in database must extend SugarRecord
Example ORM - Sugar ORM

```java
public class Book extends SugarRecord<Book> {
    String title;
    String edition;

    public Book(Context ctx){
        super(ctx);
    }

    public Book(Context ctx, String title, String edition){
        super(ctx);
        this.title = title;
        this.edition = edition;
    }
}
```
Example ORM - Sugar ORM

• CRUD operations
  – create, read, update, destroy
  – working with the data

Save Entity:

```java
Book book = new Book(ctx, "Title here", "2nd edition")
book.save();
```

Load Entity:

```java
Book book = Book.findById(Book.class, 1);
```

http://satyan.github.io/sugar/getting-started.html
Example ORM - Sugar ORM

Update Entity:

```java
Book book = Book.findById(Book.class, 1);
book.title = "updated title here";  // modify the values
book.edition = "3rd edition";
book.save();  // updates the previous entry with new values.
```

Delete Entity:

```java
Book book = Book.findById(Book.class, 1);
book.delete();
```

Bulk Operations:

```java
List<Book> books = Book.listAll(Book.class);

Book.deleteAll(Book.class);
```
Implications for Movie Rater

• Simple syntax and method calls to make queries on the database
• In the demo app, Movie Rating should be its own class
• Could use Sugar ORM to simplify dealing with the sqlite database