

# Kotlin Android Databases



CS 402: Mobile Development



# Additional Reading

<http://zetcode.com/db/sqlite/introduction/>



# Databases

Database store data types in Entities.

Databases store a set of data in rows. Each unique piece of data of similar structure, gets its own row.

Attributes are defined by columns in the table

Tables can be linked by foreign keys to a row's id



# Database Relationships

A table row may be linked to one other table row. This is called a **one to one relationship**.

Example: A building has one location

A table row can be linked to one or many other rows in a second table. This is called a **one to many relationship**.

Example: A car dealership may have many cars on its lot

Table rows may be linked to many other rows, and those rows may be linked back to many rows. This is called a **many to many relationship**.

Example: A movie has many actors. Actors may be in many other movies.



# Database Tables

Tables store sets of data

Similar to a Java Class

Entity name: Coordinates

Table: Coordinates		
id	latitude	longitude
1	1.1	10.1
2	1.2	10.2
3	1.1	10.2
4	5.4	5.2
5	8.3	-4.4
6	9.0	-4.5
7	10.2	-1.2
8	5.6	-4.4
9	15.3	5.3
10	15.4	5.6
11	16.9	3.2



# Database Columns

Columns define the attributes of a table

Similar to members in a Class

Example: id, latitude, longitude

**Table: Coordinates**

id	latitude	longitude
1	1.1	10.1
2	1.2	10.2
3	1.1	10.2
4	5.4	5.2
5	8.3	-4.4
6	9.0	-4.5
7	10.2	-1.2
8	5.6	-4.4
9	15.3	5.3
10	15.4	5.6
11	16.9	3.2



# Database Rows

Rows define instances of data

Similar to an instance of a Class

Each row usually has an ID, or a unique primary key, to identify it

You can use any unique field as a primary key but an ID is preferred since it will never change.

Table: Coordinates		
id	latitude	longitude
1	1.1	10.1
2	1.2	10.2
3	1.1	10.2
4	5.4	5.2
5	8.3	-4.4
6	9.0	-4.5
7	10.2	-1.2
8	5.6	-4.4
9	15.3	5.3
10	15.4	5.6
11	16.9	3.2



# Database Relations

Tables are linked together by foreign keys


A column with an ID can point to a row in another table

Table Name: Locations		
id	location_name	location_id
1	Library	1
2	CompSci 497 class room	5
4	CompSci Department	5
5	Stadium	2
7	Student Center	3
8	Bee hives	3
9	Parking Garage	4



# Linking Tables

Table Name: Locations				Table: Coordinates		
id	location_name	location_id	description	id	latitude	longitude
1	Library	1	Where books get checked out.	1	1.1	10.1
2	CompSci 497 class room	5	Mobile Development class.	2	1.2	10.2
4	CompSci Department	5	Houses a bunch of important people.	3	1.1	10.2
5	Stadium	2	Blue turf grows here.	4	5.4	5.2
7	Student Center	3	Games and things.	5	8.3	-4.4
8	Bee hives	3	Buzzing can be heard here.	6	9.0	-4.5
9	Parking Garage	4	A place where people forget where they parked.	7	10.2	-1.2
				8	5.6	-4.4
				9	15.3	5.3
				10	15.4	5.6
				11	16.9	3.2



# JOINing Two or More Entities in a Single Query

You can merge a single row from one table into the column of another table's row using a JOIN. In other words, you can make a link from an attribute to another instance of data in another table.

Ex: A movie entity may have a Director entity. No need to duplicate the director's information in that movie entity, so link to it instead using a foreign key. That foreign key is an id to the director in the Director's table.

<http://zetcode.com/db/sqlite/joins/>



# SQLite

SQLite is an easy-to-use desktop database program. Used in Android and iOS.

SQLite is comprised of a single executable file, and does not have a server component.

SQLite databases are stored in a single file that can easily be backed-up and transferred from one computer to another.



# SQLite

SQLite is open source software, and is available for Windows, Mac, Linux, and Unix systems.

The standard SQLite database engine uses a command line interface, but there are also several GUIs available. Android and iOS use their own classes to interact with SQLite.

The SQLite command line executable is pre-installed on Mac OSX, and most Linux and Unix based systems.



# SQLite

When multiple users will need update your database at the same time.

When users will need to access your database remotely (However, they can still download a static clone to their device).



# SQL Insert

```
INSERT INTO table_name(column1,column2,column3,...)VALUES  
(value1,value2,value3,...);
```

```
INSERT INTO Coordinates(latitude, longitude)VALUES (28.123,-114.123);
```

```
INSERT INTO Locations(location_name, location_id, description)VALUES  
("Name", [coordinate_id], "Description");
```



# SQL Select

```
SELECT * FROM Locations WHERE id=2
```



# SQL Update

```
UPDATE Locations set location_name="new name"WHERE  
id=4
```

```
UPDATE Locations set location_id=5
```





# SQL Delete

```
DELETE FROM Locations WHERE id=2
```



# SQL on Android

<http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

<http://developer.android.com/training/basics/data-storage/databases.html>

<http://www.vogella.com/articles/AndroidSQLite/article.html>

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



# Data Contract

```
public final class LocationsContract{
    public LocationsContract(){}
    public static abstract class LocationsEntry implements BaseColumns
    {
        /* LOCATIONS TABLE */
        public static final String TABLE_LOCATIONS    = "Locations";
        public static final String COLUMN_LOCATION_NAME = "location_name";
        public static final String COLUMN_LOCATION_ID   = "location_id";
        public static final String COLUMN_DESCRIPTION  = "description";

        /* COORDINATES TABLE */
        public static final String TABLE_COORDINATES = "Coordinates";
        public static final String COLUMN_LATITUDE    = "latitude";
        public static final String COLUMN_LONGITUDE   = "longitude";
    }
}
```



# Coordinate Model

```
public class GPSCoordinate
{
    private int coordinateID;

    private double latitude;

    private double longitude;

    /* Getters and Setters ... */
}
```



# Location Model

```
public class Location
{
    private int locationID;

    private String locationName;

    private String locationDescription;

    private GPSCoordinate locationCoordinates;

    /* Getters and Setters ... */
}
```



# Database Controller

```
public class DatabaseController extends SQLiteOpenHelper{
    /* DATABASE INFO */
    private static final String DB_NAME = "android_db";
    private static final int DB_VERSION = 1;
    private mCursorFactory = null;

    public DatabaseController( Context context )
    {
        super( context, DB_NAME, mCursorFactory, DB_VERSION );
    }

    /* Overrides, query methods ... */
}
```



# Create Table

```
CREATE TABLE Coordinates( _id integer primary key  
autoincrement,  latitude real, longitude real)
```



# Create Table

```
CREATE TABLE Locations( _id integer primary key  
autoincrement, location_name varchar(255),  
location_id integer references Coorindates(_id),  
description varchar(1023))
```





# onCreate()

```
@Override public void onCreate(SQLiteDatabase database){  
    // Create Coordinates Table  
    String locationsTableQuery = "CREATE TABLE " +  
        LocationsContract.LocationsEntry.TABLE_LOCATIONS + "(" +  
        "_id integer primary key autoincrement, " +  
        LocationsContract.LocationsEntry.COLUMN_LATITUDE + " real," +  
        LocationsContract.LocationsEntry.COLUMN_LONGITUDE + " real," + ")";  
  
    database.execSQL( locationsTableQuery );  
  
    // Create Locations Table  
    String coordinatesTableQuery = "CREATE TABLE " +  
        LocationsContract.LocationsEntry.TABLE_COORDINATES +  
        "(" +  
            "_id integer primary key autoincrement, " +  
            LocationsContract.LocationsEntry.COLUMN_LOCATION_NAME + " varchar(255)," +  
            LocationsContract.LocationsEntry.COLUMN_LOCATION_ID + " integer references " +  
            LocationsContract.LocationsEntry.TABLE_LOCATIONS + "(_id)," +  
            LocationsContract.LocationsEntry.COLUMN_DESCRIPTION + " varchar(1023)," + ")";  
  
    database.execSQL( coordinatesTableQuery );  
}
```



# onUpgrade()

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
    switch ( oldVersion )
    {
    case 1:
        // Adds new table Majors: db version 2
    case 2:
        // Adds new column 'times' to table Majors: db version 3
    case 3:
        // Removes 'times' from table Majors: db version 4
    Default:
        break; // Unhandled migration
    }
}
```



## Add a Record

```
public void addLocation( Location newLocation ){  
    ....  
}
```

```
private void addLocationAsync( Location newLocation ){  
    ....  
}
```



# Create New AsyncTask

```
public void addLocation( Location newLocation ){  
  
    AddLocationTask addLocationTask = new AddLocationTask();  
  
    addLocationTask.execute(newLocation);  
  
}
```



# AsyncTask Definition

```
private class AddLocationTask extends AsyncTask<Location, Void, Void>{

    @Override
    protected Void doInBackground( Location...parameters )
    {
        for (Location currentLocation : parameters) {
            addLocationAsync(currentLocation);
        }

        return null;
    }
}
```



# AsyncTask

```
private void addLocationAsync( Location newLocation ){
    ContentValues contentValues = new ContentValues();
    contentValues.put( LocationsContract.LocationsEntry.COLUMN_LATITUDE, newLocation.getLocationCoordinates().getLatitude() );
    contentValues.put( LocationsContract.LocationsEntry.COLUMN_LONGITUDE, newLocation.getLocationCoordinates().getLongitude() );
    long coordinateID = getWritableDatabase().insert( LocationsContract.LocationsEntry.TABLE_COORDINATES, null, contentValues);

    contentValues = new ContentValues();
    contentValues.put( LocationsContract.LocationsEntry.COLUMN_LOCATION_NAME, newLocation.getLocationName() );
    contentValues.put( LocationsContract.LocationsEntry.COLUMN_DESCRIPTION, newLocation.getLocationDescription() );
    contentValues.put( LocationsContract.LocationsEntry.COLUMN_LOCATION_ID,
        coordinateID );

    long locationID = getWritableDatabase().insert( LocationsContract.LocationsEntry.TABLE_LOCATIONS, null, contentValues);
}
```



There's a better way

ObjectBox