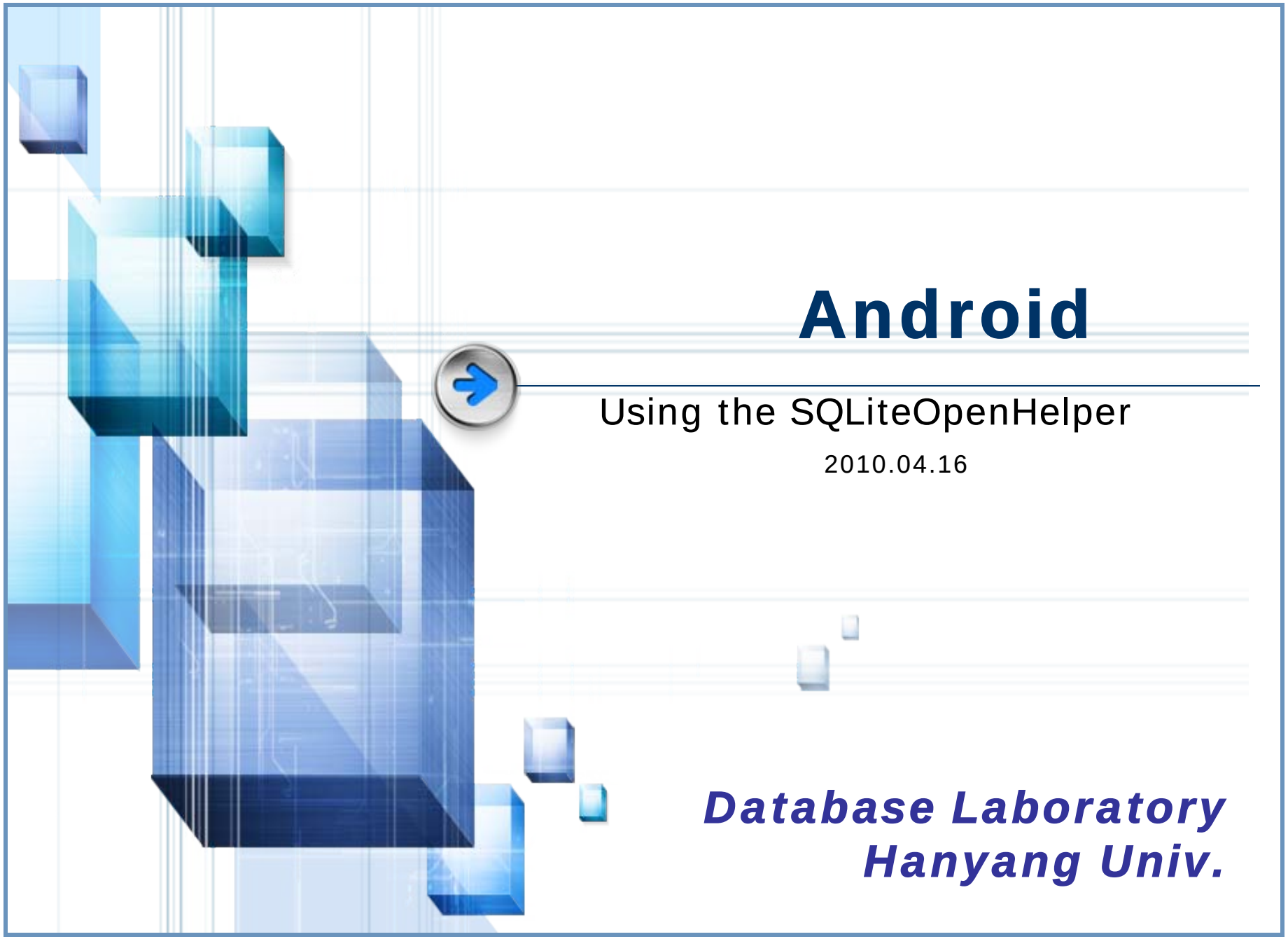

Android

Using the SQLiteOpenHelper

2010.04.16

Database Laboratory
Hanyang Univ.

Saving Your To-Do List

- you enhanced the To-Do List example to persist the Activity's UI state across sessions
- in the following example, you'll create a private database to save the to-do items

1. Start by creating a new ToDoDBAdapter class. It will be used to manage your database interactions
   - Create private variables to store the SQLiteDatabase object and the Context of the calling application
   - Add a constructor that takes the owner application's Context, and include static class variables for the name and version of the database and a name for the to-do item table

Saving Your To-Do List

```
package com.paad.todolist;

import java.util.Date;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.util.Log;
import com.paad.todolist.ToDoItem;

/** Database Adapter for persisting Todo Items */
public class ToDoDBAdapter {
  private static final String DATABASE_NAME = "todoList.db";
  private static final String DATABASE_TABLE = "todoItems";
  private static final int DATABASE_VERSION = 1;

  private SQLiteDatabase db;
  private final Context context;

  public ToDoDBAdapter(Context _context) {
    context = _context;
  }
```

Saving Your To-Do List

2. Create public convenience variables that defi ne the column names and indexes

- this will make it easier to find the correct columns when extracting values from query result Cursors

```
public static final String KEY_ID = "_id";
public static final String KEY_TASK = "task";
public static final String KEY_CREATION_DATE = "creation_date";

public static final int TASK_COLUMN = 1;
public static final int CREATION_DATE_COLUMN = 2;
```

3. Create a new taskDBOpenHelper class within the ToDoDBAdapter that extends SQLiteOpenHelper

- It will be used to simplify version management of your database
- Within it, overwrite the onCreate and onUpgrade methods to handle the database creation and upgrade logic

Saving Your To-Do List

```
private static class toDoDBOpenHelper extends SQLiteOpenHelper {

    public toDoDBOpenHelper(Context context, String name, CursorFactory factory, int version) {
      super(context, name, factory, version);
    }

    /** SQL Statement to create a new database */
    private static final String DATABASE_CREATE = "create table " +
      DATABASE_TABLE + " (" + KEY_ID + " integer primary key autoincrement, " +
      KEY_TASK + " text not null, " + KEY_CREATION_DATE + " integer);";

    @Override
    public void onCreate(SQLiteDatabase _db) {
      _db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
      Log.w("TaskDBAdapter", "Upgrading from version " +
                             _oldVersion + " to " +
                             _newVersion + ", which will destroy all old data");

      // Drop the old table.
      _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
      // Create a new one.
      onCreate(_db);
    }
  }
}
```

Saving Your To-Do List

4. **Within the ToDoDBAdapter class, add a private instance variable to store an instance of the toDoDBOpenHelper class you just created**
   - assign it within the constructor

```
private toDoDBOpenHelper dbHelper;

public ToDoDBAdapter(Context _context) {
  context = _context;
  dbHelper = new toDoDBOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

5. **Still in the adapter class, create open and close methods that encapsulate the open and close logic for your database**
   - Start with a close method that simply calls close on the database object

```
public void close() {
    db.close();
  }
```

Saving Your To-Do List

6. The open method should use the toDoDBOpenHelper class

- Call getWritableDatabase to let the helper handle database creation and version checking
- Wrap the call to try to provide a readable database if a writable instance can't be opened

```
public void open() throws SQLiteException {
    try {
        db = dbHelper.getWritableDatabase();
    }
    catch (SQLiteException ex) {
        db = dbHelper.getReadableDatabase();
    }
  }
```

Saving Your To-Do List

7. Add strongly typed methods for adding, removing, and updating items

```
public long insertTask(ToDoItem _task) {
  // Create a new row of values to insert.
  ContentValues newTaskValues = new ContentValues();

  // Assign values for each row.
  newTaskValues.put(KEY_TASK, _task.getTask());
  newTaskValues.put(KEY_CREATION_DATE, _task.getCreated().getTime());

  // Insert the row.
  return db.insert(DATABASE_TABLE, null, newTaskValues);
}

/** Remove a task based on its index */
public boolean removeTask(long _rowIndex) {
  return db.delete(DATABASE_TABLE, KEY_ID + "=" + _rowIndex, null) > 0;
}

/** Update a task */
public boolean updateTask(long _rowIndex, String _task) {
  ContentValues newValue = new ContentValues();
  newValue.put(KEY_TASK, _task);
  return db.update(DATABASE_TABLE, newValue, KEY_ID + "=" + _rowIndex, null) > 0;
}
```

Saving Your To-Do List

8. **Now add helper methods to handle queries. Write three methods**
   - one to return all the items, another to return a particular row as a Cursor, and finally, one that returns a strongly typed ToDoItem.

Saving Your To-Do List

```
public Cursor getAllToDoItemsCursor() {
    return db.query(DATABASE_TABLE, new String[] { KEY_ID, KEY_TASK, KEY_CREATION_DATE},
            null, null, null, null, null);
  }

/** Return a Cursor to a specific row */
  public Cursor setCursorToToDoItem(long _rowIndex) throws SQLException {
    Cursor result = db.query(true, DATABASE_TABLE, new String[] {KEY_ID, KEY_TASK},
                             KEY_ID + "=" + _rowIndex, null, null, null, null, null);

    if ((result.getCount() == 0) || !result.moveToFirst()) {
      throw new SQLException("No Todo items found for row: " + _rowIndex);
    }
    return result;
  }

  /** Return a Todo Item based on its row index */
  public ToDoItem getToDoItem(long _rowIndex) throws SQLException {
    Cursor cursor = db.query(true, DATABASE_TABLE, new String[] {KEY_ID, KEY_TASK},
                             KEY_ID + "=" + _rowIndex, null, null, null, null, null);
    if ((cursor.getCount() == 0) || !cursor.moveToFirst()) {
      throw new SQLException("No to do item found for row: " + _rowIndex);
    }

    String task = cursor.getString(TASK_COLUMN);
    long created = cursor.getLong(CREATION_DATE_COLUMN);

    ToDoItem result = new ToDoItem(task, new Date(created));
    return result;
  }
```

Saving Your To-Do List

9. Return the ToDoList Activity, and update it to persist the to-do list array

- Start by updating the Activity's onCreate method to create an instance of the toDoDBAdapter
- open a connection to the database. Also include a call to the populateTodoList method stub

```
private ToDoDBAdapter toDoDBAdapter;

@Override
public void onCreate(Bundle icicle) {

* [... existing onCreate logic ... ] */

  toDoDBAdapter = new ToDoDBAdapter(this);
  // Open or create the database
  toDoDBAdapter.open();

  populateTodoList();
}

private void populateTodoList() { }
```

Saving Your To-Do List

10. Create a new instance variable to store a Cursor over all the to-do items in the database

- Update the populateTodoList method to use the toDoDBAdapter instance to query the database
- call startManagingCursor to let the Activity manage the Cursor
- It should also make a call to updateArray, a method that will be used to repopulate the to-do list array using the Cursor

```
private void populateTodoList() {
  // Get all the todo list items from the database.
  toDoListCursor = toDoDBAdapter. getAllToDoItemsCursor();
  startManagingCursor(toDoListCursor);

  // Update the array.
  updateArray();
}

private void updateArray() { }
```

Saving Your To-Do List

11. Now implement the updateArray method to update the current to-do list array

- Call requery on the result Cursor to ensure that it's fully up to date, then clear the array and iterate over the result set
- When complete, call notifyDataSetChanged on the Array Adapter

```
private void updateArray() {
    toDoListCursor.requery();

    todoItems.clear();

    if (toDoListCursor.moveToFirst())
      do {
        String task = toDoListCursor.getString(ToDoDBAdapter.TASK_COLUMN);
        long created = toDoListCursor.getLong(ToDoDBAdapter.CREATION_DATE_COLUMN);

        ToDoItem newItem = new ToDoItem(task, new Date(created));
        todoItems.add(0, newItem);
      } while(toDoListCursor.moveToNext());

    aa.notifyDataSetChanged();
  }
```

Saving Your To-Do List

12. To join the pieces together, modify the OnKeyListener assigned to the text entry box in the onCreate method, and update the removeItem method

- Both should now use the toDoDBAdapter to add and remove items from the database rather than modifying the to-do list array directly

1) Start with the OnKeyListener, insert the new item into the database, and refresh the array.

```
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  // Inflate your view
  setContentView(R.layout.main);

  // Get references to UI widgets
  myListView = (ListView)findViewById(R.id.myListView);
  myEditText = (EditText)findViewById(R.id.myEditText);

  todoItems = new ArrayList<ToDoItem>();
  int resID = R.layout.todolist_item;
  aa = new ToDoItemAdapter(this, resID, todoItems);
  myListView.setAdapter(aa);

  myEditText.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
      if (event.getAction() == KeyEvent.ACTION_DOWN)
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
          ToDoItem newItem = new ToDoItem(myEditText.getText().toString());
          toDoDBAdapter.insertTask(newItem);
          updateArray();
          myEditText.setText("");
          aa.notifyDataSetChanged();
          cancelAdd();
          return true;
        }
      return false;
    }
  });

  registerForContextMenu(myListView);
  restoreUIState();

  toDoDBAdapter = new ToDoDBAdapter(this);
  // Open or create the database
  toDoDBAdapter.open();

  populateTodoList();
}
```

Saving Your To-Do List

2) Then modify the removeItem method to remove the item from the database and refresh the array list

```
private void removeItem(int _index) {
    // Items are added to the listview in reverse order, so invert the index.
    toDoDBAdapter.removeTask(todoItems.size()-_index);
    updateArray();
  }
```

13. As a final step, override the onDestroy method of your Activity to close your database connection.

```
public void onDestroy() {
  // Close the database
  toDoDBAdapter.close();

  super.onDestroy();
}
```

- **Your to-do items will now be saved between sessions**