Application Context

Marco Ronchetti
Università degli Studi di Trento

The Context

An interface to global information about an application environment.

It allows accessing application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

We have seen it in various cases:

- Activity is subclass of Context
- new Intent(Context c, Class c);
- isIntentAvailable(Context context, String action)

A global Application Context

Is there a simple way to maintain and access the application context from everywhere it's needed?

a) Modify the Android Manifest adding the "name" parameter to the application tag

```
<application android:name="myPackage.MyApplication"> …
</application>
```

b) Write the class

```
public class MyApplication extends Application{
    private static Context context;
    public void onCreate(){
        super.onCreate();
        MyApplication.context = getApplicationContext();
    }
    public static Context getAppContext() {
        return MyApplication.context;
    }
}
```

c) Access MyApplication.getAppContext() to get your application context statically from eveywhere.

4

Internal Database

Marco Ronchetti
Università degli Studi di Trento

Why an internal database?

Useful for easy handling of structured data.

The main classes

SQLiteOpenHelper

- responsible for creating, opening, and upgrading a program's database.

SQLiteDatabase

- responsible for communicating changes to the data within the database.

Cursor

- exposes results from a query on a SQLiteDatabase.

ContentValues

- a convenience map to pass values

SQLiteOpenHelper

SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

- context The application context
- name of the db file (null for an in-memory db)
- factory for creating (custom) cursor objects, or null for the default
- version number of the database (starting at 1)

SQLiteOpenHelper - lifecycle

onCreate

onOpen

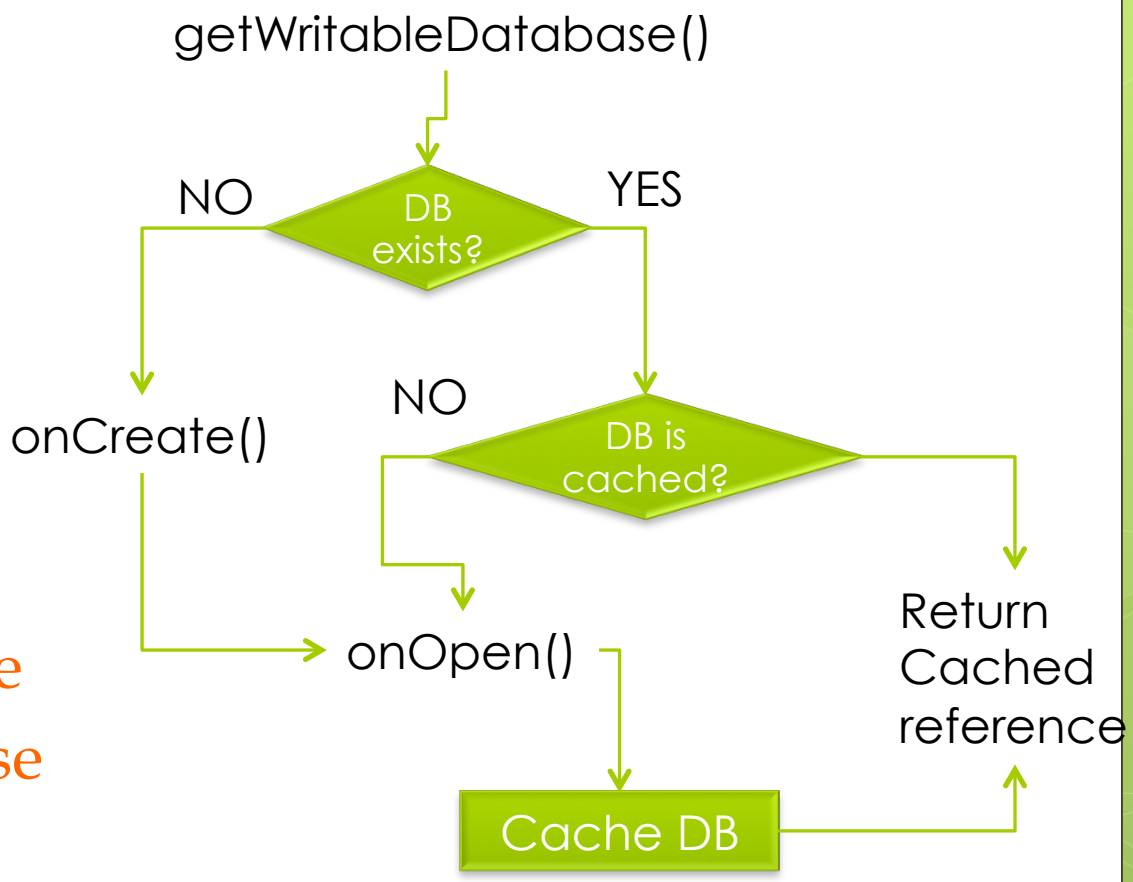
onClose

onUpgrade

onDowngrade

getWritableDatabase

getReadableDatabase



SQLiteOpenHelper

Call close() when the handle to DB is not needed any more (you can reaccess it later).

If the DB is opened for reading and you call getWritableDatabase, it gets closed and reopened.

Utility class: ContentValues

ContentValues (similar to Extras)

- A key-value map. Methods:
  - void put(String s, #TYPE# val);
  - Object get(String s);
  - #TYPE# getAs#TYPE# (String s): getAsByte, getAsByteArray, getAsFloat, getAsInteger, getAsLong, getAsShort, getAsString
  - Set keySet(), Set valueSet()
  - int size(); void clear();

SQLiteDatabase

long insert(String table, String nullColumnHack, ContentValues values)

- table: the table to insert the row into
- nullColumnHack: optional; may be null. Trick to enter an empty row: put in the field the name of the column where a NULL is explicitly inserted.
- values this map contains the initial column values for the row. The keys should be the column names and the values the column values
- Returns the ID

long replace(String table, String nullColumnHack, ContentValues values)

int delete (String table, String whereClause, String[] whereArgs)

Es:

delete("MyTable", "A=?, B<?, C>?", new String[] {"pippo", "2", "6"});

means delete * from MyTable where A="pippo", B<2, C<6;

- returns the number of affected rows

void execSQL (String sql) where sql is an sql query NOT returning values.

SQLiteDatabase

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

- table The table name to compile the query against.
- columns A list of which columns to return. Passing null will return all columns
- selection list of rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
- selectionArgs You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
- groupBy how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
- having which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
- orderBy How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.

SQLiteDatabase

- close()

DB Status

- isOpen()
- isReadOnly()

Transaction support

- beginTransaction()
- endTransaction()
- setTransactionSuccessful()

Utility class: Cursor

provides random read-write access to the result set returned by a database query

Metadata methods:

- int getCount()
  - Returns the numbers of rows in the cursor.
- int getColumnCount()
  - Return total number of columns
- String getColumnName(int columnIndex)
  - Returns the column name at the given zero-based column index.
- String [] getColumnNames()
  - Returns a string array holding the names of all of the columns in the result set in the order in which they were listed in the result.
- int getType(int columnIndex)
  - Returns data type of the given column's value.

FIELD_TYPE_BLOB
FIELD_TYPE_FLOAT
FIELD_TYPE_INTEGER
FIELD_TYPE_NULL
FIELD_TYPE_STRING

Utility class: Cursor

Position check

- boolean isFirst()
- boolean isAfterLast()
- boolean isBeforeFirst()
- boolean isLast()

Position move

- boolean move(int offset)
  - Move the cursor by a relative amount, forward or backward, from the current position.
- boolean moveToPosition(int position)
  - Move the cursor to an absolute position.
- boolean moveToFirst()
  - Move the cursor to the first row.
- boolean moveToLast()
  - Move the cursor to the last row.
- boolean moveToNext()
  - Move the cursor to the next row.
- boolean moveToPrevious()
  - Move the cursor to the previous row.

All the move methods return true
If the move was successful

Utility class: Cursor

void close()

- closes the Cursor, releasing all of its resources and making it completely invalid.

boolean isClosed()

- return true if the cursor is closed

Getter methods

- double getDouble(int columnIndex)
- float getFloat(int columnIndex)
- int getInt(int columnIndex)
- long getLong(int columnIndex)
- short getShort(int columnIndex)
- String getString(int columnIndex)
- byte[] getBlob(int columnIndex)

All the getter methods return the value of the requested column as the specified type

Utility methods in Context

String[] databaseList()

- Returns an array of strings naming the private databases associated with this Context's application package.

boolean deleteDatabase(String name)

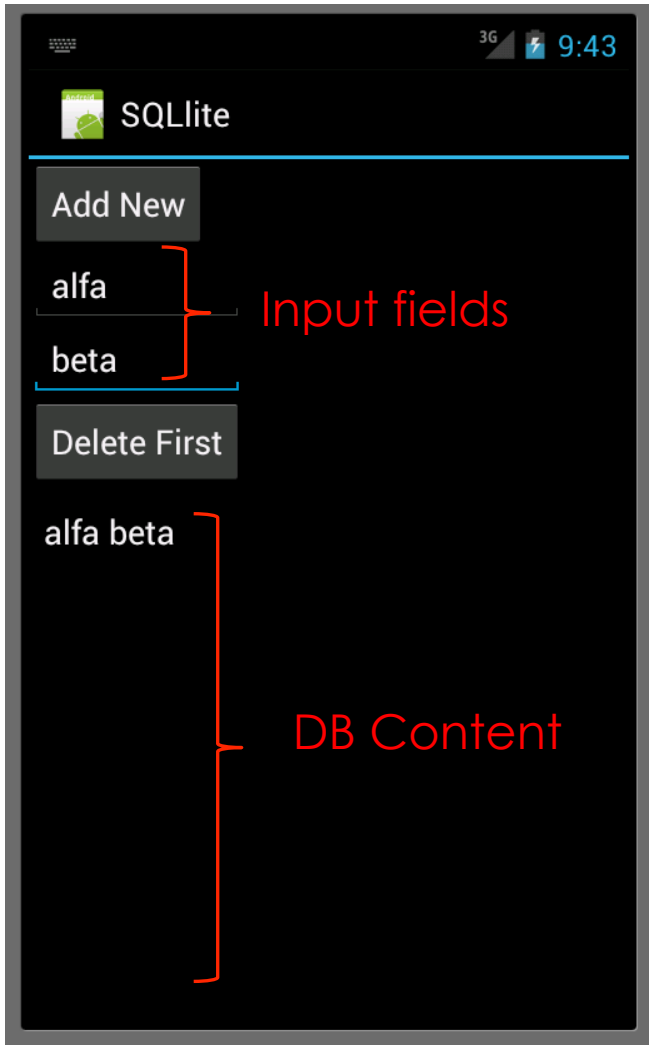
- Delete an existing private SQLiteDatabase associated with this Context's application package.

Internal Database: an example

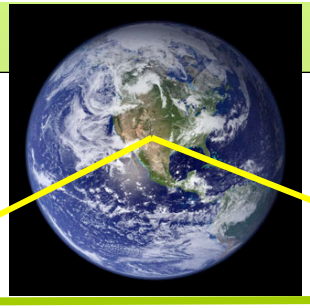
Marco Ronchetti
Università degli Studi di Trento

Derived by Lars Vogel, with modifications
http://www.vogella.de/articles/AndroidSQLite/article.html

Our toy target



ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DB

Actual storage

FS

platforms

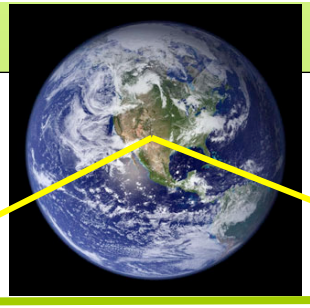
temp

tools

Data

Object

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DB

Actual storage

FS

platforms

temp

tools

Data

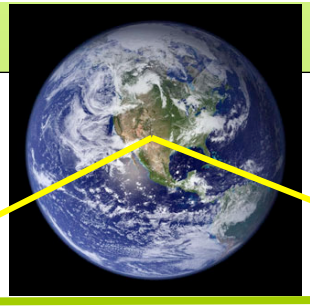
Object

The Object

```
package it.unitn.science.latemar;

public class Person {
        private long id;
        private String name;
        private String surname;
        Person(){}
        Person(String name, String surname){
                this.name=name;
                this.surname=surname;
                this.id=-1; // means: not in DB
        }
        Person(long id, String name, String surname){
                this.name=name;
                this.surname=surname;
                this.id=id; // means: not in DB
        }
        public long getId() { return id; }
        public void setId(long id) { this.id = id;}
        public String getName() { return name; }
        public void setName(String name) {
                this.name = name; }
        public String getSurname() {
                return surname; }
        public void setSurname(String surname) {
                this.surname = surname; }
        @Override
        public String toString() {
                return name+" "+surname;
        }
}
```

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DAO

DB

Actual storage

FS

platforms

temp

tools

Data

Object

The DAO interface

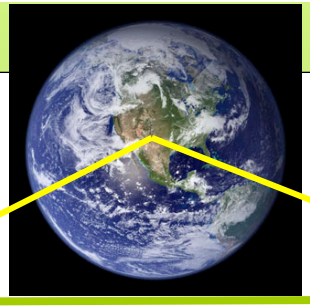
```
package it.unitn.science.latemar;

import java.util.List;

public interface PersonDAO {
        public void open();
        public void close();

        public Person insertPerson(Person person) ;
        public void deletePerson(Person person) ;
        public List<Person> getAllPerson() ;
}
```

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DB

Actual storage

FS

platforms

temp

tools

Data

Object

The DB

```
package it.unitn.science.latemar;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
```

Define constants

```
public class MySQLiteHelper extends SQLiteOpenHelper {
        public static final String TABLE_PEOPLE = "people";
        public static final String COLUMN_ID = "_id";
        public static final String COLUMN_NAME = "name";
        public static final String COLUMN_SURNAME = "surname";
        private static final String DATABASE_NAME = "contacts.db";
        private static final int DATABASE_VERSION = 1;
        // Database creation sql statement
        private static final String DATABASE_CREATE = "create table "
                + TABLE_PEOPLE + "( "
                + COLUMN_ID + " integer primary key autoincrement, "
                + COLUMN_NAME + " text not null,"
                + COLUMN_SURNAME + " text not null);";
        public MySQLiteHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
```

Using default Cursor factory

The DB – part 2

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

@Override
public void onUpgrade(SQLiteDatabase db,
                      int oldVersion, int newVersion) {
    Log.w(MySQLiteHelper.class.getName(),
          "Upgrading database from version " + oldVersion + " to "
          + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_PEOPLE);
    onCreate(db);
}
}
```

The DAO implementation - DB

```
package it.unitn.science.latemar;
import ...
```

```
public class PersonDAO_DB_impl implements PersonDAO {

        private SQLiteDatabase database;
        private MySQLiteHelper dbHelper;
        private String[] allColumns = { MySQLiteHelper.COLUMN_ID,
                        MySQLiteHelper.COLUMN_NAME,
                        MySQLiteHelper.COLUMN_SURNAME};
@Override
        public void open() throws SQLException {
                if (dbHelper==null) dbHelper =
                        new MySQLiteHelper(MyApplication.getAppContext());
                database = dbHelper.getWritableDatabase();
        }
@Override
        public void close() {
                dbHelper.close();
        }
```

Using the code we discussed to access the Global Context

The DAO impl. – utility methods

```
private ContentValues personToValues(Person person) {
        ContentValues values = new ContentValues();
        values.put(MySQLiteHelper.COLUMN_NAME,
                person.getName());
        values.put(MySQLiteHelper.COLUMN_SURNAME,
                person.getSurname());
        return values;
}

private Person cursorToPerson(Cursor cursor) {
        long id = cursor.getLong(0);
        String name=cursor.getString(1);
        String surname=cursor.getString(2);
        return new Person(id,name,surname);
}
```

From Object To DB

From DB To Object

The DAO impl. – data access 1

```
@Override
  public Person insertPerson(Person person) {
        long insertId = database.insert(MySQLiteHelper.TABLE_PEOPLE, null,
                                        personToValues(person));
        // Now read from DB the inserted person and return it
        Cursor cursor = database.query(MySQLiteHelper.TABLE_PEOPLE,
                allColumns, MySQLiteHelper.COLUMN_ID + " = ?" ,
                new String[] {""+insertId}, null, null, null);
        cursor.moveToFirst();
        Person p=cursorToPerson(cursor);
        cursor.close();
        return p;
  }
```

The DAO impl. – data access 2

```
@Override
    public void deletePerson(Person person) {
            long id = person.getId();

            //database.delete(MySQLiteHelper.TABLE_PEOPLE,
            //                MySQLiteHelper.COLUMN_ID + " = " + id,
            //                null);

            database.delete(MySQLiteHelper.TABLE_PEOPLE,
                            MySQLiteHelper.COLUMN_ID + " = ?",
                            new String[] {""+id});
    }
```

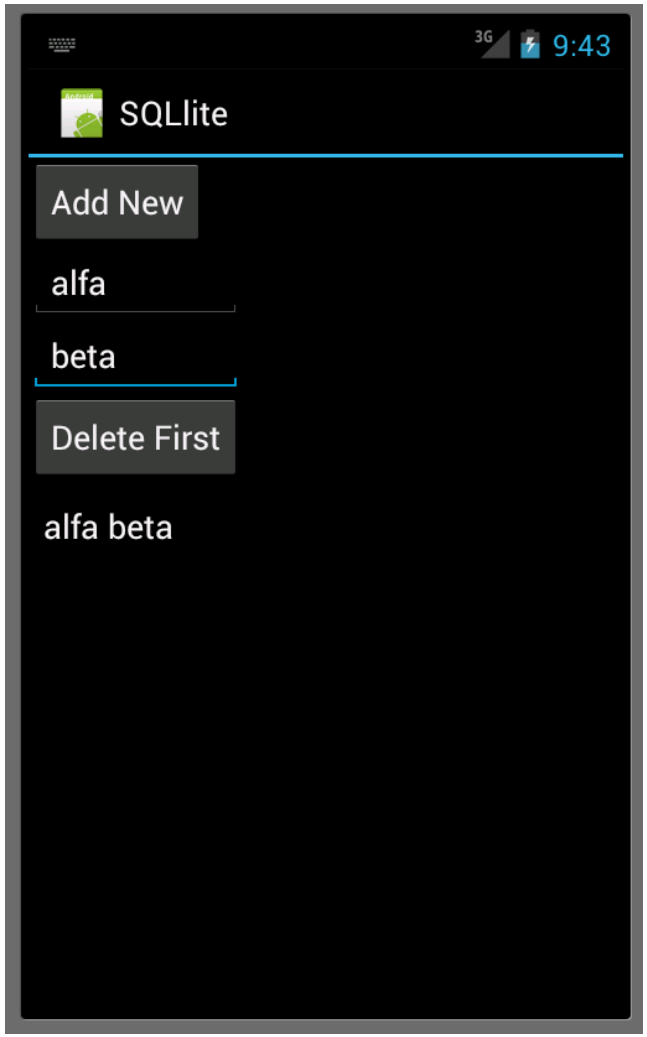
RED version preferred to the BLUE one!

The DAO impl. – data access 3

```
@Override
  public List<Person> getAllPersons() {
        List<Person> people = new ArrayList<Person>();
        Cursor cursor = database.query(MySQLiteHelper.TABLE_PEOPLE,
                        allColumns, null, null, null, null, null);
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
              Person person = cursorToPerson(cursor);
              people.add(person);
              cursor.moveToNext();
        }
        cursor.close();  // Remember to always close the cursor!
        return people;
  }
```

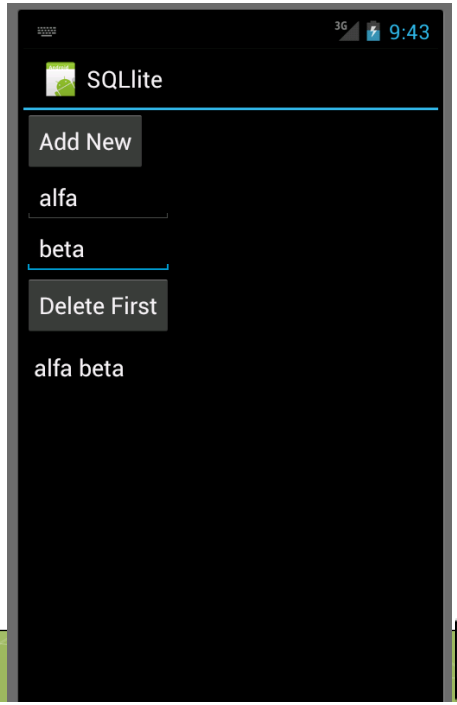
Select * from people

Let us write the activity



Our Activity – main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout … android:orientation="vertical" >
   <LinearLayout android:id="@+id/group" … android:orientation="vertical" >
       <Button android:id="@+id/add" … android:text="Add New"
       android:onClick="onClick"/>
       <EditText android:id="@+id/editText1"…><requestFocus /> </EditText>
       <EditText android:id="@+id/editText2"…></EditText>
       <Button android:id="@+id/delete"… android:text="Delete First"
       android:onClick="onClick"/>
   </LinearLayout>
   <ListView
    android:id="@android:id/list"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
</LinearLayout>
```



Our Activity

```
package it.unitn.science.latemar;
import ...
```

```
public class SQLliteActivity extends ListActivity {
        private PersonDAO dao;
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                dao = new PersonDAO_DB_impl();
                dao.open();
                List<Person> values = dao.getAllPersons();

                // Use the SimpleCursorAdapter to show the
                // elements in a ListView
                ArrayAdapter<Person> adapter = new ArrayAdapter<Person>(this,
                                android.R.layout.simple_list_item_1, values);
                setListAdapter(adapter);
        }
```

Our Activity

```
@Override
  protected void onResume() {
            dao.open();            super.onResume();
  }

@Override
   protected void onPause() {
            dao.close();           super.onPause();
  }

// Will be called via the onClick attribute of the buttons in main.xml
  public void onClick(View view) {
     ArrayAdapter<Person> adapter = (ArrayAdapter<Person>) getListAdapter();
     Person person = null;
      final EditText tf1 = (EditText) findViewById(R.id.editText1);
      final EditText tf2 = (EditText) findViewById(R.id.editText2);
```

Our Activity

```
switch (view.getId()) {
        case R.id.add:
                String name=tf1.getText().toString();
                String surname=tf2.getText().toString();
                person = dao.insertPerson(new Person(name,surname));
                adapter.add(person);  tf1.setText("");  tf2.setText("");
                break;
        case R.id.delete:
                if (getListAdapter().getCount() > 0) {
                        person = (Person) getListAdapter().getItem(0);
                        dao.deletePerson(person);
                        adapter.remove(person);
                }
        break;
   }
  adapter.notifyDataSetChanged();
}// end of method
}// end of class
```

Next time…

we will change persistence implementation, writing on File System instead of Database