Android Lecture #10

What are we going to do today?

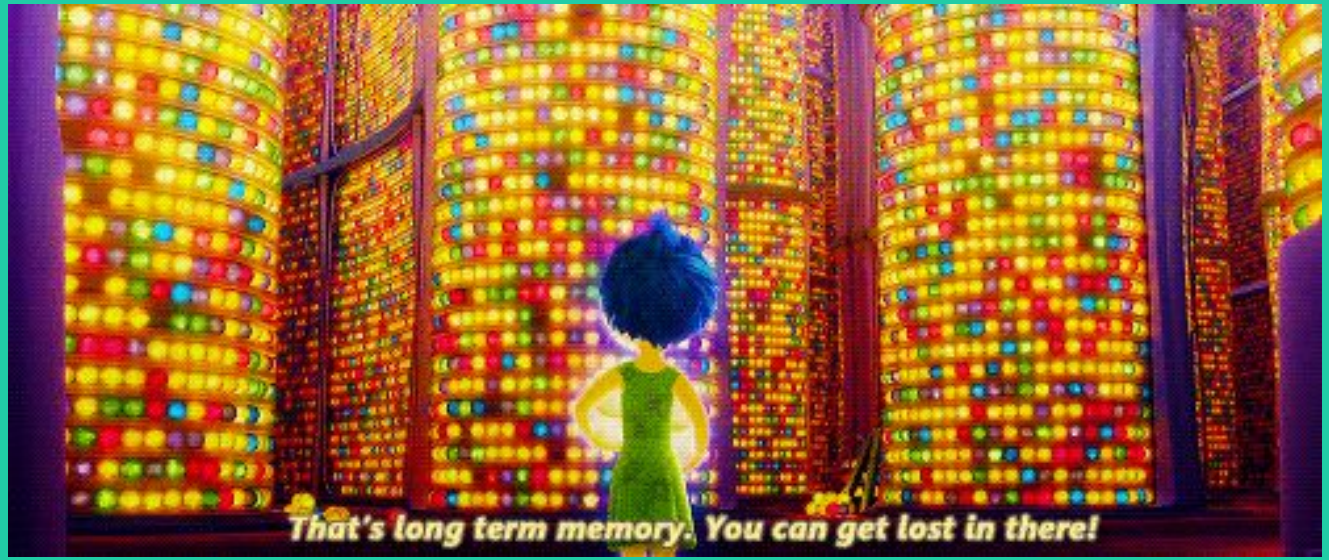
Persistency

SharedPref

SQL

SQLite/Room

Persistency



Let’s Persist!

- Long term memory
- Storage for data
- Cache - faster loading time

How Much Faster?

```
L1 cache reference ......................... 0.5 ns
L2 cache reference ........................... 7 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

Number Every Programmer Must Know
https://gist.github.com/hellerbarde/2843375

How Much Faster?

250 µs VS 150 ms

what's the big deal?

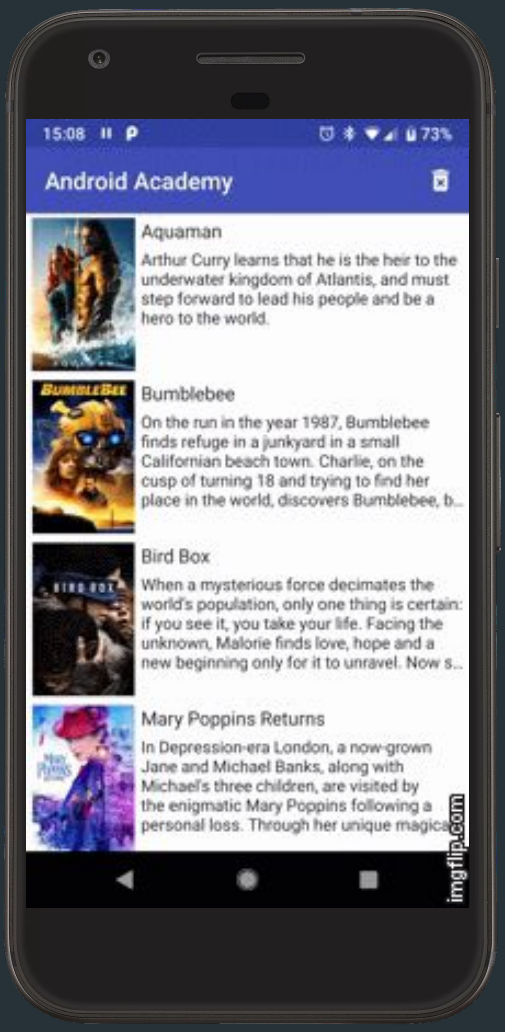
How Much Faster?

Let's multiply these durations by a billion

How Much Faster?

```
Read 1 MB sequentially from memory  =  2.9 days

Like A long weekend
```

```
Send packet CA->Netherlands->CA  =  4.8 years

Like Average time it takes to complete a bachelor's degree
```


15:08
73%
Android Academy
Aquaman
Arthur Curry learns that he is the heir to the underwater kingdom of Atlantis, and must step forward to lead his people and be a hero to the world.
Bumblebee
On the run in the year 1987, Bumblebee finds refuge in a junkyard in a small Californian beach town. Charlie, on the cusp of turning 18 and trying to find her place in the world, discovers Bumblebee, b...
Bird Box
When a mysterious force decimates the world's population, only one thing is certain: if you see it, you take your life. Facing the unknown, Malorie finds love, hope and a new beginning only for it to unravel. Now s...
Mary Poppins Returns
In Depression-era London, a now-grown Jane and Michael Banks, along with Michael's three children, are visited by the enigmatic Mary Poppins following a personal loss. Through her unique magica
imgflip.com

SharedPreferences

Key Value XML based Storage

- Small storage space
- No structure (key-value)
- Not secured
- Primitive types (booleans, floats, ints, longs, and strings)
- Complex objects serialize to String

Creating SharedPreferences

```
Context context = getActivity();
```

Creating SharedPreferences

```
Context context = getActivity();
SharedPreferences sharedPref
```

Creating SharedPreferences

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                                                            ,
                                                           );
```

Creating SharedPreferences

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                    "preference_file_key”,
                                        );
```

Creating SharedPreferences

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                    "preference_file_key”,
                     Context.MODE_PRIVATE);
```

Creating SharedPreferences

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

Adding Data

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

Adding Data

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                        getString(R.string.preference_file_key),
                        Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_STRING","Android is awesome!");
```

Adding Data

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                        getString(R.string.preference_file_key),
                        Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_STRING","Android is awesome!");
editor.putInt("KEY_INT",1);
editor.putFloat("KEY_FLOAT",1F);
editor.putBoolean("KEY_BOOLEAN",true);
editor.putLong("KEY_LONG",1L);
```

Saving Data - Commit

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                    getString(R.string.preference_file_key),
                    Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_STRING","Android is awesome!");
...
boolean success = editor.commit();
```

Saving Data - Commit

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                    getString(R.string.preference_file_key),
                    Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_STRING","Android is awesome!");
...
boolean success = editor.commit() //blocking, returns success indication
```

Saving Data - Apply

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
                        getString(R.string.preference_file_key),
                        Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_STRING","Android is awesome!");
...
editor.apply(); //background thread, no return value
```

Saving Data - Commit vs Apply

- commit - we know the result
- apply - we have no idea what happened.

Loading Data

```
SharedPreferences sharedPref = context.getSharedPreferences(
                        getString(R.string.preference_file_key),
                        Context.MODE_PRIVATE);
String myString = sharedPref.getString("KEY_STRING", null);

int myInt = sharedPref.getInt("KEY_INT", 0);

Float myFloat = sharedPref.getFloat("KEY_FLOAT", 0F);

Boolean myBoolean = sharedPref.getBoolean("KEY_BOOLEAN", false);

Long myLong = sharedPref.getLong("KEY_LONG", 0L);
```

A Look From The Inside

Command Prompt - adb shell

```
generic_x86:/data/data/tlv.androidacademy.sample/shared_prefs $
generic_x86:/data/data/tlv.androidacademy.sample/shared_prefs $ ls -l
total 8
-rw-rw---- 1 u0_a74 u0_a74 331 2019-01-01 13:15 tlv.androidacademy.sample_preferences.xml
generic_x86:/data/data/tlv.androidacademy.sample/shared_prefs $
```

A Look From The Inside - XML

Command Prompt - adb shell

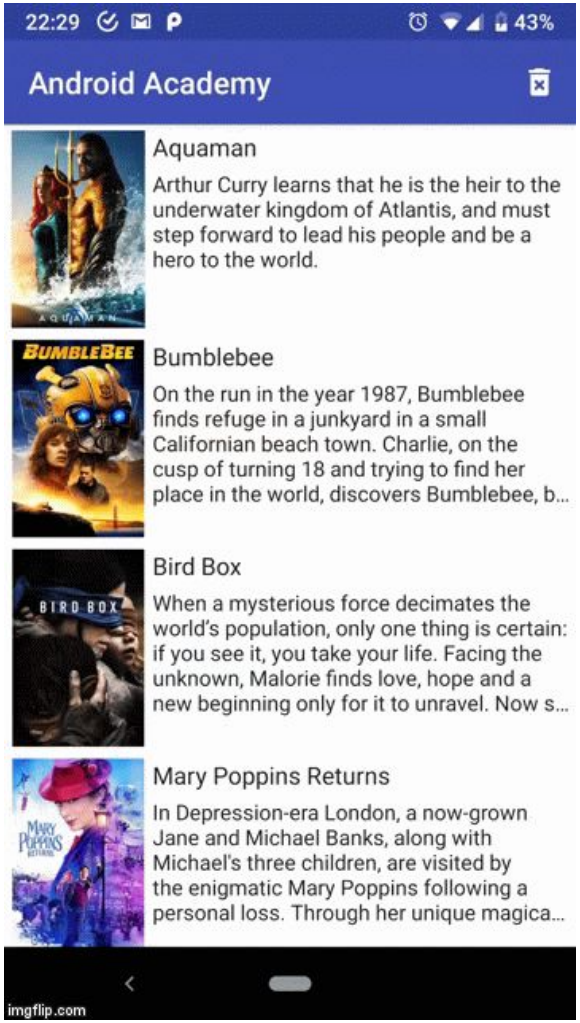
```
generic_x86:/data/data/tlv.androidacademy.sample/shared_prefs $ cat tlv.androidacademy.sample_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="KEY_INT" value="1" />
    <float name="KEY_FLOAT" value="1.0" />
    <long name="KEY_LONG" value="1" />
    <string name="KEY_STRING">Android Academy is awesome!</string>
    <set name="KEY_SET" />
    <boolean name="KEY_BOOLEAN" value="true" />
</map>
generic_x86:/data/data/tlv.androidacademy.sample/shared_prefs $
```

When to use Shared Preferences

- User preferences!
- Global state - user id, user token, isLoggedIn etc.
- Small, unstructured data
- Unsecured - very easy to take a look inside.

AndroidManifest - backup

```
<manifest ... >
    ...
    <application android:allowBackup="true" ... >
        ...
    </application>
</manifest>
```


22:29
43%
Android Academy
Aquaman
Arthur Curry learns that he is the heir to the underwater kingdom of Atlantis, and must step forward to lead his people and be a hero to the world.
Bumblebee
On the run in the year 1987, Bumblebee finds refuge in a junkyard in a small Californian beach town. Charlie, on the cusp of turning 18 and trying to find her place in the world, discovers Bumblebee, b...
Bird Box
When a mysterious force decimates the world's population, only one thing is certain: if you see it, you take your life. Facing the unknown, Malorie finds love, hope and a new beginning only for it to unravel. Now s...
Mary Poppins Returns
In Depression-era London, a now-grown Jane and Michael Banks, along with Michael's three children, are visited by the enigmatic Mary Poppins following a personal loss. Through her unique magica...
imgflip.com

Database

Database

Database:

A structured set of data

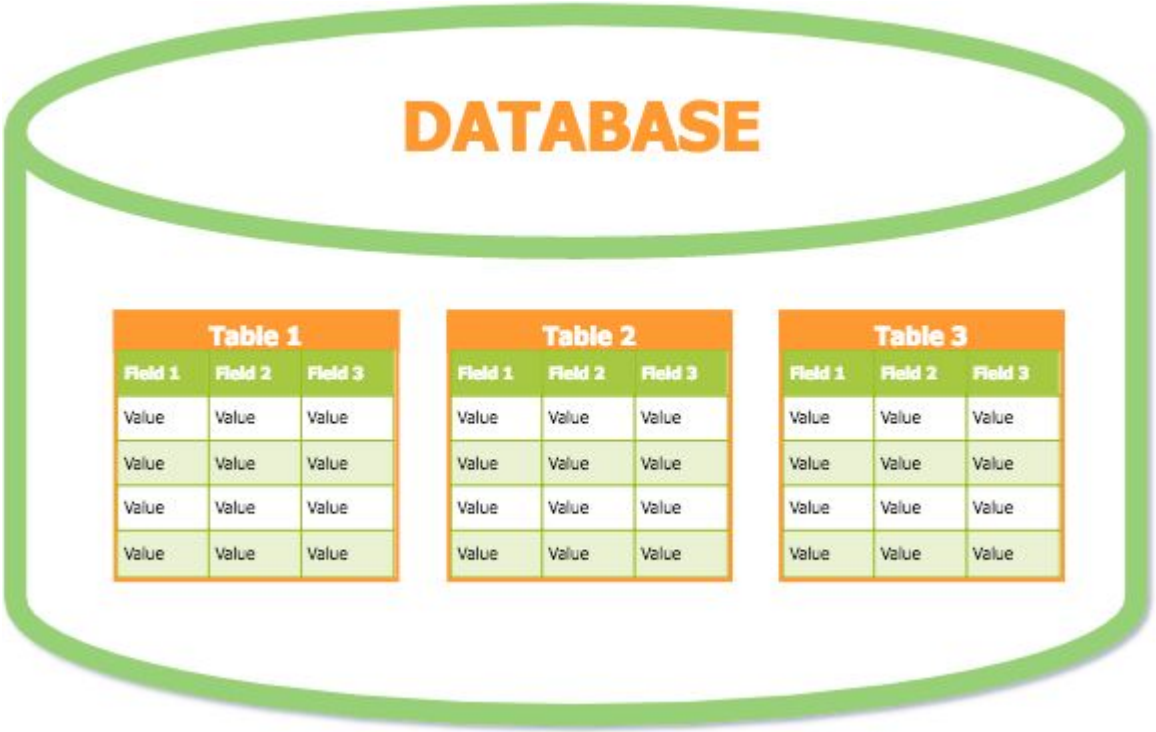
Relational database:

A Database which uses tables and relations to organize the data,
Usually uses SQL for operations

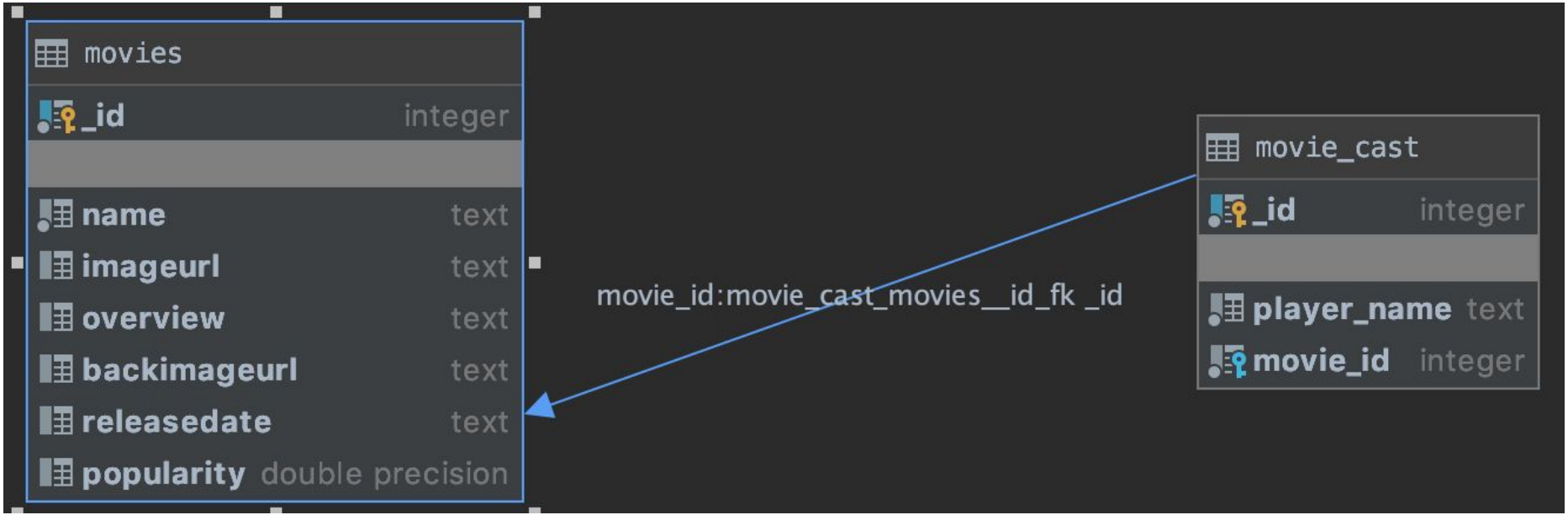
Relational Database management system:

A program (*or less*) that implements a Relational Database.
Common vendors:PostgreSQL, Oracle, MySQL, Microsoft SQL Server

Database


DATABASE
Table 1
Field 1 Field 2 Field 3
Value Value Value
Value Value Value
Value Value Value
Value Value Value
Table 2
Field 1 Field 2 Field 3
Value Value Value
Value Value Value
Value Value Value
Value Value Value
Table 3
Field 1 Field 2 Field 3
Value Value Value
Value Value Value
Value Value Value
Value Value Value

Tables


movies
_id integer
name text
imageurl text
overview text
backimageurl text
releasedate text
popularity double precision
movie_id:movie_cast_movies__id_fk _id
movie_cast
_id integer
player_name text
movie_id integer

When to use a Database - requirements

- Large datasets
- Structured data
- Cache / Preload data

When to use a Database - results

- Better loading times
- Better battery utilization
- Better network utilization by saving traffic

Our Database

SQLite is a mini-RDBMS.

Unlike most:

- Serverless (Runs in your process, not on its own)
- Zero-Configuration
- Most widely deployed database

Get to know it better: https://www.sqlite.org/

SQL - Structured Query Language

the language to communicate with database.

Creating our table

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

CREATE TABLE statement

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

Table Name

```
CREATE TABLE Movies (

    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL

)
```

Read More: https://www.sqlite.org/lang_createtable.html

Primary Key

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

Column Types

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

Column Types

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

Column Types

```
CREATE TABLE Movies (
    movieId INTEGER PRIMARY KEY,
    name TEXT,
    imageUri TEXT,
    releaseDate TEXT,
    popularity REAL
)
```

Read More: https://www.sqlite.org/lang_createtable.html

movieId	name	imageUri	releaseDate	popularity
297802	Aquaman	https://image.tmdb.org/t/p/w342/i2dF9UxOeb77CAJrOflj0RpqJRF.jpg	2018-12-07	616.683
299536	Avengers: Infinity War	https://image.tmdb.org/t/p/w342/7WsyChQLEftFiDOVTGkv3hFpyyt.jpg	2018-04-25	128.347
324857	Spider-Man: Into the Spider-Verse	https://image.tmdb.org/t/p/w342/laMM4lpQSh5z6KIBPwWoqkjzBVQ.jpg	2018-12-07	152.037
335983	Venom	https://image.tmdb.org/t/p/w342/2uNW4WbgBXL25BAbXGLnLqX71Sw.jpg	2018-10-03	227.227
338952	Fantastic Beasts: The Crimes of Grindelwald	https://image.tmdb.org/t/p/w342/kQKcbJ9uYkTQql2R8L4jTUz7l90.jpg	2018-11-14	272.343

Actions:

- ***Insert*** - Adds records to tables.
- ***Update*** - modifies values in existing records
- ***Select*** - gets part of the table
- ***Delete*** - Removes records from tables.

Select - gets part of the table

```
SELECT ( * | [column, column])
FROM (table)
WHERE (condition)
ORDER BY (column) (ASC | DESC)
```

This is super-simplified. Read more: https://www.sqlite.org/lang_select.html

```
SELECT *
FROM Movies
WHERE name = 'Aquaman'
```

movieId	name	imageUri	releaseDate	popularity
297802	Aquaman	https://image.tmdb.org/t/p/w342/i2dF9UxOeb77CAJrOflj0RpqJRF.jpg	2018-12-07	616.683

```
SELECT *
FROM Movies
ORDER BY popularity ASC
```

movieId	name	imageUri	releaseDate	popularity
299536	Avengers: Infinity War	https://image.tmdb.org/t/p/w342/7WsyChQLEftFiDOVTGkv3hFpyyt.jpg	2018-04-25	128.347
324857	Spider-Man: Into the Spider-Verse	https://image.tmdb.org/t/p/w342/laMM4IpQSh5z6KIBPwWogkjzBVQ.jpg	2018-12-07	152.037
335983	Venom	https://image.tmdb.org/t/p/w342/2uNW4WbgBXL25BAbXGLnLgX71Sw.jpg	2018-10-03	227.227
338952	Fantastic Beasts: The Crimes of Grindelwald	https://image.tmdb.org/t/p/w342/kQKcbJ9uYkTQqI2R8L4jTUz7I90.jpg	2018-11-14	272.343
297802	Aquaman	https://image.tmdb.org/t/p/w342/i2dF9UxOeb77CAJrOflj0RpgJRF.jpg	2018-12-07	616.683

```
SELECT *
FROM Movies
WHERE name LIKE '% the %'
```

movieId	name	imageUri	releaseDate	popularity
338952	Fantastic Beasts: the Crimes of Grindelwald	https://image.tmdb.org/t/p/w342/kQKcbJ9uYkTQql2R8L4jTUz7l90.jpg	2018-11-14	272.343
324857	Spider-Man: Into the Spider-Verse	https://image.tmdb.org/t/p/w342/laMM4lpQSh5z6KIBPwWogkjzBVQ.jpg	2018-12-07	152.037

```
SELECT *
FROM Movies
WHERE title LIKE '% the %'
AND popularity > 200
```

movieId	name	imageUri	releaseDate	popularity
338952	Fantastic Beasts: the Crimes of Grindelwald	https://image.tmdb.org/t/p/w342/kQKcbJ9uYkTQql2R8L4jTUz7l90.jpg	2018-11-14	272.343

Databases - The Old Way

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    // Table Name
    public static final String TABLE_NAME = "MOVIE_ENTRY";





}
```

SQLiteOpenHelper

```java
public class DatabaseHelper extends SQLiteOpenHelper {

    // Table Name
    public static final String TABLE_NAME = "MOVIE_ENTRY";

    // Table columns
    public static final String COLUMN_NAME_ID = "id";
    public static final String COLUMN_NAME_NAME = "name";
    public static final String COLUMN_NAME_CONTENT = "content";
    public static final String COLUMN_NAME_IMAGE_URI = "image_uri";
    public static final String COLUMN_NAME_RELEASE_DATE = "release_date";

}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...

    // Database Information
    static final String DB_NAME = "MoviesDb.db";



}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...

    // Database Information
    static final String DB_NAME = "MoviesDb.db";

    // database version
    static final int DB_VERSION = 1;

}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    // Creating table query
    private static final String SQL_CREATE_ENTRIES

                                                    ;

    }
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    // Creating table query
    private static final String SQL_CREATE_ENTRIES = "CREATE TABLE " + TABLE_NAME +




                                                                                ;

    }
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    // Creating table query
    private static final String SQL_CREATE_ENTRIES = "CREATE TABLE " + TABLE_NAME +
"(" +
            COLUMN_NAME_ID + " INTEGER PRIMARY KEY, " +




                                                    ;

    }
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    // Creating table query
    private static final String SQL_CREATE_ENTRIES = "CREATE TABLE " + TABLE_NAME +
"(" +
            COLUMN_NAME_ID + " INTEGER PRIMARY KEY, " +
            COLUMN_NAME_NAME + " TEXT, " +
            COLUMN_NAME_CONTENT + " TEXT, " +
            COLUMN_NAME_RELEASE_DATE + " TEXT, " +
            COLUMN_NAME_IMAGE_URI + " TEXT);";

    }
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...

    public DatabaseHelper(Context context) {
        super(context,                                  );
    }


}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
     // Database Information
    static final String DB_NAME = "MoviesDb.db";

    public DatabaseHelper(Context context) {
        super(context, DB_NAME,                    );
    }


}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
     // database version
    static final int DB_VERSION = 1;

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


}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...

    public DatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }


}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }

}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    ...
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

Save Data

Save Data - Best practice

- Get DB
- Create transaction
- save data
- Close transaction
- Close DB connection

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private SQLiteDatabase database;





}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private SQLiteDatabase database;

    public void open() throws SQLException {
        database = this.getWritableDatabase();
    }



}
```

SQLiteOpenHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private SQLiteDatabase database;

    public void open() throws SQLException {
        database = this.getWritableDatabase();
    }

    public void close() {
        database.close();
    }

}
```

Save Data

```
public void saveMovie(MovieResult movie) {

        open();



}
```

Save Data

```
public void saveMovie(MovieResult movie) {
    try {
        open();









    } catch (SQLException e) {
        e.printStackTrace();
    }



}
```

Save Data

```
public void saveMovie(MovieResult movie) {
    try {
        open();









    } catch (SQLException e) {
        e.printStackTrace();
    }finally {

        close();
    }

}
```

Filling Content Value

```
public void saveMovie(MovieResult movie) {
    try {
        open();
        ContentValues contentValue =new ContentValues();





    } catch (SQLException e) {
        e.printStackTrace();
    }finally {

        close();
    }

}
```

Filling Content Value

```
public void saveMovie(MovieResult movie) {
    try {
        open();
        ContentValues contentValue =new ContentValues();
        contentValue.put(DatabaseHelper.COLUMN_NAME_ID, movie.getId());
        contentValue.put(DatabaseHelper.COLUMN_NAME_NAME, movie.getTitle());
        contentValue.put(DatabaseHelper.COLUMN_NAME_CONTENT, movie.getOverview());
        contentValue.put(DatabaseHelper.COLUMN_NAME_IMAGE_URI, movie.getPosterPath());
        contentValue.put(DatabaseHelper.COLUMN_NAME_RELEASE_DATE, movie.getReleaseDate());

    } catch (SQLException e) {
        e.printStackTrace();
    }finally {

        close();
    }

}
```

Inserting Values

```
public void saveMovie(MovieResult movie) {
    try {
        open();
        ContentValues contentValue =new ContentValues();
        contentValue.put(DatabaseHelper.COLUMN_NAME_ID, movie.getId());
        contentValue.put(DatabaseHelper.COLUMN_NAME_NAME, movie.getTitle());
        contentValue.put(DatabaseHelper.COLUMN_NAME_CONTENT, movie.getOverview());
        contentValue.put(DatabaseHelper.COLUMN_NAME_IMAGE_URI, movie.getPosterPath());
        contentValue.put(DatabaseHelper.COLUMN_NAME_RELEASE_DATE, movie.getReleaseDate());
        database.insert(TABLE_NAME, null, contentValue);
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {

        close();
    }

}
```

Restore Data

Restore Data

```
public ArrayList<MovieResult> getAllMovies() {

}
```

Restore Data

```
public ArrayList<MovieResult> getAllMovies() {
    open();

}
```

Creating result list

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies = new ArrayList<>();

}
```

Projection Array

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    String[] columns = new String[]{DatabaseHelper.COLUMN_NAME_ID,
            DatabaseHelper.COLUMN_NAME_NAME,
            DatabaseHelper.COLUMN_NAME_CONTENT,
            DatabaseHelper.COLUMN_NAME_IMAGE_URI,
            DatabaseHelper.COLUMN_NAME_RELEASE_DATE};

}
```

Cursor

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    String[] columns = new String[]{DatabaseHelper.COLUMN_NAME_ID,
            DatabaseHelper.COLUMN_NAME_NAME,
            DatabaseHelper.COLUMN_NAME_CONTENT,
            DatabaseHelper.COLUMN_NAME_IMAGE_URI,
            DatabaseHelper.COLUMN_NAME_RELEASE_DATE};
    Cursor cursor

}
```

Cursor

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

Query

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    String[] columns = new String[]{DatabaseHelper.COLUMN_NAME_ID,
            DatabaseHelper.COLUMN_NAME_NAME,
            DatabaseHelper.COLUMN_NAME_CONTENT,
            DatabaseHelper.COLUMN_NAME_IMAGE_URI,
            DatabaseHelper.COLUMN_NAME_RELEASE_DATE};
    Cursor cursor = database.query(
                        );
}
```

Query

query

added in API level 1

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

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

Parameters	
`table`	`String`: The table name to compile the query against.
`columns`	`String`: 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`	`String`: 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.

Query

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    String[] columns = new String[]{DatabaseHelper.COLUMN_NAME_ID,
            DatabaseHelper.COLUMN_NAME_NAME,
            DatabaseHelper.COLUMN_NAME_CONTENT,
            DatabaseHelper.COLUMN_NAME_IMAGE_URI,
            DatabaseHelper.COLUMN_NAME_RELEASE_DATE};
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null, null,
                null, null);
}
```

Iterating Over Values

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {

    }
}
```

Getting column index

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId                                                    ;
}
```

Getting column index

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
}
```

Getting column index

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);




    }
}
```

Iterating Over Cursor

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);

      while (cursor.moveToNext()) {





      }
    }
}
```

Creating MovieModel

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);

      while (cursor.moveToNext()) {
          movies.add(
                                                                  );
      }
    }
}
```

Creating MovieModel

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);

      while (cursor.moveToNext()) {
          movies.add(new MovieResult(
                                                  ,
                                                      ,
                                                          ,
                                                        ,
                                                              ));
      }
    }
}
```

Creating MovieModel

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);

      while (cursor.moveToNext()) {
          movies.add(new MovieResult(
                  cursor.getInt(columnIndexId),
                                                  ,
                                                      ,
                                                    ,
                                                        ));
      }
    }
}
```

Creating MovieModel

```
public ArrayList<MovieResult> getAllMovies() {
    ...
    if (cursor.getCount() > 0) {
      int columnIndexId = cursor.getColumnIndexOrThrow(COLUMN_NAME_ID);
      int columnIndexName = cursor.getColumnIndexOrThrow(COLUMN_NAME_NAME);
      int columnIndexContent = cursor.getColumnIndexOrThrow(COLUMN_NAME_CONTENT);
      int columnIndexImageUri = cursor.getColumnIndexOrThrow(COLUMN_NAME_IMAGE_URI);
      int columnIndexReleaseDate = cursor.getColumnIndexOrThrow(COLUMN_NAME_RELEASE_DATE);

      while (cursor.moveToNext()) {
          movies.add(new MovieResult(
                  cursor.getInt(columnIndexId),
                  cursor.getString(columnIndexName),
                  cursor.getString(columnIndexImageUri),
                  cursor.getString(columnIndexContent),
                  cursor.getString(columnIndexReleaseDate)));
      }
    }
}
```

Close cursor

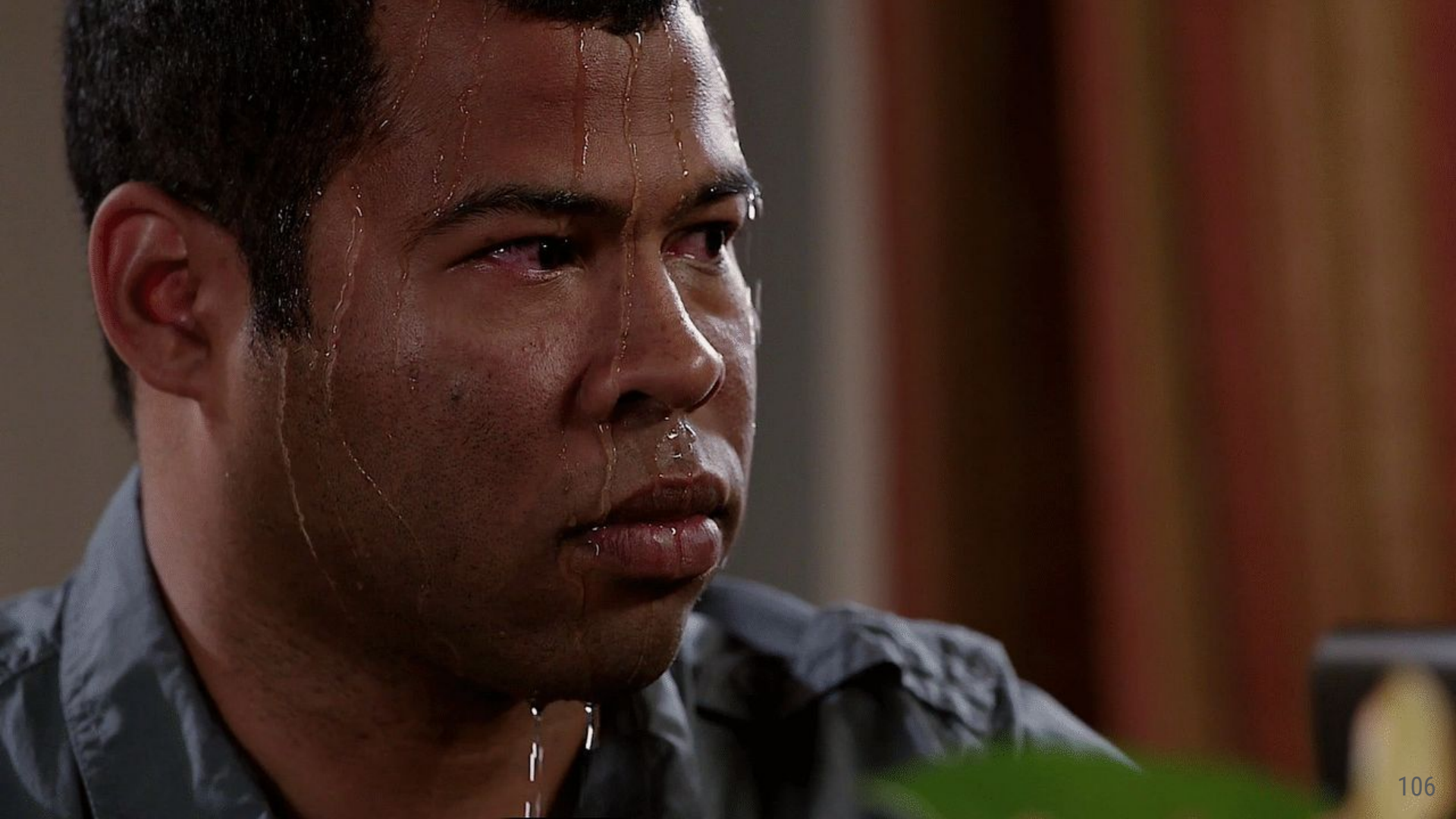
```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    ...
    ...
    cursor.close();


}
```

Close connection

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    ...
    ...
    cursor.close();
    close();

}
```

Return Data

```
public ArrayList<MovieResult> getAllMovies() {
    open();
    ArrayList<MovieResult> movies =new ArrayList<>();
    ...
    ...
    cursor.close();
    close();
    return movies;
}
```



There is a Room for improvement!

Android Architecture Components

Together or Separately.

Handling lifecycles

Create a UI that automatically responds to lifecycle events.

LiveData

Build data objects that notify views when the underlying database changes.

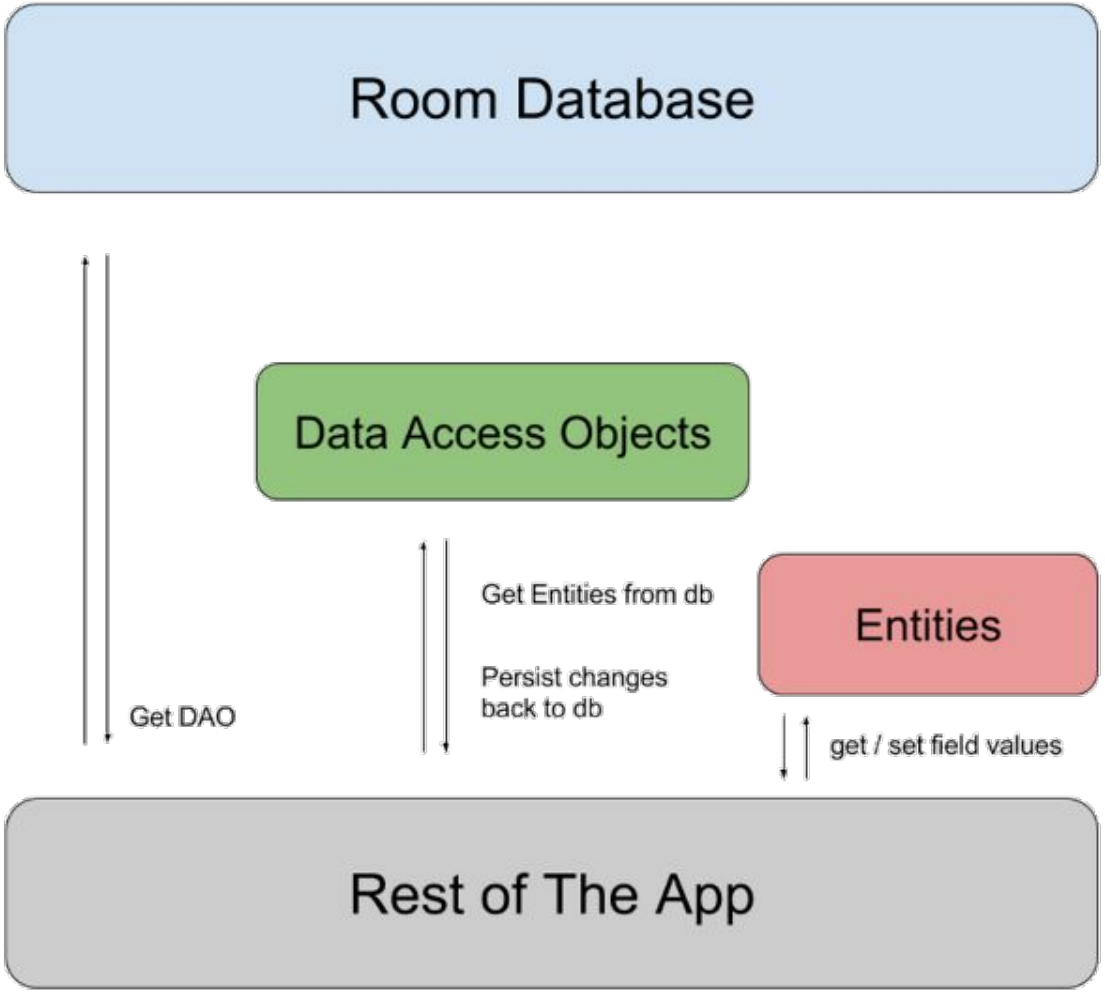
ViewModel

Store UI related data that isn't destroyed on app rotations.

Room

Access your data with the power of SQLite and safety of in-app objects.

Room Persistence Library


Room Database
Data Access Objects
Get DAO
Get Entities from db
Persist changes
back to db
Entities
get / set field values
Rest of The App

Entities -> Tables

MovieModel Entity

```
public class MovieModel{


    private int movieId;
    private String name;
    private String imageUri;
    private String backImageUri;
    private String overview;
    private String releaseDate;
    private Double popularity;

}
```

MovieModel Entity

```
public class MovieModel implements Parcelable {


    private int movieId;
    private String name;
    private String imageUri;
    private String backImageUri;
    private String overview;
    private String releaseDate;
    private Double popularity;

}
```

Entity

```
@Entity
public class MovieModel implements Parcelable {


    private int movieId;
    private String name;
    private String imageUri;
    private String backImageUri;
    private String overview;
    private String releaseDate;
    private Double popularity;

}
```

Primary Key

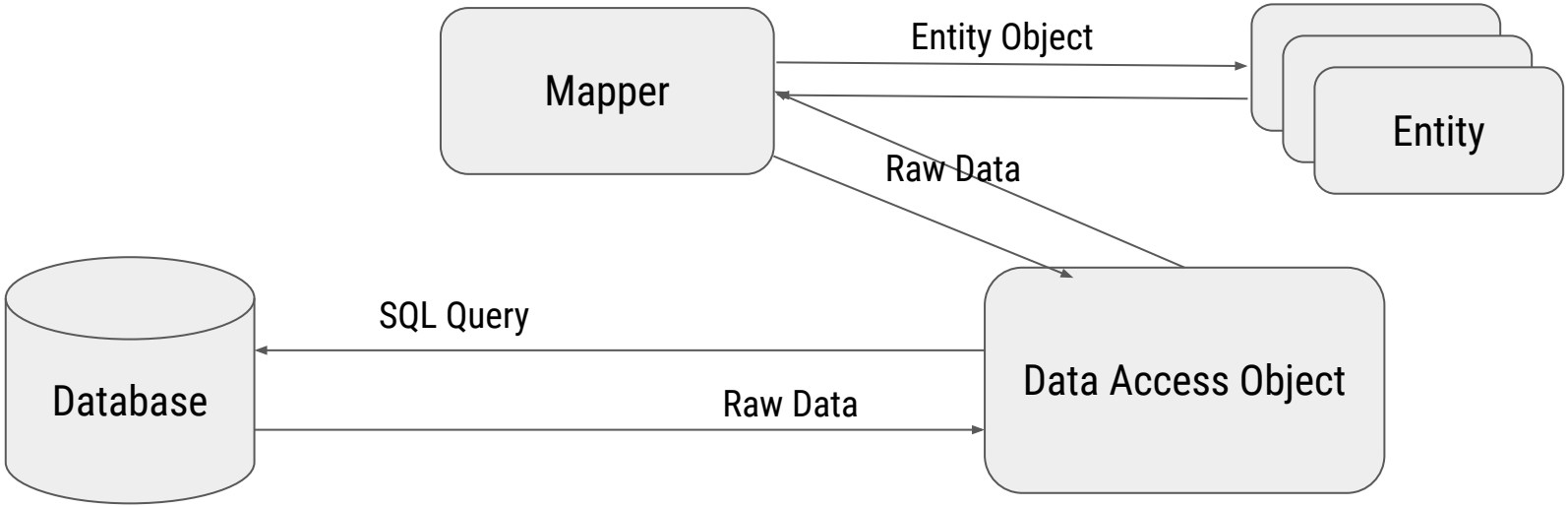
```
@Entity
public class MovieModel implements Parcelable {

    @PrimaryKey
    private int movieId;
    private String name;
    private String imageUri;
    private String backImageUri;
    private String overview;
    private String releaseDate;
    private Double popularity;

}
```

Columns

```
@Entity
public class MovieModel implements Parcelable {

    @PrimaryKey
    private int movieId;
    private String name;
    private String imageUri;
    private String backImageUri;
    private String overview;
    private String releaseDate;
    private Double popularity;

}
```

DAO - Data Access Object

Data Access Object



Data Access Object

```
@Dao
public interface MovieDao {

}
```

Raw Query - Select

```
@Dao
public interface MovieDao {

    @Query("SELECT * FROM MovieModel ORDER BY popularity DESC")
    List<MovieModel> getAll();

}
```

Insert - Strategy REPLACE

```
@Dao
public interface MovieDao {

    @Query("SELECT * FROM MovieModel ORDER BY popularity DESC")
    List<MovieModel> getAll();

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insertAll(Collection<MovieModel> movies);



}
```

Raw Query - Delete

```
@Dao
public interface MovieDao {

    @Query("SELECT * FROM MovieModel ORDER BY popularity DESC")
    List<MovieModel> getAll();

    @Insert(onConflict = OnConflictStrategy. REPLACE)
    void insertAll(Collection<MovieModel> movies);

    @Query("DELETE FROM MovieModel")
    void deleteAll();


}
```

Raw Query - Select with LIKE

```
@Dao
public interface MovieDao {


    @Query("SELECT * FROM movieModel WHERE name LIKE :name LIMIT 1")
    MovieModel findByName(String name);




}
```

Raw Query - Select with array

```
@Dao
public interface MovieDao {


    @Query("SELECT * FROM movieModel WHERE name LIKE :name LIMIT 1")
    MovieModel findByName(String name);

    @Query("SELECT * FROM movieModel WHERE movieId IN (:movieIds)")
    List<MovieModel> loadAllByIds(String[] movieIds);

}
```

Syntax Errors Highlight

```
13
14  @Dao
15  public interface MovieDao {
16
17      @Query("SELECT * FORM MovieModel ORDER BY popularity DESC")
18      List<MovieModel> getAll();
```

<compound operator>, FROM, GROUP, LIMIT, ORDER, WHERE, comma or semicolon expected, got 'FORM'

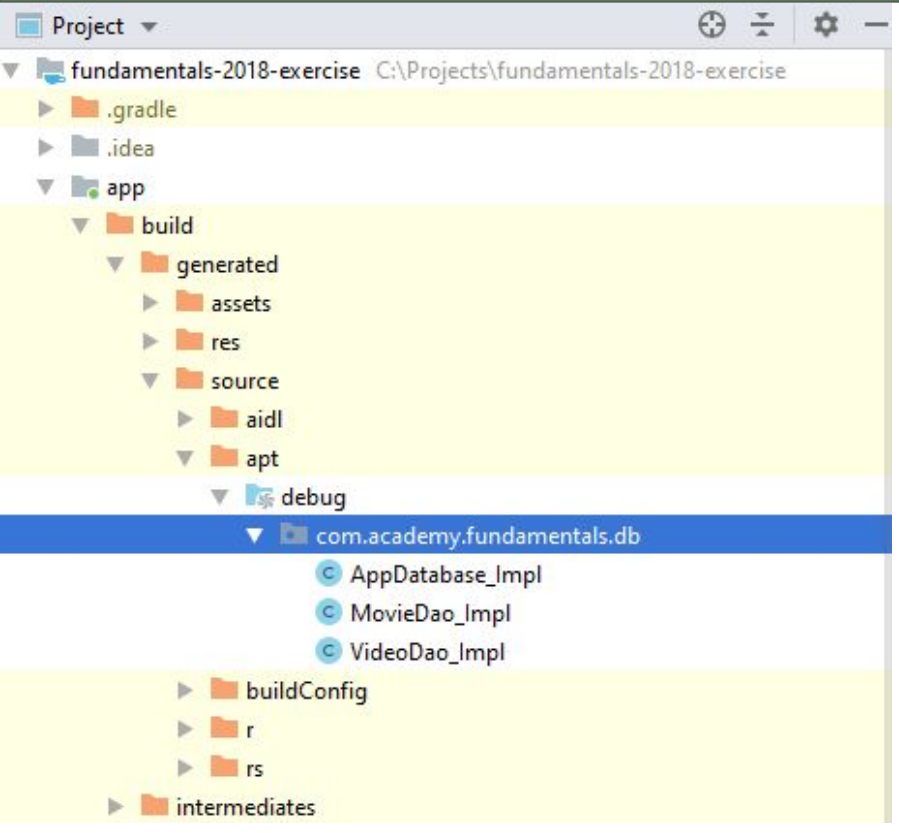
```
20      @Insert(onConflict = OnConflictStrategy.REPLACE)
21      void insertAll(Collection<MovieModel> movies);
22
23      @Query("DELETE FROM MovieModel")
24      void deleteAll();
```

Syntax Errors Highlight

```
@Dao
public interface MovieDao {

    @Query("SELECT * FROM MoveModel ORDER BY popularity DESC")
    List<MovieModel> getAll();

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insertAll(Collection<MovieModel> movies);

    @Query("DELETE FROM MovieModel")
    void deleteAll();

    @Query("SELECT * FROM movieModel WHERE name LIKE :name LIMIT 1")
    MovieModel findByName(String name);
```

Cannot resolve symbol 'MoveModel' more... (Ctrl+F1)

How The Magic Works?



How The Magic Works?

```
@Override
public void insertAll(final Collection<MovieModel> movies) {
 __db.assertNotSuspendingTransaction();
 __db.beginTransaction();
 try {
   __insertionAdapterOfMovieModel.insert(movies);
   __db.setTransactionSuccessful();
 } finally {
   __db.endTransaction();
 }
}
```

How The Magic Works?

```
@Override
public void deleteAll() {
 __db.assertNotSuspendingTransaction();
 final SupportSQLiteStatement _stmt = __preparedStmtOfDeleteAll.acquire();
 __db.beginTransaction();
 try {
   _stmt.executeUpdateDelete();
   __db.setTransactionSuccessful();
 } finally {
   __db.endTransaction();
   __preparedStmtOfDeleteAll.release(_stmt);
 }
}
```

How The Magic Works?

```
@Override
public List<MovieModel> getAll() {
 final String _sql = "SELECT * FROM MovieModel ORDER BY popularity DESC";
 final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
 __db.assertNotSuspendingTransaction();
 final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
 try {
   final int _cursorIndexOfMovieId = CursorUtil.getColumnIndexOrThrow(_cursor, "movieId");
   ...
   final List<MovieModel> _result = new ArrayList<MovieModel>(_cursor.getCount());
   while(_cursor.moveToNext()) {
     final MovieModel _item;
     final int _tmpMovieId; _tmpMovieId = _cursor.getInt(_cursorIndexOfMovieId);
     final String _tmpName; _tmpName = _cursor.getString(_cursorIndexOfName);
     ...
     _item = new MovieModel(_tmpMovieId,_tmpName,_tmpImageUrl,_tmpOverview,_tmpBackImageUrl,_tmpReleaseDate,_tmpPopularity);
     _result.add(_item);
   }
   return _result;
 } finally {
   _cursor.close();
   _statement.release();
 }
}
```

RoomDatabase

Declaring a Database

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }

}
```

Declaring a Database

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }

}
```

Declaring a Database

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }

}
```

Declaring a Database name

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
   static AppDatabase INSTANCE;
   public abstract MovieDao movieDao();
   public abstract VideoDao videoDao();
   public static AppDatabase getInstance(Context context) {
       if (INSTANCE == null) {
           INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                   AppDatabase.class, DATABASE_NAME)
                   .allowMainThreadQueries()
                   .fallbackToDestructiveMigration()
                   .build();
       }
       return INSTANCE;
   }

   public static void destroyInstance() {
       INSTANCE = null;
   }

}
```

Create singleton instance

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

   public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
   public abstract MovieDao movieDao();
   public abstract VideoDao videoDao();
   public static AppDatabase getInstance(Context context) {
       if (INSTANCE == null) {
           INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                   AppDatabase.class, DATABASE_NAME)
                   .allowMainThreadQueries()
                   .fallbackToDestructiveMigration()
                   .build();
       }
       return INSTANCE;
   }

   public static void destroyInstance() {
       INSTANCE = null;
   }

}
```

Adding DAOs

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Creating a Database

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Creating a Database

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

   public static final String DATABASE_NAME = "movies";
   static AppDatabase INSTANCE;
   public abstract MovieDao movieDao();
   public abstract VideoDao videoDao();
   public static AppDatabase getInstance(Context context) {
       if (INSTANCE == null) {
           INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                          AppDatabase. class, DATABASE_NAME)
                          .allowMainThreadQueries()
                          .fallbackToDestructiveMigration()
                          .build();
       }
       return INSTANCE;
   }
   public static void destroyInstance() {
       INSTANCE = null;
   }
}
```

Application Context

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext() ,
                                AppDatabase. class, DATABASE_NAME)
                                .allowMainThreadQueries()
                                .fallbackToDestructiveMigration()
                                .build();
        }
        return INSTANCE;
    }
    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Builder params

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                        AppDatabase.class, DATABASE_NAME)
                        .allowMainThreadQueries()
                        .fallbackToDestructiveMigration()
                        .build();
        }
        return INSTANCE;
    }
    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Access the database from the main thread

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                        AppDatabase. class, DATABASE_NAME)
                        . allowMainThreadQueries()
                        .fallbackToDestructiveMigration()
                        .build();
        }
        return INSTANCE;
    }
    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Re-create database tables

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                        AppDatabase. class, DATABASE_NAME)
                        .allowMainThreadQueries()
                        . fallbackToDestructiveMigration ()
                        .build();
        }
        return INSTANCE;
    }
    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Using Singleton

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

   public static final String DATABASE_NAME = "movies";
   static AppDatabase INSTANCE;
   public abstract MovieDao movieDao();
   public abstract VideoDao videoDao();
   public static AppDatabase getInstance(Context context) {
         if (INSTANCE == null) {
               INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                AppDatabase.class, DATABASE_NAME)
                .allowMainThreadQueries()
                .fallbackToDestructiveMigration()
                .build();
      }
         return INSTANCE;
    }
   ...

}
```

Destroy the Instance

```
@Database(entities = {MovieModel.class, VideoModel.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public static final String DATABASE_NAME = "movies";
    static AppDatabase INSTANCE;
    public abstract MovieDao movieDao();
    public abstract VideoDao videoDao();
    public static AppDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return INSTANCE;
    }

    public static void destroyInstance() {
        INSTANCE = null;
    }
}
```

Saving Data

Movies Content

```
public class MoviesContent {

    public static final ArrayList<MovieModel> MOVIES = new ArrayList<>();

    public static void clear() {
        MOVIES.clear();
    }

    public static void addMovie(MovieModel movie) {
        MOVIES.add(movie);
    }
}
```

Saving Data

```
private void loadMovies() {
    MoviesService moviesService = RestClientManager.getMovieServiceInstance();
    moviesService.getPopular().enqueue(new Callback<MovieListResult>() {
        @Override
        public void onResponse(@NonNull Call<MovieListResult> call, @NonNull Response<MovieListResult> response) {

            if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));
                adapter.setData(MoviesContent.MOVIES);
                AppDatabase.getInstance(MoviesActivity.this).movieDao().deleteAll();
                AppDatabase.getInstance(MoviesActivity.this).movieDao().insertAll(MoviesContent.MOVIES);
            }
        }

        @Override
        public void onFailure(Call<MovieListResult> call, Throwable t) {
            Toast.makeText(MoviesActivity.this, R.string.something_went_wrong_text, Toast.LENGTH_SHORT).show();

        }
    });
}
```

Saving Data - receving from network

```
private void loadMovies() {

    MoviesService moviesService = RestClientManagergetMovieServiceInstance();
    moviesService.getPopular().enqueue(new Callback<MovieListResult>() {
        @Override
        public void onResponse(@NonNull Call<MovieListResult> call, @NonNull Response<MovieListResult> response) {

            if (response.code() == 200 && response.body() != null) {

                ...

            }
        }
       @Override
        public void onFailure(Call<MovieListResult> call, Throwable t) {
            Toast.makeText(MoviesActivity.this, R.string.something_went_wrong_text, Toast.LENGTH_SHORT).show();

        }

    });
}
```

Saving Data

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager. getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();





                }
            }

    });
}
```

Saving Data

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager. getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(response.body());




                }
            }

    });
}
```

Saving Data

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager. getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));




                }
            }

    });
}
```

Movie Model Converter

```
public class MovieModelConverter {

    public static ArrayList<MovieModel> convertResult(MovieListResult movieListResult) {

        ArrayList<MovieModel> result = new ArrayList<>();
        for (MovieResult movieResult : movieListResult.getResults()) {
            result.add(new MovieModel(movieResult.getId(), movieResult.getTitle(),
                    MoviesService.POSTER_BASE_URL + movieResult.getPosterPath(),
                    MoviesService.BACKDROP_BASE_URL + movieResult.getBackdropPath(),
                    movieResult.getOverview(), movieResult.getReleaseDate(),
                    movieResult.getPopularity()));
        }

        return result;
    }
}
```

Saving Data - updating the User

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager. getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));
                adapter.setData(MoviesContent.MOVIES);



                }
            }

    });
}
```

Database Instance

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager.  getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));
                adapter.setData(MoviesContent.MOVIES);
                AppDatabase.getInstance(MoviesActivity.this)                    ;



               }
         }

    });
}
```

Saving Data - delete existing data

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager.  getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));
                adapter.setData(MoviesContent.MOVIES);
                AppDatabase.getInstance(MoviesActivity.this).movieDao().deleteAll();


               }
         }

    });
}
```

Saving Data - insert new data

```
private void loadMovies() {

    MoviesService moviesService = RestClientManager.  getMovieServiceInstance();
    moviesService.getPopular().enqueue( new Callback<MovieListResult>() {
        @Override
        public void onResponse(...) {

          if (response.code() == 200 && response.body() != null) {
                MoviesContent.clear();
                MoviesContent.MOVIES.addAll(MovieModelConverter.convertResult(response.body()));
                adapter.setData(MoviesContent.MOVIES);
                AppDatabase.getInstance(MoviesActivity.this).movieDao().deleteAll();
                AppDatabase.getInstance(MoviesActivity.this).movieDao()
                                                             .insertAll(MoviesContent.MOVIES);
               }
         }

    });
}
```

Restoring Data

Restoring Data

```
private void getCachedMoviesFromDataBase() {

    List<MovieModel> cachedMovies = AppDatabase.getInstance(this).movieDao().getAll();
    MoviesContent.MOVIES.addAll(cachedMovies);

    adapter = new MoviesViewAdapter(MoviesContent.MOVIES, MoviesActivity.this);
    recyclerView.setAdapter(adapter);
}
```

Restoring Data - load from database

```
private void getCachedMoviesFromDataBase() {

    List<MovieModel> cachedMovies = AppDatabase.getInstance(this).movieDao().getAll();



}
```

Restoring Data - adding data

```
private void getCachedMoviesFromDataBase() {

    List<MovieModel> cachedMovies = AppDatabase.getInstance(this).movieDao().getAll();
    MoviesContent.MOVIES.addAll(cachedMovies);



}
```

Restoring Data - set adapter

```
private void getCachedMoviesFromDataBase() {

    List<MovieModel> cachedMovies = AppDatabase.getInstance(this).movieDao().getAll();
    MoviesContent.MOVIES.addAll(cachedMovies);

    adapter = new MoviesViewAdapter(MoviesContent.MOVIES, MoviesActivity.this);
    recyclerView.setAdapter(adapter);
}
```

Restoring Data - notify to update UI

```
private void getCachedMoviesFromDataBase() {

    List<MovieModel> cachedMovies = AppDatabase.getInstance(this).movieDao().getAll();
    MoviesContent.MOVIES.addAll(cachedMovies);

    adapter.setData(MoviesContent.MOVIES);
    adapter.notifyDataSetChanged();

}
```

Home Work!!!

Exercise 10

The End