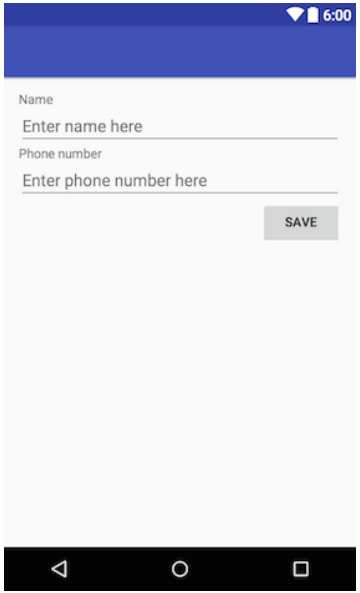
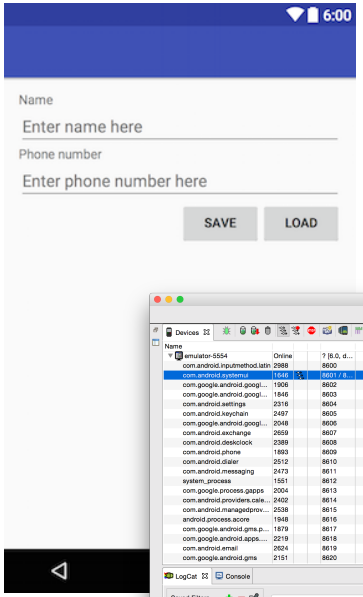
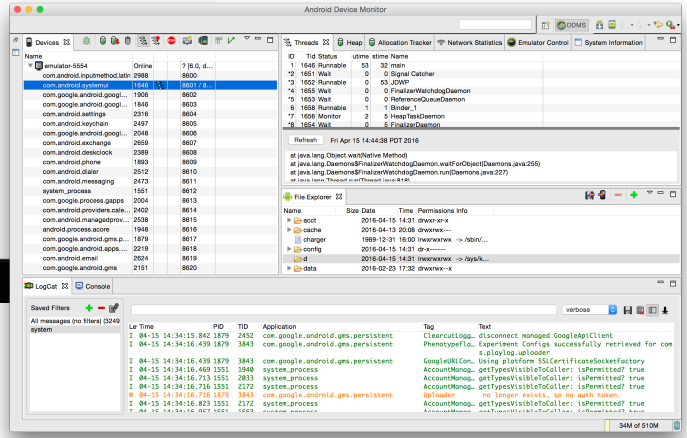
Data Persistence

Lab 1

- Data persistence choice
- SharedPreference
- Writing/reading to files
- External storage and permissions







Choices for data persistence

Saving Key-Value Sets

Learn to use a shared preferences file for storing small amounts of information in key-value pairs.

Saving Files

Learn to save a basic file, such as to store long sequences of data that are generally read in order.

Saving Data in SQL Databases

Learn to use a SQLite database to read and write structured data.

- SharedPreferences are often used for a limited set of data that represent user choices about the way they want the app configured.
- Flat files are useful for backing up data and transmitting to other users.
- Databases are the workhorses for data manipulation, storage, and retrieval.

SHAREDPREFERENCES

SharedPreferences

Note: The `SharedPreferences` APIs are only for reading and writing key-value pairs and you should not confuse them with the `Preference` APIs, which help you build a user interface for your app settings (although they use `SharedPreferences` as their implementation to save the app settings). For information about using the `Preference` APIs, see the Settings guide.

YO

•

SharedPreferences

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

- `getSharedPreferences()` — Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any `Context` in your app.
- `getPreferences()` — Use this from an `Activity` if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name.

SharedPreferences

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

SharedPreferences example

```
TextView name ;
TextView phone;
TextView email;
TextView street;
TextView place;
public static final String MyPREFERENCES = "MyPrefs" ;
public static final String Name = "nameKey";
public static final String Phone = "phoneKey";
public static final String Email = "emailKey";
public static final String Street = "streetKey";
public static final String Place = "placeKey";

SharedPreferences sharedpreferences;
```

http://www.tutorialspoint.com/android/android_shared_preferences.htm

SharedPreferences example

```
String n   = name.getText().toString();
String ph  = phone.getText().toString();
String e   = email.getText().toString();
String s   = street.getText().toString();
String p   = place.getText().toString();
Editor editor = sharedpreferences.edit();
editor.putString(Name, n);
editor.putString(Phone, ph);
editor.putString(Email, e);
editor.putString(Street, s);
editor.putString(Place, p);

editor.commit();
```

```
sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

if (sharedpreferences.contains(Name))
{
   name.setText(sharedpreferences.getString(Name, ""));

}
if (sharedpreferences.contains(Phone))
{
   phone.setText(sharedpreferences.getString(Phone, ""));
```

Modes

int	MODE_APPEND File creation mode: for use with openFileOutput(String, int), if the file already exists then write data to the end of the existing file instead of erasing it.
int	MODE_ENABLE_WRITE_AHEAD_LOGGING Database open flag: when set, the database is opened with write-ahead logging enabled by default.
int	MODE_MULTI_PROCESS *This constant was deprecated in API level 23. MODE_MULTI_PROCESS does not work reliably in some versions of Android, and furthermore does not provide any mechanism for reconciling concurrent modifications across processes. Applications should not attempt to use it. Instead, they should use an explicit cross-process data management approach such as ContentProvider.*
int	MODE_NO_LOCALIZED_COLLATORS Database open flag: when set, the database is opened without support for localized collators.
int	MODE_PRIVATE File creation mode: the default mode, where the created file can only be accessed by the calling application (or all applications sharing the same user ID).
int	MODE_WORLD_READABLE *This constant was deprecated in API level 17. Creating world-readable files is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; instead, applications should use more formal mechanism for interactions such as ContentProvider, BroadcastReceiver, and Service. There are no guarantees that this access mode will remain on a file, such as when it goes through a backup and restore.*
int	MODE_WORLD_WRITEABLE *This constant was deprecated in API level 17. Creating world-writable files is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; instead, applications should use more formal mechanism for interactions such as ContentProvider, BroadcastReceiver, and Service. There are no guarantees that this access mode will remain on a file, such as when it goes through a backup and restore.*

https://developer.android.com/reference/android/content/Context.html

Editor methods

Public methods	
abstract void	apply() Commit your preferences changes back from this Editor to the SharedPreferences object it is editing.
abstract SharedPreferences.Editor	clear() Mark in the editor to remove *all* values from the preferences.
abstract boolean	commit() Commit your preferences changes back from this Editor to the SharedPreferences object it is editing.
abstract SharedPreferences.Editor	putBoolean(String key, boolean value) Set a boolean value in the preferences editor, to be written back once commit() or apply() are called.
abstract SharedPreferences.Editor	putFloat(String key, float value) Set a float value in the preferences editor, to be written back once commit() or apply() are called.
abstract SharedPreferences.Editor	putInt(String key, int value) Set an int value in the preferences editor, to be written back once commit() or apply() are called.
abstract SharedPreferences.Editor	putLong(String key, long value) Set a long value in the preferences editor, to be written back once commit() or apply() are called.
abstract SharedPreferences.Editor	putString(String key, String value) Set a String value in the preferences editor, to be written back once commit() or apply() are called.
abstract SharedPreferences.Editor	putStringSet(String key, Set<String> values) Set a set of String values in the preferences editor, to be written back once commit() or apply() is called.
abstract SharedPreferences.Editor	remove(String key) Mark in the editor that a preference value should be removed, which will be done in the actual preferences once commit() is called.

https://developer.android.com/reference/android/content/SharedPreferences.Editor.html

WRITING TO FILES

Storage options

Internal storage:

- It's always available.
- Files saved here are accessible by only your app by default.
- When the user uninstalls your app, the system removes all your app's files from internal storage.

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage:

- It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
- It's world-readable, so files saved here may be read outside of your control.
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from `getExternalFilesDir()`.

External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

File objects

You'll need the storage path. For the internal storage, use:

```
String path = context.getFilesDir().getAbsolutePath();
```

For the external storage (SD card), use:

```
String path = context.getExternalFilesDir(null).getAbsolutePath();
```

Then create your file object:

```
File file = new File(path + "/my-file-name.txt");
```

http://stackoverflow.com/questions/14376807/how-to-read-write-string-from-a-file-in-android

Write to files

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

```
FileOutputStream stream = new FileOutputStream(file);
try {
    stream.write("text-to-write".getBytes());
} finally {
    stream.close();
}
```

Read from files

```
int length = (int) file.length();

byte[] bytes = new byte[length];

FileInputStream in = new FileInputStream(file);
try {
    in.read(bytes);
} finally {
    in.close();
}

String contents = new String(bytes);
```

Get permissions

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
<manifest ...>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    ...
</manifest>
```

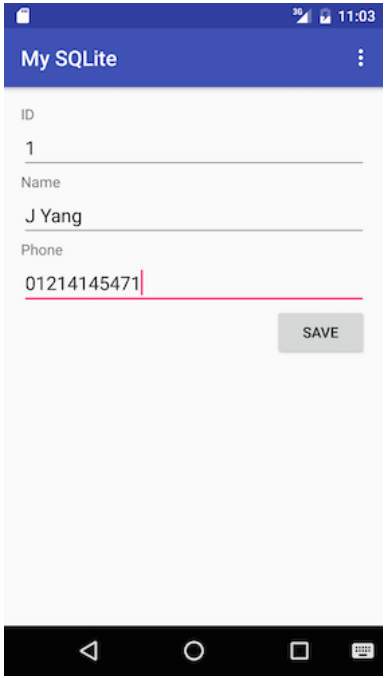
External storage

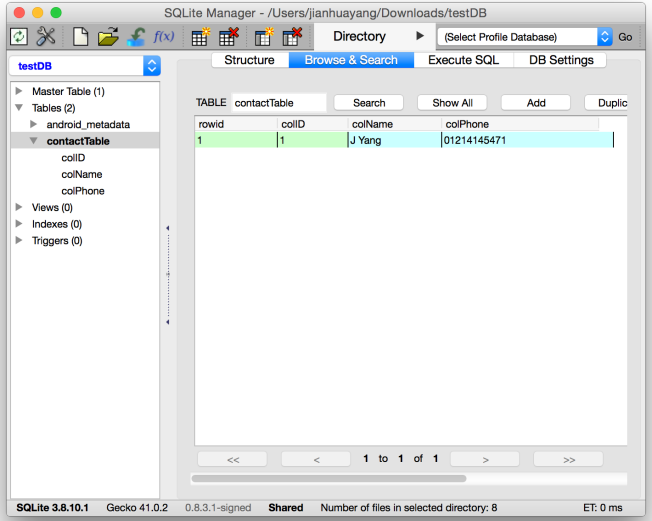
```java
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

- MEDIA_BAD_REMOVAL
- MEDIA_CHECKING
- MEDIA_MOUNTED
- MEDIA_MOUNTED_READ_ONLY
- MEDIA_NOFS
- MEDIA_REMOVED
- MEDIA_SHARED
- MEDIA_UNKNOWN
- MEDIA_UNMOUNTABLE
- MEDIA_UNMOUNTED

Lab 2

- SQL syntax
- SQLite package
- Contact app example
- Pre-created database





SQL SYNTAX

SQLite

Small. Fast. Reliable.
Choose any three.

Welcome.

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

Sponsors

Ongoing development and maintenance of SQLite is sponsored in part by SQLite Consortium members, including:

mozilla

Mozilla - Working to preserve choice and innovation on the internet.

Bentley Sustaining Infrastructure

Bentley - Comprehensive software solutions for Sustaining Infrastructure.

Bloomberg

Bloomberg - A world leader in financial-information technology.

Current Status

- Version 3.8.8.2 of SQLite is recommended for all new development. Upgrading from version 3.8.8 and 3.8.8.1 is optional. Upgrading from all other versions of SQLite is recommended.

Common Links

SQL commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

CREATE

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

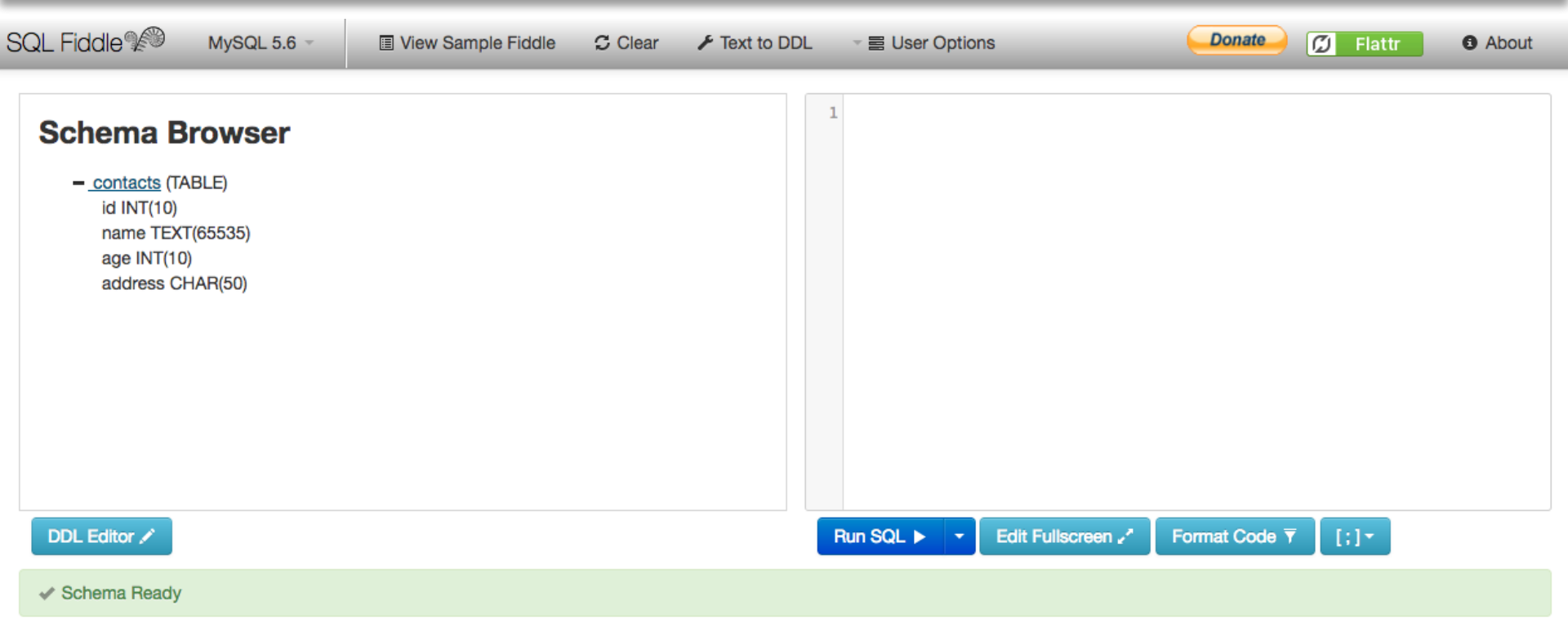
```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

SELECT

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL Fiddle



http://sqlfiddle.com

SQLITE PACKAGE

android.database.sqlite

SQLiteClosable	An object created from a SQLiteDatabase that can be closed.
SQLiteCursor	A Cursor implementation that exposes results from a query on a `SQLiteDatabase`.
SQLiteDatabase	Exposes methods to manage a SQLite database.
SQLiteOpenHelper	A helper class to manage database creation and version management.
SQLiteProgram	A base class for compiled SQLite programs.
SQLiteQuery	Represents a query that reads the resulting rows into a `SQLiteQuery`.
SQLiteQueryBuilder	This is a convience class that helps build SQL queries to be sent to `SQLiteDatabase` objects.
SQLiteStatement	Represents a statement that can be executed against a database.

Data manipulation

SQLiteDatabase

- Exposes methods to manage a SQLite database.
- SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.

Operation	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update (Modify)	UPDATE
Delete (Destroy)	DELETE

long	insert (String table, String nullColumnHack, ContentValues values) Convenience method for inserting a row into the database.

Cursor	rawQuery (String sql, String[] selectionArgs) Runs the provided SQL and returns a `Cursor` over the result set.

SQLite Helper

A useful set of APIs is available in the `SQLiteOpenHelper` class. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and *not during app startup*. All you need to do is call `getWritableDatabase()` or `getReadableDatabase()`.

abstract void	onCreate (SQLiteDatabase db) Called when the database is created for the first time.
void	onDowngrade (SQLiteDatabase db, int oldVersion, int newVersion) Called when the database needs to be downgraded.
void	onOpen (SQLiteDatabase db) Called when the database has been opened.
abstract void	onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) Called when the database needs to be upgraded.

SQLite Helper

public **SQLiteOpenHelper** (Context context, String name, SQLiteDatabase.CursorFactory factory, int version) Added in API level 1

Create a helper object to create, open, and/or manage a database. This method always returns very quickly. The database is not actually created or opened until one of `getWritableDatabase()` or `getReadableDatabase()` is called.

Parameters

context	to use to open or create the database
name	of the database file, or null for an in-memory database
factory	to use for creating cursor objects, or null for the default
version	number of the database (starting at 1); if the database is older, `onUpgrade(SQLiteDatabase, int, int)` will be used to upgrade the database; if the database is newer, `onDowngrade(SQLiteDatabase, int, int)` will be used to downgrade the database

The database that you create for an application is only accessible to itself; other applications will not be able to access it.

/data/data/<package_name>/databases

A SIMPLE EXAMPLE

SQLite example

Table Structure

Table Name: Contacts

Field	Type	Key
id	INT	PRI
name	TEXT	
phone_number	TEXT	

http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/

Contact class

Contact.java

```
package com.androidhive.androidsqlite;

public class Contact {

    //private variables
    int _id;
    String _name;
    String _phone_number;

    // Empty constructor
    public Contact(){

    }
    // constructor
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }

    // constructor
    public Contact(String name, String _phone_number){
        this._name = name;
        this._phone_number = _phone_number;
    }

    // getting ID
    public int getID(){
        return this._id;
    }

    // setting id
    public void setID(int id){
        this._id = id;
    }

    // getting name
    public String getName(){
        return this._name;
    }

    // setting name
    public void setName(String name){
        this._name = name;
    }

    // getting phone number
    public String getPhoneNumber(){
        return this._phone_number;
    }

    // setting phone number
    public void setPhoneNumber(String phone_number){
        this._phone_number = phone_number;
    }
}
```

Database handler

```
public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "contactsManager";

    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";

    // Contacts Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";

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

Database handler

```
// Creating Tables
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
            + KEY_PH_NO + " TEXT" + ")";
    db.execSQL(CREATE_CONTACTS_TABLE);
}

// Upgrading database
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Drop older table if existed
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

    // Create tables again
    onCreate(db);
}
```

CRUD Operations

```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

addContact()

```
addContact()

    // Adding new contact
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName()); // Contact Name
    values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number

    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

addContact()

public long **insert** (String table, String nullColumnHack, ContentValues values) Added in API level 1

Convenience method for inserting a row into the database.

Parameters

table	the table to insert the row into
nullColumnHack	optional; may be `null`. 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.
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 row ID of the newly inserted row, or -1 if an error occurred

getContact()

```
getContact()
    // Getting single contact
public Contact getContact(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
            KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
            new String[] { String.valueOf(id) }, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();

    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    // return contact
    return contact;
}
```

Cursor	query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy) Query the given table, returning a `Cursor` over the result set.
abstract String	getString (int columnIndex) Returns the value of the requested column as a String.

getContact()

```
String[] tableColumns = new String[] {
    "column1",
    "(SELECT max(column1) FROM table2) AS max"
};
String whereClause = "column1 = ? OR column1 = ?";
String[] whereArgs = new String[] {
    "value1",
    "value2"
};
String orderBy = "column1";
Cursor c = sqLiteDatabase.query("table1", tableColumns, whereClause, whereArgs,
        null, null, orderBy);

// since we have a named column we can do
int idx = c.getColumnIndex("max");
```

is equivalent to the following raw query

```
String queryString =
    "SELECT column1, (SELECT max(column1) FROM table1) AS max FROM table1 " +
    "WHERE column1 = ? OR column1 = ? ORDER BY column1";
sqLiteDatabase.rawQuery(queryString, whereArgs);
```

http://stackoverflow.com/questions/10600670/sqlitedatabase-query-method

getAllContacts()

```
getAllContacts()
    // Getting All Contacts
 public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<Contact>();
    // Select All Query
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }

    // return contact list
    return contactList;
}
```

abstract boolean	moveToFirst () Move the cursor to the first row.
abstract boolean	moveToLast () Move the cursor to the last row.
abstract boolean	moveToNext () Move the cursor to the next row.

getAllContacts()

Questions Tags Users

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

What does it mean to “program to an interface”?

The specific example I used to give to students is that they should write

```
List myList = new ArrayList(); // programming to the List interface
```

instead of

```
ArrayList myList = new ArrayList(); // this is bad
```

http://stackoverflow.com/questions/383947/what-does-it-mean-to-program-to-an-interface
http://stackoverflow.com/questions/2279030/type-list-vs-type-arraylist-in-java

Update and delete

updateContact()

```
    // Updating single contact
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

deleteContact()

```
    // Deleting single contact
public void deleteContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
    db.close();
}
```

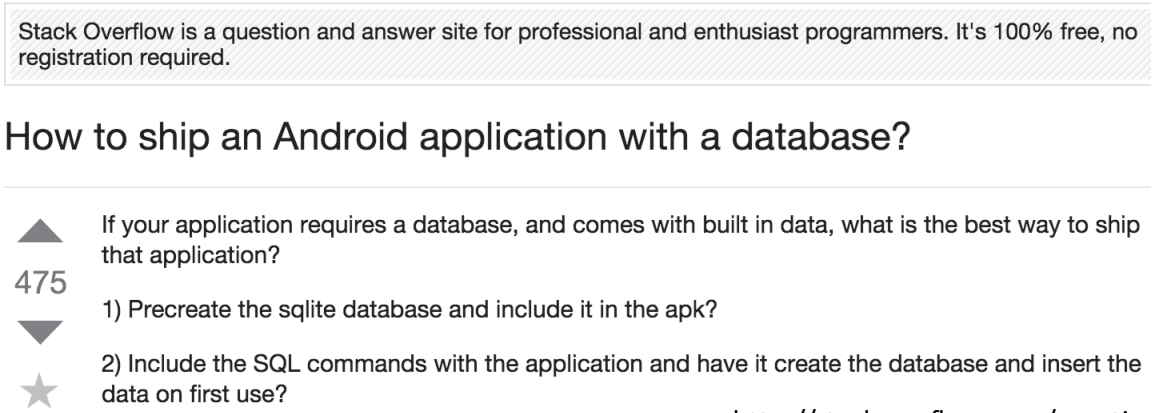
Use database

```
DatabaseHandler db = new DatabaseHandler(this);

/**
 * CRUD Operations
 * */
// Inserting Contacts
Log.d("Insert: ", "Inserting ..");
db.addContact(new Contact("Ravi", "9100000000"));
db.addContact(new Contact("Srinivas", "9199999999"));
db.addContact(new Contact("Tommy", "9522222222"));
db.addContact(new Contact("Karthik", "9533333333"));

// Reading all contacts
Log.d("Reading: ", "Reading all contacts..");
List<Contact> contacts = db.getAllContacts();

for (Contact cn : contacts) {
    String log = "Id: "+cn.getID()+" ,Name: " + cn.getName()
        // Writing Contacts to log
Log.d("Name: ", log);
```

PRE-CREATED DATABASE

Pre-created database

1. Create your database,
2. Put it in your assets directory in your apk
3. On first use copy to "/data/data/YOUR_PACKAGE/databases/" directory.



Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

How to ship an Android application with a database?

475

If your application requires a database, and comes with built in data, what is the best way to ship that application?

1) Precreate the sqlite database and include it in the apk?

2) Include the SQL commands with the application and have it create the database and insert the data on first use?

http://stackoverflow.com/questions/513084/ship-an-application-with-a-database

Get the system path

```
DBAdapter db = new DBAdapter(this);
try {
    String destPath = "/data/data/" + getPackageName() +
        "/databases";
    File f = new File(destPath);
    if (!f.exists()) {
        f.mkdirs();
        f.createNewFile();

        //---copy the db from the assets folder into
        // the databases folder---
        CopyDB(getBaseContext().getAssets().open("mydb"),
            new FileOutputStream(destPath + "/MyDB"));
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Copy the database

```
public void CopyDB(InputStream inputStream,
OutputStream outputStream) throws IOException {
    //---copy 1K bytes at a time---
    byte[] buffer = new byte[1024];
    int length;
    while ((length = inputStream.read(buffer)) > 0) {
        outputStream.write(buffer, 0, length);
    }
    inputStream.close();
    outputStream.close();
}
```

Use the copied database

```
//---get all contacts---
db.open();
Cursor c = db.getAllContacts();
if (c.moveToFirst())
{
    do {
        DisplayContact(c);
    } while (c.moveToNext());
}
db.close();
```