Mobile App Development

SharedPreferences and SQLite

Dr. Christelle Scharff
cscharff@pace.edu
Pace University, USA

Objectives

- Understand and use SharedPreferences for settings and preferences
- Understand and use SQLite
- Review SQL
- Understand how to interact with databases directly in Java without using SQL
- Manipulate Android Java methods for select, insert, delete, update etc.
- Mention the use of mobile backends (e.g., Firebase)

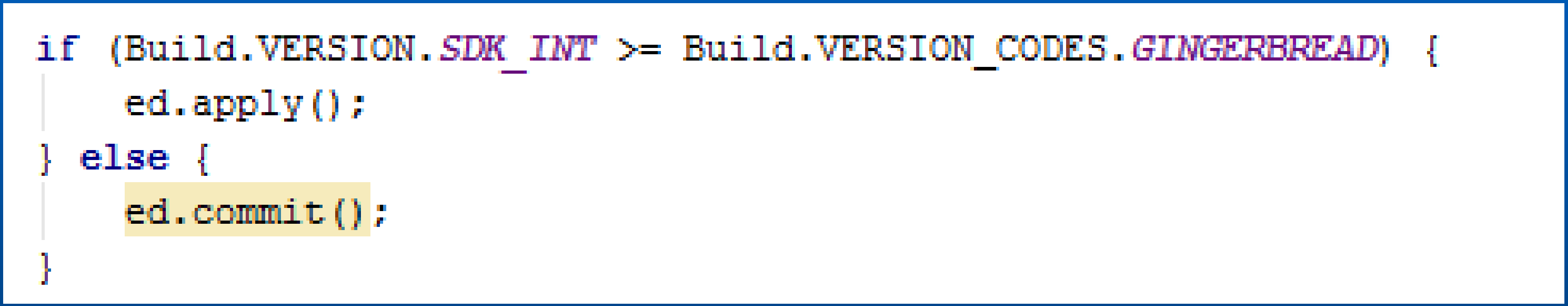
SharedPreferences

SharedPreferences

- SharedPreferences is typically used to save user settings and preferences (username, password, language etc.)
- It is easily accessible from any component of the application (activities, services etc.)
- It uses key / value pairs
- It writes to the disk with the `commit` method or the `apply` method depending on the SDK version
  - commit - < Android SDK 2.3
  - apply - >= Android SDK 2.3
- http://developer.android.com/reference/android/content/SharedPreferences.html

Android SDK Versions

- It is often required to deal with different Android SDK versions

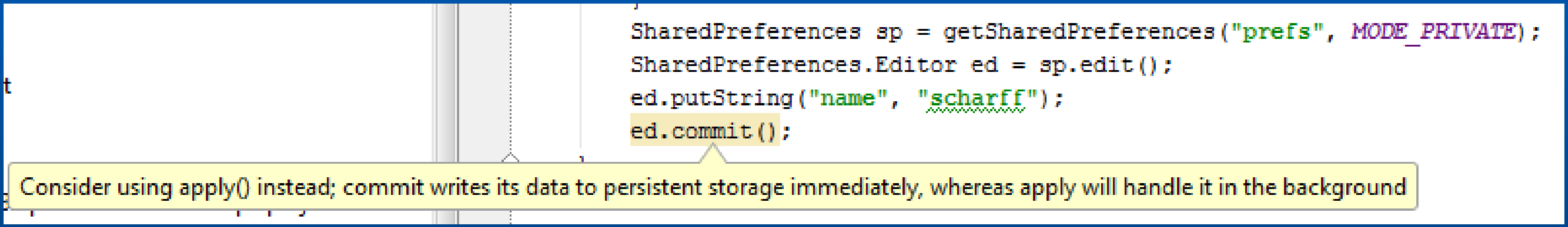


```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
    ed.apply();
} else {
    ed.commit();
}
```

- http://developer.android.com/reference/android/os/Build.VERSION.html
- http://developer.android.com/reference/android/os/Build.VERSION_CODES.html

Put and Get

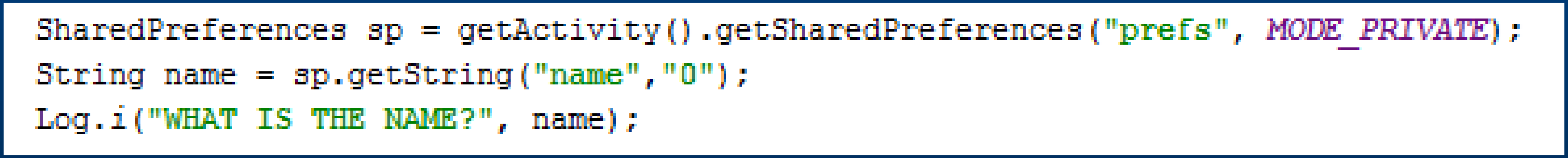
- Persistence of the name from one screen to another
- Save the name in one activity (onCreate method)



```
SharedPreferences sp = getSharedPreferences("prefs", MODE_PRIVATE);
SharedPreferences.Editor ed = sp.edit();
ed.putString("name", "scharff");
ed.commit();
```

Consider using apply() instead; commit writes its data to persistent storage immediately, whereas apply will handle it in the background

- Get the name in another activity



```
SharedPreferences sp = getActivity().getSharedPreferences("prefs", MODE_PRIVATE);
String name = sp.getString("name","0");
Log.i("WHAT IS THE NAME?", name);
```

SQLite and SQL

SQLite

- SQLite is a very popular embedded database engine written in Ansi C and available in the public domain
  - http://www.sqlite.org
- SQLite implements a self-contained, serverless, zero-configuration, transactional SQL database engine
- SQLite size can be less than 300KB depending on compiler optimization settings (180KB without optional features)
- SQLite can also be made to run in minimal stack space (4KB) and very little heap (100KB)
  - SQLite is adapted to devices with memory constraints such as mobile phones, PDAs and MP3 players
- Supports terabyte-sized databases and gigabyte-sized strings and blobs
- SQLite does not support foreign keys and some of the advanced SQL features

SQLite

- SQLite reads and writes directly from and to ordinary disk files (encrypted or not)
  - A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file
  - The database file format is cross-platform
- Standard SQL commands
  - Create databases
  - Create tables
  - Insert, select, update and delete operations
- Capability to handle transactions via the ACID properties
  - Atomicity, Consistency, Isolation and Durability
  - E.g., atomicity to group SQL commands together is implemented with commit and rollback

SQL Review

- `CREATE DATABASE`
- `CREATE TABLE`
- `INSERT INTO`
- `DELETE FROM`
- `UPDATE`
- `SELECT … FROM … WHERE … GROUP BY … HAVING …`
- `DROP TABLE`
- Try out some commands here: http://sqlfiddle.com/

Android and SQLite

Android and SQLite

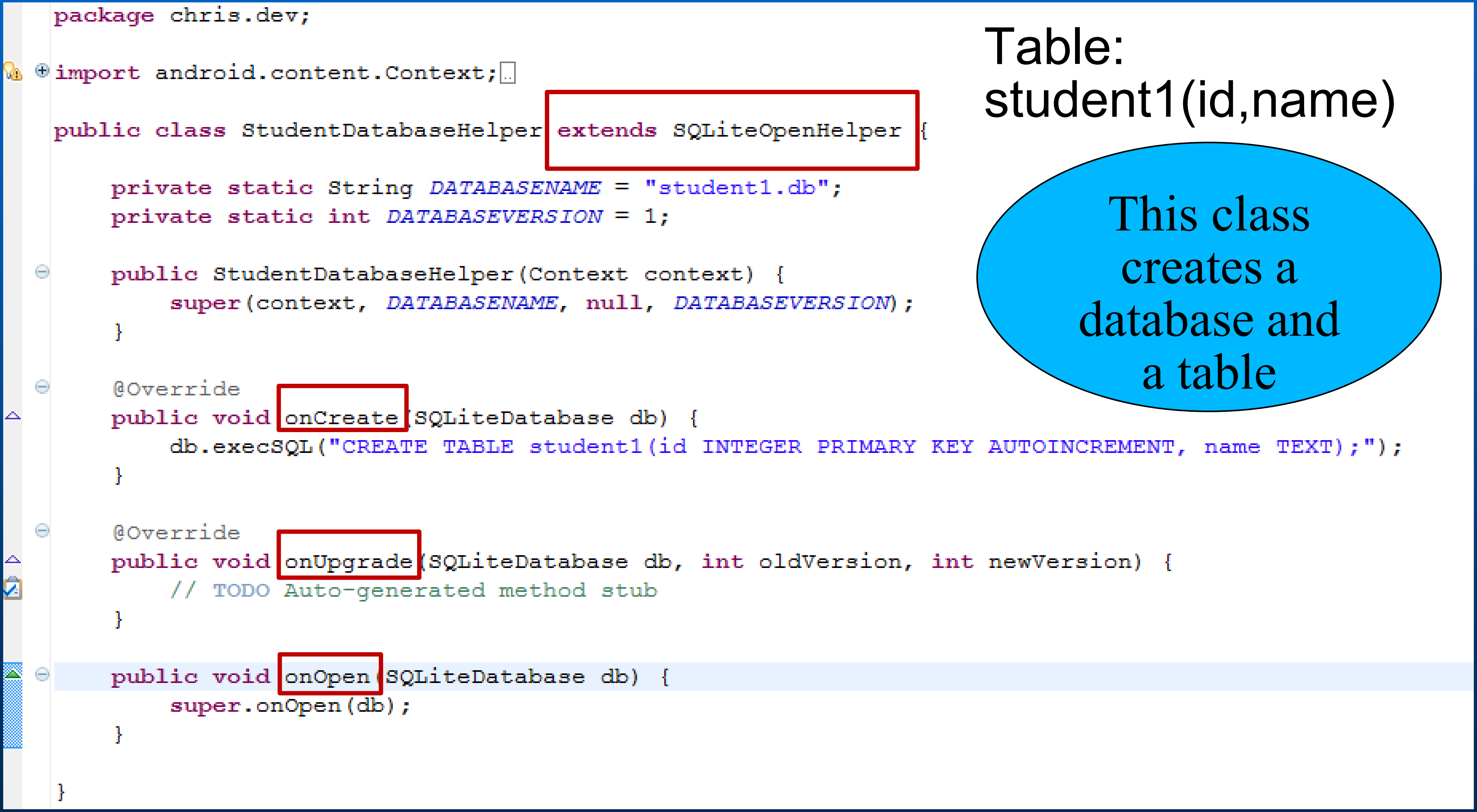
- Android provides an interface for an app to interact with SQLite
  - Creation of a connection to the database through `SQLiteOpenHelper` and an instance of `SQLiteDatabase`
  - An application creates a database that is used for the rest of the application’s lifetime
  - Database creation and versioning
  - Database management
  - Query builder helpers to format proper SQL statements programmatically instead of writing SQL statements
  - Cursor objects to iterate query results
  - Specialized SQLite exceptions
- To share data between applications, the data have to be exposed by making the application a content provider (*Not covered here)*

S T E P S

SQLiteOpenHelper

- The creation of the connection to the database requires the subclassing of `SQLiteOpenHelper` and the use of `SQLiteDatabase`
- Methods of `SQLiteOpenHelper`
  - `onCreate` - Executed when the database is created for the first time
  - `onUpdate` - Executed when the database is upgraded
  - `onOpen` – Executed when the database has been open
- http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html
- http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Creation of the Class: *StudentDatabaseHelper*



```
package chris.dev;

import android.content.Context;

public class StudentDatabaseHelper extends SQLiteOpenHelper {

    private static String DATABASENAME = "student1.db";
    private static int DATABASEVERSION = 1;

    public StudentDatabaseHelper(Context context) {
        super(context, DATABASENAME, null, DATABASEVERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE student1(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // TODO Auto-generated method stub
    }

    public void onOpen(SQLiteDatabase db) {
        super.onOpen(db);
    }

}
```

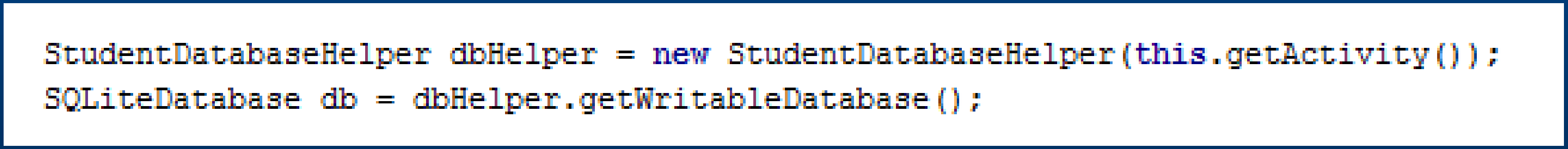
Table: student1(id,name)

This class creates a database and a table

Using the Class: *StudentDatabaseHelper*

- `SQLiteDatabase` offers the following methods:
  - `insert`
  - `query`
  - `update`
  - `delete`
  - `execSQL`

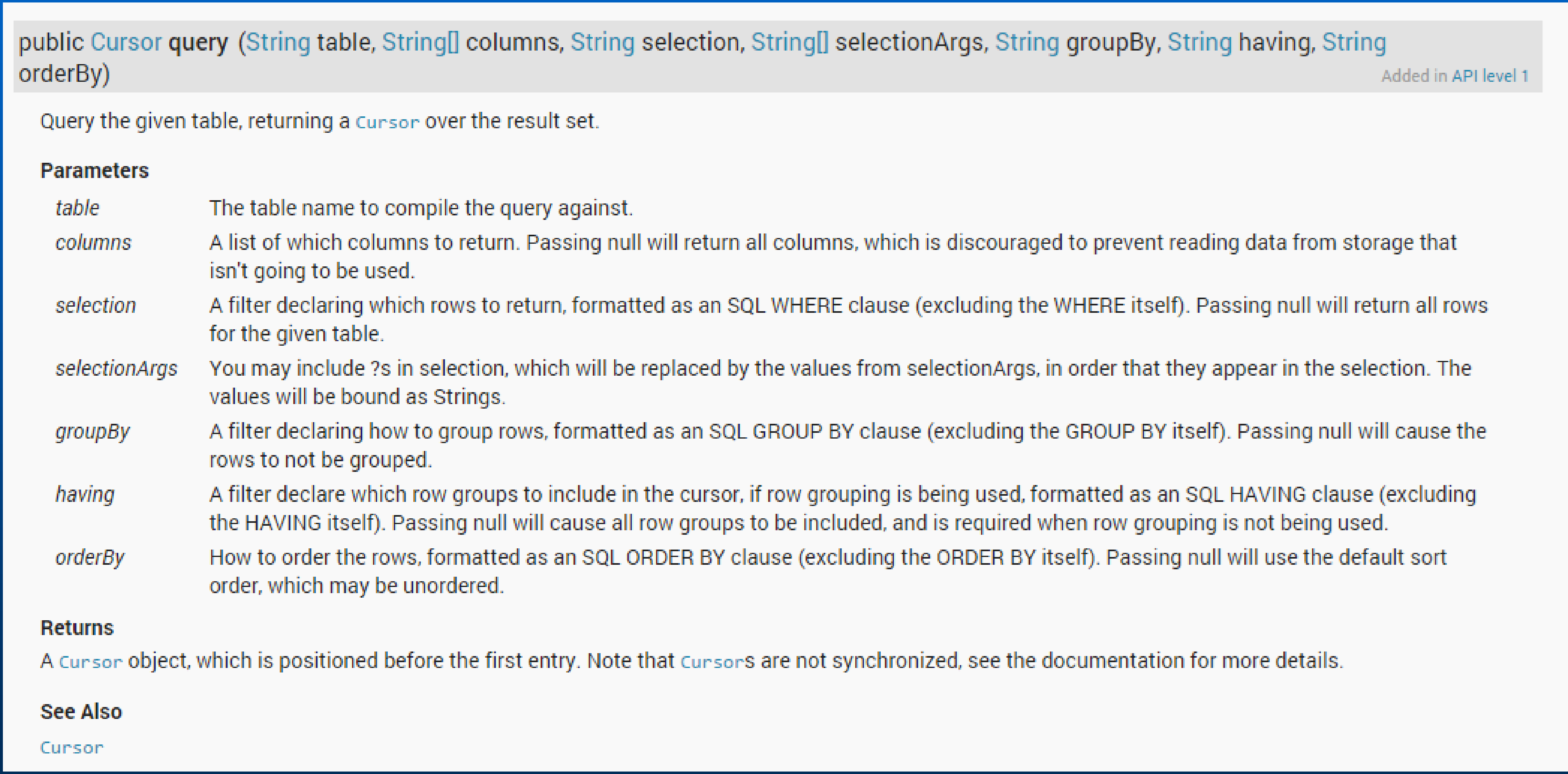
Insert, query, update and delete do NOT use SQL



```
StudentDatabaseHelper dbHelper = new StudentDatabaseHelper(this.getActivity());
SQLiteDatabase db = dbHelper.getWritableDatabase();
```

- http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Method *query* of *SQLiteOpenHelper*



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

Added in API level 1

Query the given table, returning a `Cursor` over the result set.

Parameters

table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which 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	A filter declaring 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	A filter declare 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.

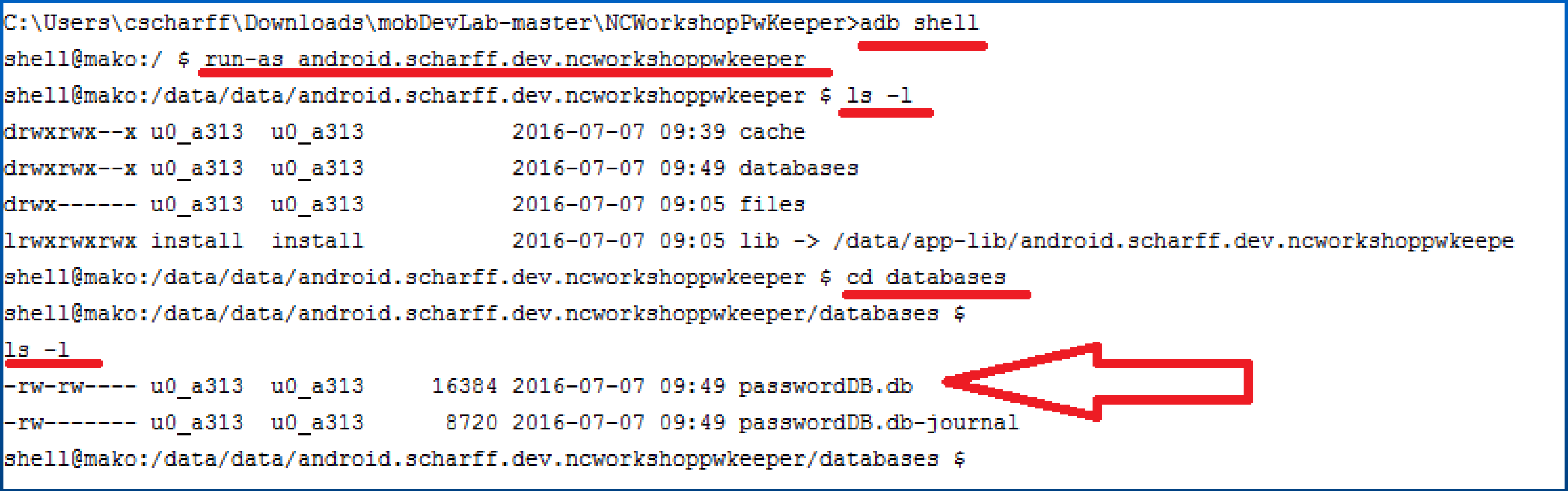
Returns

A `Cursor` object, which is positioned before the first entry. Note that `Cursor`s are not synchronized, see the documentation for more details.

See Also

`Cursor`

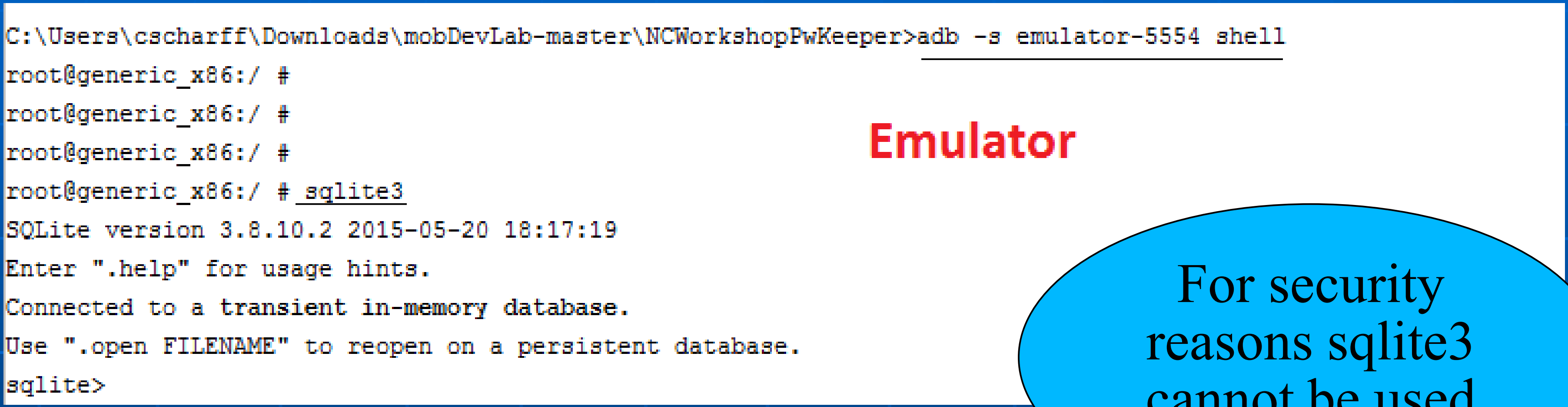
Accessing the Database on the Device or Emulator



```
C:\Users\cscharff\Downloads\mobDevLab-master\NCWorkshopPwKeeper>adb shell
shell@mako:/ $ run-as android.scharff.dev.ncworkshoppwkeeper
shell@mako:/data/data/android.scharff.dev.ncworkshoppwkeeper $ ls -l
drwxrwx--x u0_a313  u0_a313           2016-07-07 09:39 cache
drwxrwx--x u0_a313  u0_a313           2016-07-07 09:49 databases
drwx------ u0_a313  u0_a313           2016-07-07 09:05 files
lrwxrwxrwx install  install           2016-07-07 09:05 lib -> /data/app-lib/android.scharff.dev.ncworkshoppwkeepe
shell@mako:/data/data/android.scharff.dev.ncworkshoppwkeeper $ cd databases
shell@mako:/data/data/android.scharff.dev.ncworkshoppwkeeper/databases $
ls -l
-rw-rw---- u0_a313  u0_a313     16384 2016-07-07 09:49 passwordDB.db
-rw------- u0_a313  u0_a313      8720 2016-07-07 09:49 passwordDB.db-journal
shell@mako:/data/data/android.scharff.dev.ncworkshoppwkeeper/databases $
```

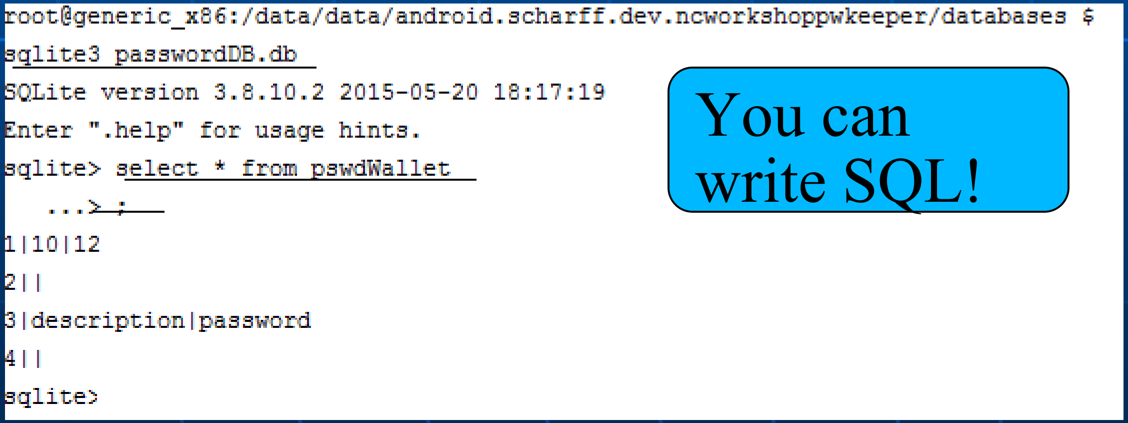
- *Code to see if a database was created (here passwordDB.db)*
- *ls –l is a command to list the elements of a folder (See linux/unix)*
- *Be sure that adb and sqlite3 are in your path*

Accessing the Database on the **Emulator**



```
C:\Users\cscharff\Downloads\mobDevLab-master\NCWorkshopPwKeeper>adb -s emulator-5554 shell
root@generic_x86:/ #
root@generic_x86:/ #
root@generic_x86:/ #
root@generic_x86:/ # sqlite3
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

For security reasons sqlite3 cannot be used on the device

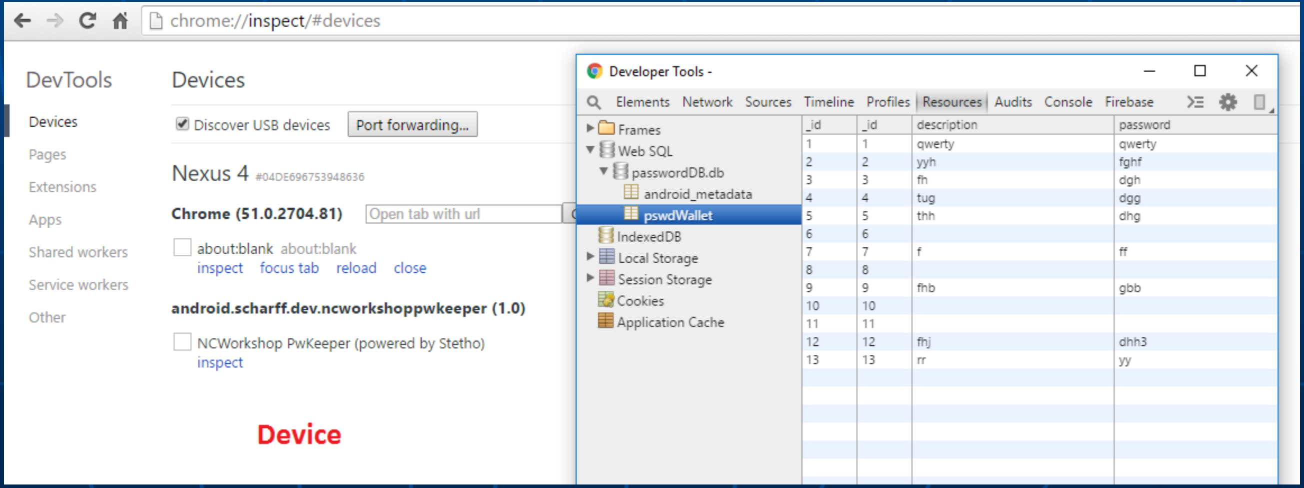


```
root@generic_x86:/data/data/android.scharff.dev.ncworkshoppwkeeper/databases $
sqlite3 passwordDB.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> select * from pswdWallet
   ...> ;
1|10|12
2||
3|description|password
4||
sqlite>
```

You can write SQL!

Accessing the Database on the Device or Emulator

- Use Stetho by Facebook
  - http://facebook.github.io/stetho/
  - Stetho is a complete debugger for Android apps and more
- Tutorial
  - http://code.tutsplus.com/tutorials/debugging-android-apps-with-facebooks-stetho--cms-24205



Devices

☑ Discover USB devices [Port forwarding...]

Nexus 4 #04DE696753948636

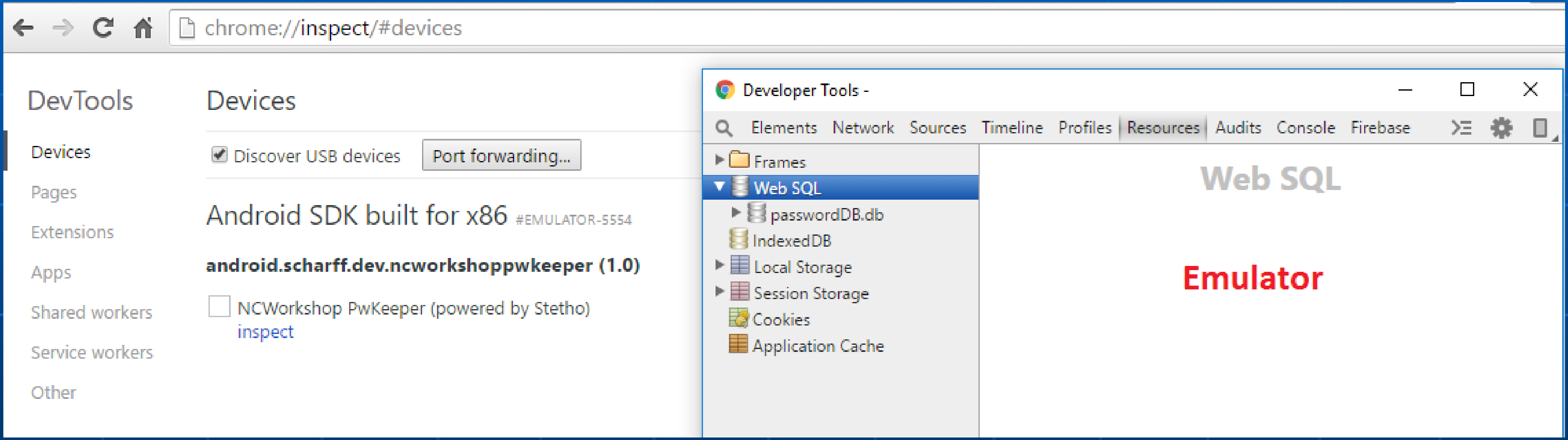
Chrome (51.0.2704.81) Open tab with url

about:blank about:blank
inspect focus tab reload close

android.scharff.dev.ncworkshoppwkeeper (1.0)

NCWorkshop PwKeeper (powered by Stetho)
inspect

_id	_id	description	password
1	1	qwerty	qwerty
2	2	yyh	fghf
3	3	fh	dgh
4	4	tug	dgg
5	5	thh	dhg
6	6		
7	7	f	ff
8	8		
9	9	fhb	gbb
10	10		
11	11		
12	12	fhj	dhh3
13	13	rr	yy



chrome://inspect/#devices

DevTools

- Devices
- Pages
- Extensions
- Apps
- Shared workers
- Service workers
- Other

Devices

☑ Discover USB devices | Port forwarding...

Android SDK built for x86 #EMULATOR-5554

android.scharff.dev.ncworkshoppwkeeper (1.0)

☐ NCWorkshop PwKeeper (powered by Stetho)
inspect

Developer Tools -

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console | Firebase

- Frames
- Web SQL
  - passwordDB.db
- IndexedDB
- Local Storage
- Session Storage
- Cookies
- Application Cache

Web SQL

Emulator

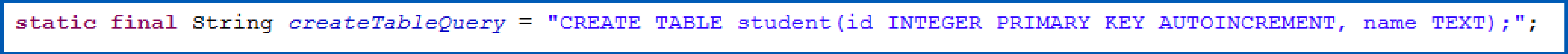
Why not using SQL?

- For security reasons
  - E.g., SQL injection
- Executing SQL is not efficient
  - Expressions are parsed each time
- Mixing SQL and Java generates lots of errors
- `SQLiteOpenHelper` and other helper classes are more robust
  - Errors in SQL can be detected before running the code!

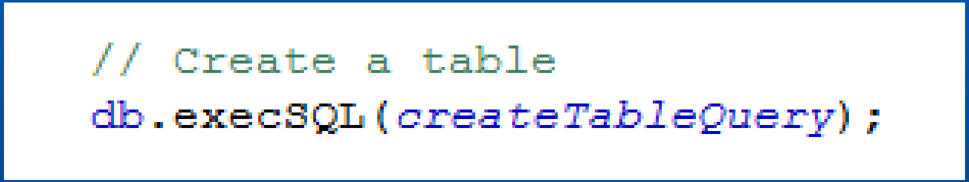
Examples

Create a Table

- Simplest way to create a table:



```
static final String createTableQuery = "CREATE TABLE student(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);";
```

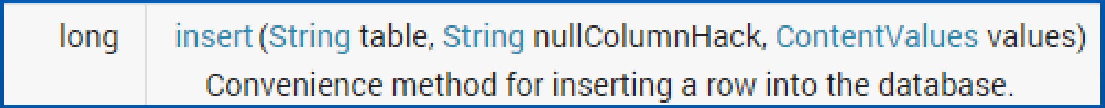


```
// Create a table
db.execSQL(createTableQuery);
```

- execSQL can be used for any type of SQL queries and commands (e.g., update, delete…)

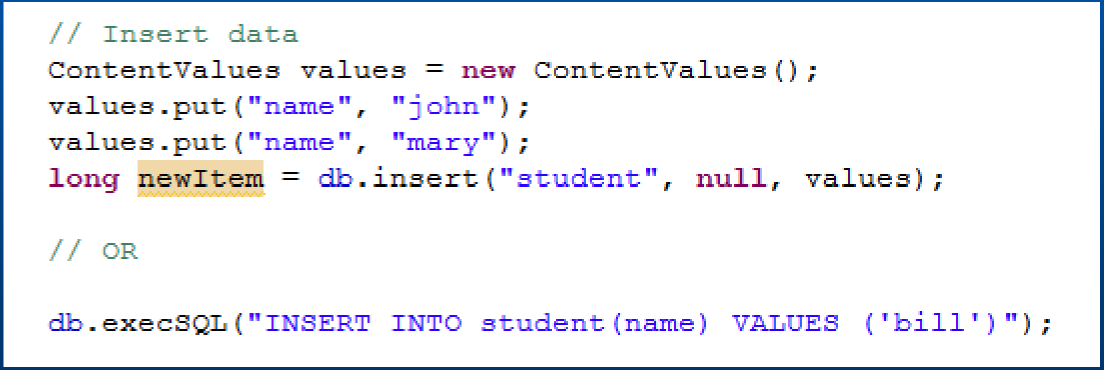
Insertion / Deletion

- replace, insert and update can use `ContentValues` to contain values
  - http://developer.android.com/reference/android/content/ContentValues.html



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

- Insertion

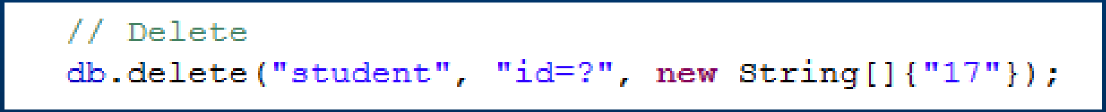


```
// Insert data
ContentValues values = new ContentValues();
values.put("name", "john");
values.put("name", "mary");
long newItem = db.insert("student", null, values);

// OR

db.execSQL("INSERT INTO student(name) VALUES ('bill')");
```

- Deletion



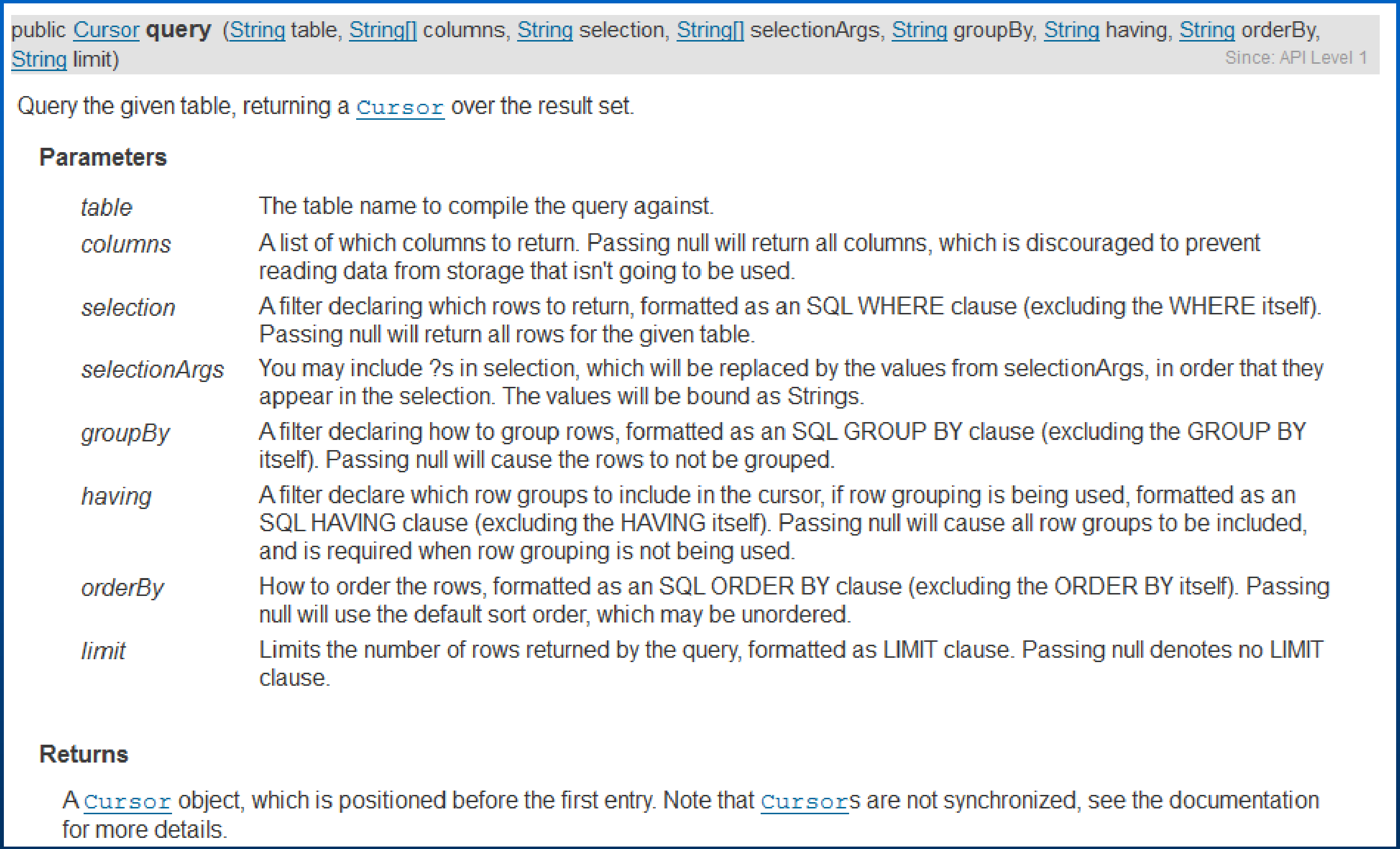
```
// Delete
db.delete("student", "id=?", new String[]{"17"});
```

// OR

Delete from student where id = 17

Selection

- The query method uses the following parameters:



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

Since: API Level 1

Query the given table, returning a `Cursor` over the result set.

Parameters

table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which 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	A filter declaring 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	A filter declare 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.
limit	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

Returns

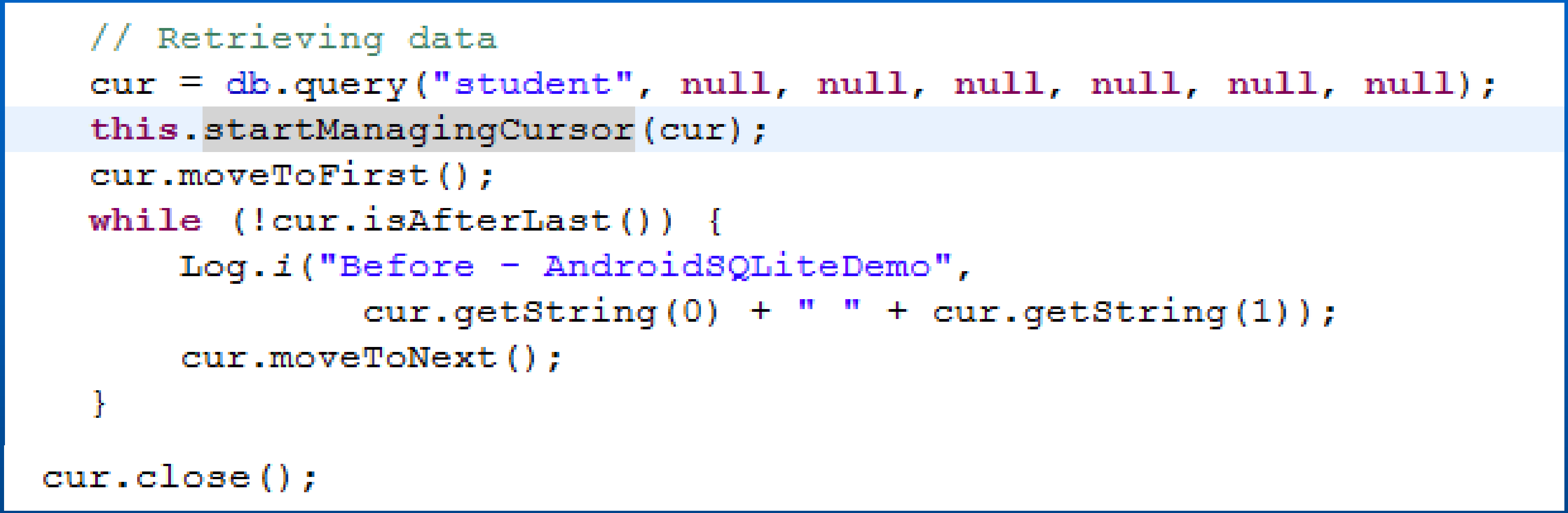
A `Cursor` object, which is positioned before the first entry. Note that `Cursor`s are not synchronized, see the documentation for more details.

http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Selection

- Use a cursor to iterate on rows using:
  - `moveToFirst()`
  - `moveToNext()`
  - `isAfterLast()`
- Cursors must be managed during the application life cycle. The activity can manage it also using `startManagingCursor()`

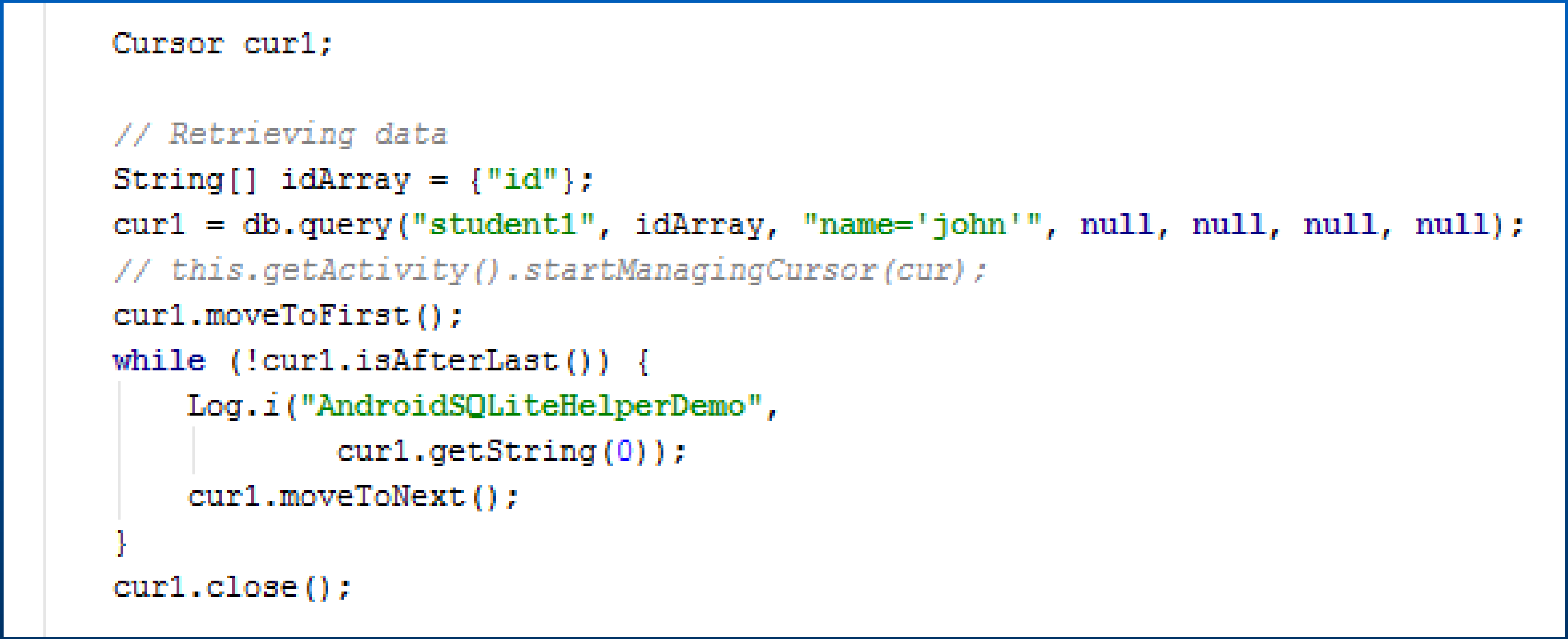
select * from student



```
    // Retrieving data
    cur = db.query("student", null, null, null, null, null, null);
    this.startManagingCursor(cur);
    cur.moveToFirst();
    while (!cur.isAfterLast()) {
        Log.i("Before - AndroidSQLiteDemo",
                cur.getString(0) + " " + cur.getString(1));
        cur.moveToNext();
    }

  cur.close();
```

select id from student1 where name=‘john’



```
Cursor cur1;

// Retrieving data
String[] idArray = {"id"};
cur1 = db.query("student1", idArray, "name='john'", null, null, null, null);
// this.getActivity().startManagingCursor(cur);
cur1.moveToFirst();
while (!cur1.isAfterLast()) {
    Log.i("AndroidSQLiteHelperDemo",
            cur1.getString(0));
    cur1.moveToNext();
}
cur1.close();
```

Mobile App Backend Platforms

Check out http://firebase.com