SQLite

C.-Z. Yang
http://syslab.cse.yzu.edu.tw/~czyang

9.4 Android DB with SQLite

- Many Android applications require the storage of complex data structured as a relational database.
- SQLite provides the foundation for managing private and embedded databases in an Android application.
- The Android SDK includes the SQLite software library that implements the SQL (Structured Query Language) database engine.
  - The Android SDK also includes an SQLite database tool for explicit database debugging purposes

SQLite

- As a condensed version of SQL (Structured Query Language), SQLite supports the standard SQL syntax and database transactions.
- Though SQLite is not a full-featured database, it supports a large set of the SQL standard and is sufficient for Android developers needing a simple database engine to plug into their applications.
- https://sqlite.org/

SQLite Databases

- SQLite is a complete relational database engine.
  - A database is implemented as a largely self-contained file.
  - It requires minimal support from external libraries or from the Android OS.
- SQLite differs from conventional databased systems in that it does not require a server.
  - SQLite is imported into an application as a library.
  - All database operations are handled by the application used methods provided by the SQLite library.

Database table for students

- Table: students

TABLE 9-1 Students' Database Table Schema

Table: students		
Column Names	**Data Type**	**Key**
_id	INTEGER	PRIMARY KEY AUTOINCREMENT
name	TEXT	UNIQUE NOT NULL
gender	TEXT	
year_born	INTEGER	NOT NULL
gpa	REAL	

Create the database table

- SQL for creating the students table

```
CREATE TABLE students(
                        _id         INTEGER PRIMARY KEY,
                        name        TEXT UNIQUE NOT NULL,
                        gender      TEXT,
                        year_born   INTEGER,
                        gpa         REAL
)
```

SQLite statements

- Two categories:
  - DDL: data definition language
  - DML: data manipulation language
- DDL: used to build and modify the structure of a database table
- DML: used to query and update data

Insert new rows and query

- SQL INSERT

INSERT INTO students VALUES(1, 'Bill Jones', 'M', 1999, 3.4);
INSERT INTO students VALUES(2, 'John Chavez', 'M', 2001, 3.7);
INSERT INTO students VALUES(3, 'Carol Wan', 'F', 2002, 3.3);
INSERT INTO students VALUES(4, 'Liz Til', 'F', 1999, 3.5);
INSERT INTO students VALUES(5, 'Bon Bon', 'M', 2000, 3.6);
INSERT INTO students VALUES(6, 'Frank Seep', 'M', 2002, 4.0);

INSERT INTO students (name, gender, year, gpa) VALUES('Elise Jack', 'F', 2000, 3.7);

- SQL SELECT

SELECT * FROM students;

SQL query

- SELECT * FROM students;

_id	Name	Gender	Year	GPA
1	Bill Jones	M	1999	3.4
2	John Chavez	M	2001	3.7
3	Carol Wan	F	2002	3.3
4	Liz Til	F	2001	3.5
5	Bon Bon	M	2000	3.6
6	Frank Seep	M	2002	4.0
7	Elise Jack	F	2000	3.7

SQL query

- SELECT name, gender FROM students;

Name	Gender
Bill Jones	M
John Chavez	M
Carol Wan	F
Liz Til	F
Bon Bon	M
Frank Seep	M
Elise Jack	F

SQL query

- SELECT name, gender FROM students WHERE gender = ‘F’ ;

Name	Gender
Carol Wan	F
Liz Til	F
Elise Jack	F

9.5 SQLiteOpenHelper

- The Android SDK provides a set of classes for working with SQLite databases.
- SQLiteOpenHelper is essential for the creation and management of database content, as well as database versioning.
- In an Android SQLite application, SQLiteOpenHelper must be subclassed.

SQLiteOpenHelper

- It must contain the implementation of onCreate() and onUpgrade().
  - onCreate(): to assume the responsibility for creating the database and opening it, or just opening it if it already exists
    - It is called when the database is created for the first time
  - onUpgrade(): to perform an upgrade of the database if necessary

onCreate()

- The schema for myTable is built with seven attributes

```
public void onCreate(SQLiteDatabase database) {
    String instruction = "CREATE TABLE " + " myTable " + "("
            + KEY_ID + " INTEGER PRIMARY KEY, "
            + KEY_COLUMN1+ " REAL, "
            + KEY_COLUMN2+ " TEXT, "
            + KEY_COLUMN3+ " REAL, "
            + KEY_COLUMN4+ " TEXT, "
            + KEY_COLUMN5+ " TEXT, "
            + KEY_COLUMN6+ " INTEGER" + ")";

   database.execSQL (instruction);
 }
```

Access to SQLiteDatabase

- Two methods
  - getReadableDatabase()
  - getWritableDatabase()

```
public Cursor fetchAll(SQLiteDatabase db, String query) {
   Cursor cursor;
   db = this.getReadableDatabase();
   if (db != null) {
     cursor = db.rawQuery(query, null);
   }
   return cursor;
}
```

Insert data

- The common method is to construct the content of a table record using a ContentValues object.

```
public void addRecord(DB_record record) {
   SQLiteDatabase db = this.getWritableDatabase();
   ContentValues values = new ContentValues();
   // Add Key_Value pairs
   values.put(KEY_ID, record.id());
   values.put(KEY_COLUMN1, record.getColumn1());
   values.put(KEY_COLUMN2, record.getColumn2());
   values.put(KEY_COLUMN3, record.getColumn3());
   values.put(KEY_COLUMN4, record.getColumn4());
   values.put(KEY_COLUMN5, record.getColumn5());
   values.put(KEY_COLUMN6, record.getColumn6());

   db.insert(DATABSE_TABLE, null, values);
   db.close();
}
```

db.insert()

- The second argument is the nullColumnHack value.
  - SQL doesn't allow inserting a completely empty row without naming at least one column name.
  - If your provided values 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.

Insert data

- Another approach is to construct an SQLite INSERT statement and then execute it.

```
public void addREcord(DB_Record record){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValue values = new ContentValues();

        String insertStmt = "INSERT or replace INTO " + "TABLE+NAME +  "("
        + Key_COLUMN1 + ", " + Key_COLUMN2, + ", "
        + Key_COLUMN3, + ", " + Key_COLUMN4, + ", "
        + Key_COLUMN5, + ", " + Key_COLUMN6, + ") "  + "VALUES (‘ "
        + record.getColumn1() + " ’, " ++ record.getColumn2() + " ’, "
        + record.getColumn3() + " ’, " ++ record.getColumn4() + " ’, "
        + record.getColumn5() + " ’, " ++ record.getColumn6() + " ’)";

        db.execSQL(insertStm);
        db.close();
}
```

Database Query

- rawQuery()
  - A Cursor object can provide read-write access to the result set of records.

```
SQLiteDatabase database = this.getReadableDatabase();

Cursor cursor = databae.rawQuery("SELECT * FROM “ + DATABASE_TABLE, null);

// COLLECT EACH ROW IN THE TABLE
If (cursor.moveToFirst()){
        do {
                // data processing
        } while (cursor.moveToNext() );
}
```

Database Query

```
public Cursor query (
        String table,
        String[] columns,
        String selection,
        String[] selectionArgs,
        String groupBy,
        String having,
        String orderBy,
        String limit)
```

- query()

```
SQLiteDatabase database = this.getReadableDatabase();

Cursor cursor = databae.query(DATABASE_TABLE, new String[] {
        KEY_COLUMN1,
        KEY_COLUMN2,
        KEY_COLUMN3,
        KEY_COLUMN4,
        KEY_COLUMN5},
        KEY_COLUMN1 + "=?",
        new String[] {String.valueOf(value)},null,null,null,null);

// COLLECT EACH ROW IN THE TABLE
If (cursor.moveToFirst()){
        do {
                // data processing
        } while (cursor.moveToNext() );
}
```

Selection args

query()

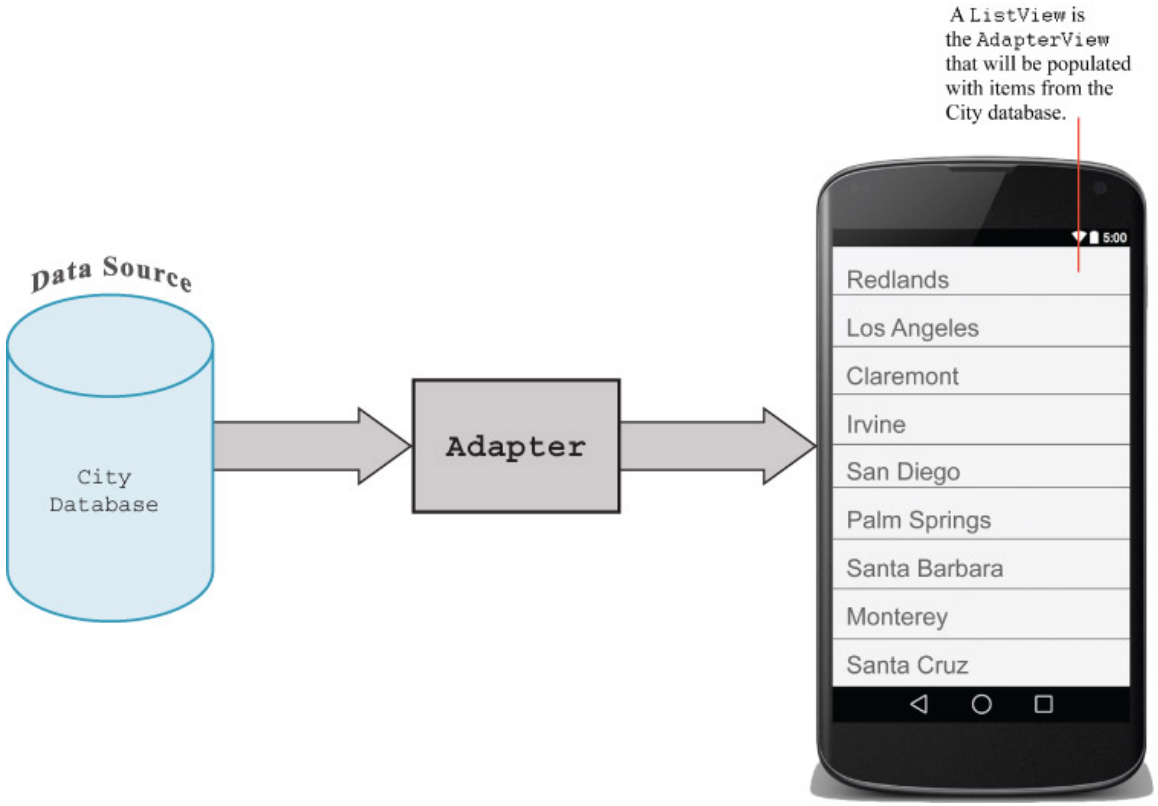
- The query() method takes the following parameters:
  - [String]:The name of the table to compile the query against
  - [String Array]: List of specific column names to return (use null for all)
  - [String] The WHERE clause: Use null for all; might include selection args as ?'s
  - [String Array]: Any selection argument values to substitute in for the ?'s in the earlier parameter
  - [String] GROUP BY clause: null for no grouping
  - [String] HAVING clause: null unless GROUP BY clause requires one
  - [String] ORDER BY clause: If null , default ordering used
  - [String] LIMIT clause: If null , no limit

9.6 Adapters and AdapterViews

- An Adapter object is a mechanism that binds a data source, such as a database, to an AdapterView.
- An AdapterView is a container widget that is populated by a data source, determined by an Adapter.
- Common AdapterViews are ListViews, GridViews, and Spinners.

Adapter

- An Adapter object binds a data source to an AdapterView

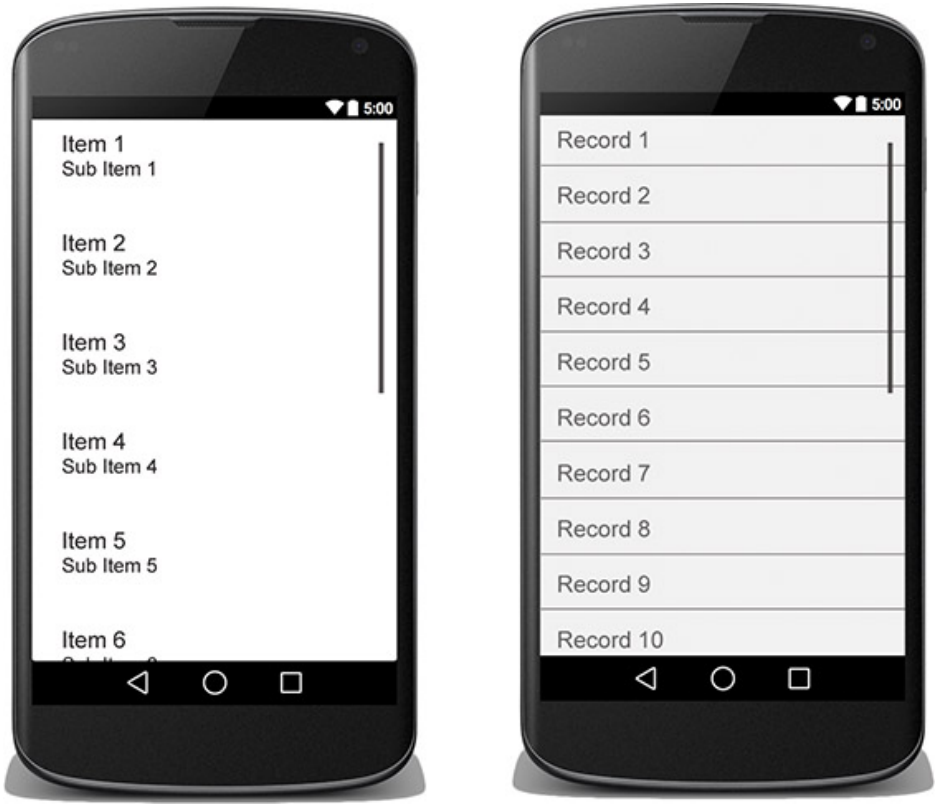


AdapterView

- A ListView is an AdapterView that provides a simple approach for an application to display a scrolling list of records.
- These records can be displayed with a default format, using a built-in style, or customized extensively .
- As an AdapterView, a ListView object requires an Adapter to provide it with data.
- A ListView consists of a collection of sequential items.

ListView

- A List can be a default format or a custom-built layout



ListView

- An AdapterView is populated from data

Lab example 9-1: ToDo Today app

- This is a database experiment that will evolve into a full application in Lab example 9-2.
- This app allows users to build and manage a list of tasks that need to be complete.
- In this lab, users can create a short-term task list: tasks that can be completed in a single day.

The todo list

- To-do tasks that can be completed in a single day

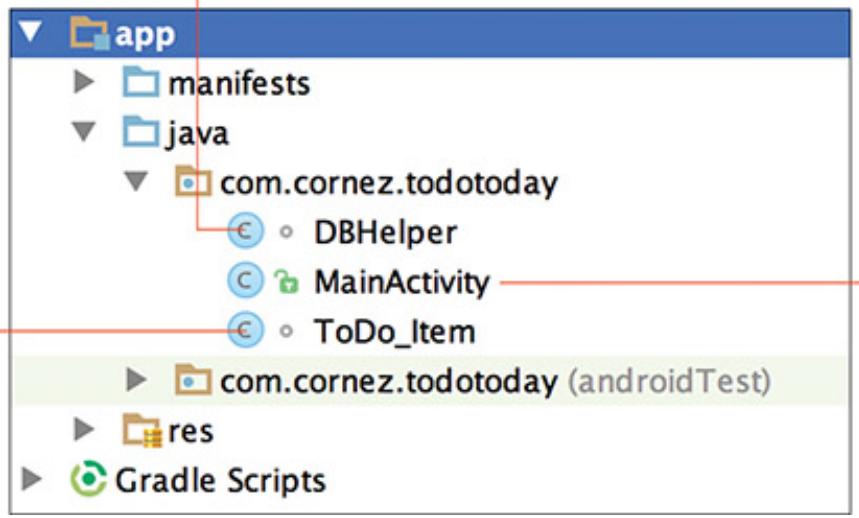
Database

- DB: toDo_Today
- Table: toDo_Items
- Schema:

Column Names	Data Type	Key
_id	INTEGER	Primary Key
description	TEXT	
is_done	INTEGER	

App structure

1. The defined structure of the table row.
2. A helper class for creating and managing the database.
3. The test file for experimenting with the creation and manipulation of the database



ToDo_Item class

```
class ToDo_Item {

   //MEMBER ATTRIBUTES
   private int _id;
   private String description;
   private int is_done;

   public ToDo_Item() {
   }

   public ToDo_Item(int id, String desc, int done) {
      _id = id;
      description = desc;
      is_done = done;
   }

   public int getId() {
      return _id;
   }
   public void setId(int id) {
      _id = id;
   }
   public String getDescription () {
      return description;
   }
   public void setDescription (String desc) {
      description = desc;
   }

   public int getIs_done() {
      return is_done;
   }

   public void setIs_done(int done) {
      is_done = done;
   }

}
```

DBHelper class

```
class DBHelper extends SQLiteOpenHelper {

   //TASK 1: DEFINE THE DATABASE AND TABLE
   private static final int DATABASE_VERSION = 1;
   private static final String DATABASE_NAME = "toDo_Today";
   private static final String DATABASE_TABLE = "toDo_Items";


   //TASK 2: DEFINE THE COLUMN NAMES FOR THE TABLE
   private static final String KEY_TASK_ID = "_id";
   private static final String KEY_DESCRIPTION = "description";
   private static final String KEY_IS_DONE = "is_done";

   private int taskCount;

   public DBHelper (Context context){
      super (context, DATABASE_NAME, null, DATABASE_VERSION);
   }
```

DBHelper class

```
@Override
  public void onCreate (SQLiteDatabase database){
     String table = "CREATE TABLE " + DATABASE_TABLE + "("
           + KEY_TASK_ID + " INTEGER PRIMARY KEY, "
           + KEY_DESCRIPTION + " TEXT, "
           + KEY_IS_DONE + " INTEGER" + ")";
     database.execSQL (table);
     taskCount = 0;
  }

@Override
  public void onUpgrade(SQLiteDatabase database,
                int oldVersion,
                int newVersion) {
     database.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
     onCreate(database);
  }
```

- onUpgrade called when schema is changed such as adding tables, removing tables, changing column data types, etc.
- This method is invoked when the DB version number specified in the constructor changes.
- When a db is altered a new version number must be supplied to the constructor of the class. This is passed in as newVersion.

Add a ToDo task

```
public void addToDoItem(ToDo_Item task) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    taskCount++;

    //ADD KEY-VALUE PAIR INFORMATION FOR THE TASK DESCRIPTION
    values.put(KEY_TASK_ID, taskCount);

    //ADD KEY-VALUE PAIR INFORMATION FOR THE TASK DESCRIPTION
    values.put(KEY_DESCRIPTION, task.getDescription()); // task name

    //ADD KEY-VALUE PAIR INFORMATION FOR IS_DONE
    //  0- NOT DONE, 1 - IS DONE
    values.put(KEY_IS_DONE, task.getIs_done());

    // INSERT THE ROW IN THE TABLE
    db.insert(DATABASE_TABLE, null, values);
        db.close();
}
```

Edit a ToDo task

```
public void editTaskItem(ToDo_Item task){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put(KEY_DESCRIPTION, task.getDescription());
    values.put(KEY_IS_DONE, task.getIs_done());

    db.update(DATABASE_TABLE, values, KEY_TASK_ID + " = ?",
        new String[]{
            String.valueOf(task.getId())
        });
    db.close();
 }
```

Return a ToDo task

```
public ToDo_Item getToDo_Task(int id) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.query(
          DATABASE_TABLE,
          new String[]{KEY_TASK_ID, KEY_DESCRIPTION, KEY_IS_DONE},
          KEY_TASK_ID + "=?",   new String[]{String.valueOf(id)},
          null, null, null, null );
    if (cursor != null)
       cursor.moveToFirst();
    ToDo_Item task = new ToDo_Item(
          cursor.getInt(0),  cursor.getString(1), cursor.getInt(2));
    db.close();
    return task;
  }
```

Delete a ToDo task

```
public void deleteTaskItem (ToDo_Item task){
    SQLiteDatabase database = this.getReadableDatabase();

    // DELETE THE TABLE ROW
    database.delete(DATABASE_TABLE, KEY_TASK_ID + " = ?",
        new String[]
            {String.valueOf(task.getId())});
    database.close();
  }

// return number of items in database; we keep track of this in an instance var
 public int getTaskCount() {
    return taskCount;
  }
```

Get all ToDo tasks

```
public ArrayList<ToDo_Item> getAllTaskItems() {
    ArrayList<ToDo_Item> taskList = new ArrayList<ToDo_Item>();
    String queryList = "SELECT * FROM " + DATABASE_TABLE;
    SQLiteDatabase database = this.getReadableDatabase();
    Cursor cursor = database.rawQuery(queryList, null);
    //COLLECT EACH ROW IN THE TABLE
    if (cursor.moveToFirst()){
       do {
          ToDo_Item task = new ToDo_Item();
          task.setId(cursor.getInt(0));
          task.setDescription(cursor.getString(1));
          task.setIs_done(cursor.getInt(2));
          //ADD TO THE QUERY LIST
          taskList.add(task);
       } while (cursor.moveToNext());
    }
    return taskList;
 }
```

MainActivity

- 4 experiments
  - Create the DB structure. Add the following five ToDo task times to DB and display records in the LogCat window

	_id	description	is_done
1	1	Read Hamlet	True
2	2	Study for exam	False
3	3	Call Andy and Sam	True
4	4	Create newsletter	True
5	5	Buy a dog	false

MainActivity

- 4 experiments
  - Experiment 2: Modify the first record of the database. Replace “Read Hamlet” with “Read newspaper.”
  - Experiment 3: Display the second record of the DB in the LogCat window.
  - Experiment 4:Delete the “Buy a dog” record from the DB. Display all records in the LogCat window.

Ex 1: Create the Database

```
public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      //EXPERIMENT 1: CREATE THE DATABASE
      DBHelper database = new DBHelper(this);
      //          ADD FIVE TASK ITEMS TO THE DATABASE
      database.addToDoItem(new ToDo_Item(
            1, "Read Hamlet", 1));
      database.addToDoItem(new ToDo_Item(
            2, "Study for exam", 1));
      database.addToDoItem(new ToDo_Item(
            3, "Call Andy and Sam", 0));
      database.addToDoItem(new ToDo_Item(
            4, "Create newsletter", 1));
      database.addToDoItem(new ToDo_Item(
            5, "Buy a dog", 0));
```

Ex 1: Create the Database

```
//          DISPLAY ALL THE TASK ITEMS IN THE TABLE
String taskItemList = "\n";
ArrayList<ToDo_Item> taskList = database.getAllTaskItems();
for (int i = 0; i < database.getTaskCount(); i++) {
   ToDo_Item task = taskList.get(i);
   taskItemList += "\n" + task.getDescription() + "\t" +
         task.getIs_done();
}
Log.v("DATABASE RECORDS", taskItemList);
```

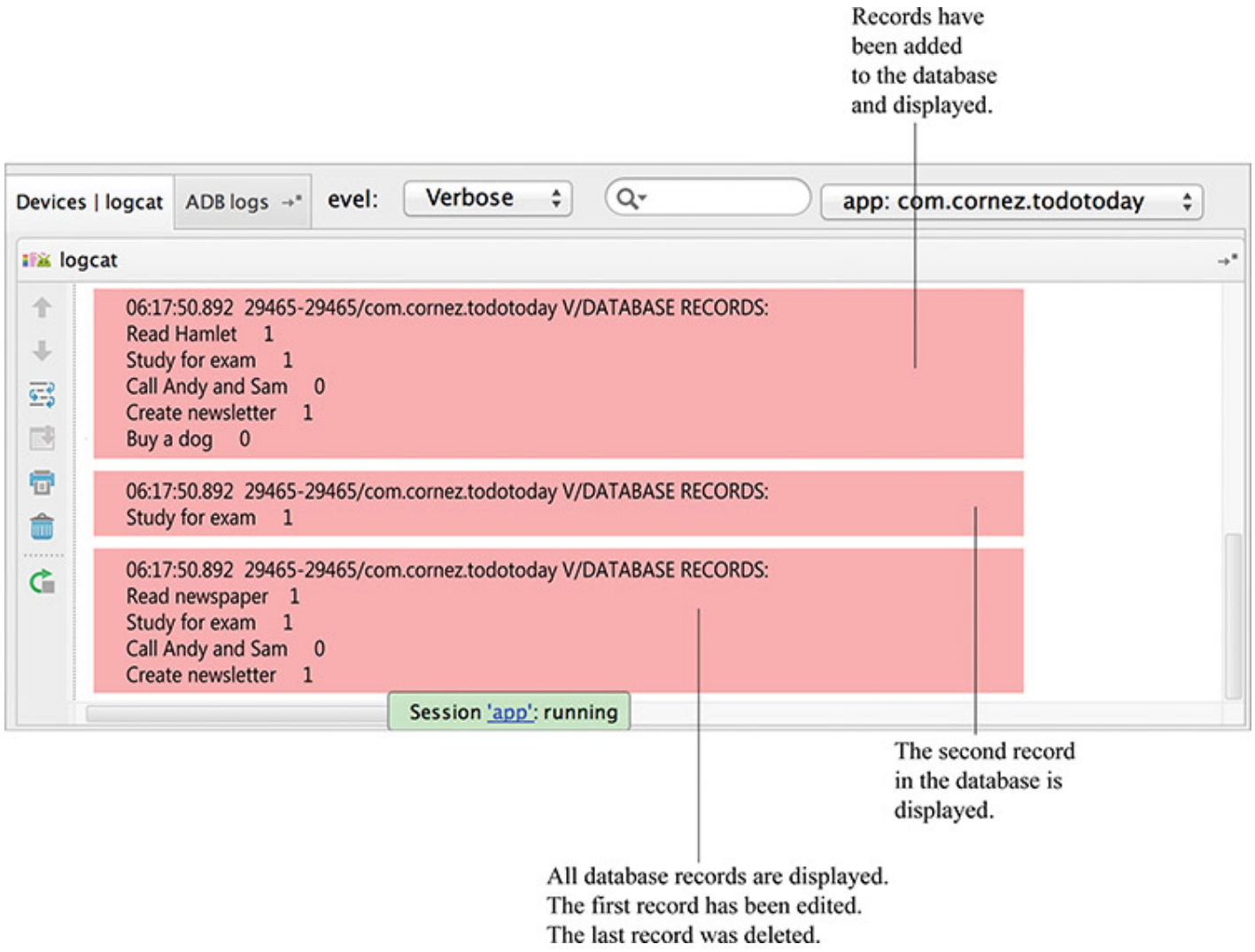
Ex 2+3: Modify/Display a Record

```
// EXPERIMENT 2: MODIFY A RECORD
      database.editTaskItem(new ToDo_Item(
            1, "Read newspaper", 1));

//EXPERIMENT 3: DISPLAY A SPECIFIC RECORD
      ToDo_Item anItem = database.getToDo_Task(2);
      Log.v("DATABASE RECORDS", anItem.getDescription());
```

Ex 4: Delete a record

```
//EXPERIMENT 4: DELETE A RECORD
      database.deleteTaskItem(new ToDo_Item(
            15, "Buy a dog", 0));

      //            DISPLAY ALL THE TASK ITEMS IN THE TABLE
      taskItemList = "\n";
      taskList = database.getAllTaskItems();
      for (int i = 0; i < database.getTaskCount(); i++) {
         ToDo_Item task = taskList.get(i);
         taskItemList += "\n" + task.getDescription() + "\t" +
               task.getIs_done();
      }
      Log.v("DATABASE RECORDS", taskItemList);
   }", anItem.getDescription());
```

Results sets



Concluding Remarks