



Android

Author : Do Phu Quy

Email : phuquycntt@gmail.com

Phone : 0935 366 007

Room library

World before Room

- Boilerplate code

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_SUBTITLE
};

// Filter results WHERE "title" = 'My Title'
String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "My Title" };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor cursor = db.query(
    FeedEntry.TABLE_NAME,           // The table to query
    projection,                     // The columns to return
    selection,                       // The columns for the WHERE clause
    selectionArgs,                  // The values for the WHERE clause
    null,                           // don't group the rows
    null,                           // don't filter by row groups
    sortOrder                       // The sort order
);
```

World before Room

- Difficult migrations

```
override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        if (oldVersion == DATABASE_VERSION_1 && newVersion == DATABASE_VERSION_2) { upgradeSchema_DBv1_To_DBv2(db); }  
        else if (oldVersion == DATABASE_VERSION_1 && newVersion == DATABASE_VERSION_3) { upgradeSchema_DBv1_to_DBv3(db); }  
        else if (oldVersion == DATABASE_VERSION_1 && newVersion == DATABASE_VERSION_4) { upgradeSchema_DBv1_to_DBv4(db); }  
        else if (oldVersion == DATABASE_VERSION_1 && newVersion == DATABASE_VERSION_5) { upgradeSchema_DBv1_To_DBv5(db); }  
        else if (oldVersion == DATABASE_VERSION_2 && newVersion == DATABASE_VERSION_3) { upgradeSchema_DBv2_To_DBv3(db); }  
        else if (oldVersion == DATABASE_VERSION_2 && newVersion == DATABASE_VERSION_4) { upgradeSchema_DBv2_To_DBv4(db); }  
        else if (oldVersion == DATABASE_VERSION_2 && newVersion == DATABASE_VERSION_5) { upgradeSchema_DBv2_To_DBv5(db); }  
        else if (oldVersion == DATABASE_VERSION_3 && newVersion == DATABASE_VERSION_4) { upgradeSchema_DBv3_To_DBv4(db); }  
        else if (oldVersion == DATABASE_VERSION_3 && newVersion == DATABASE_VERSION_5) { upgradeSchema_DBv3_To_DBv5(db); }  
        else if (oldVersion == DATABASE_VERSION_4 && newVersion == DATABASE_VERSION_5) { upgradeSchema_DBv4_To_DBv5(db); }  
        fillData(db);  
    }
```

Solutions

3d-party ORM libraries over SQLite like:

- **DBFlow**
- **Requery**
- **GreenDAO**
- **And so on, many of them!**

Solutions

...OR even replacement for SQLite like



Google I/O 2017

Android Architecture Components

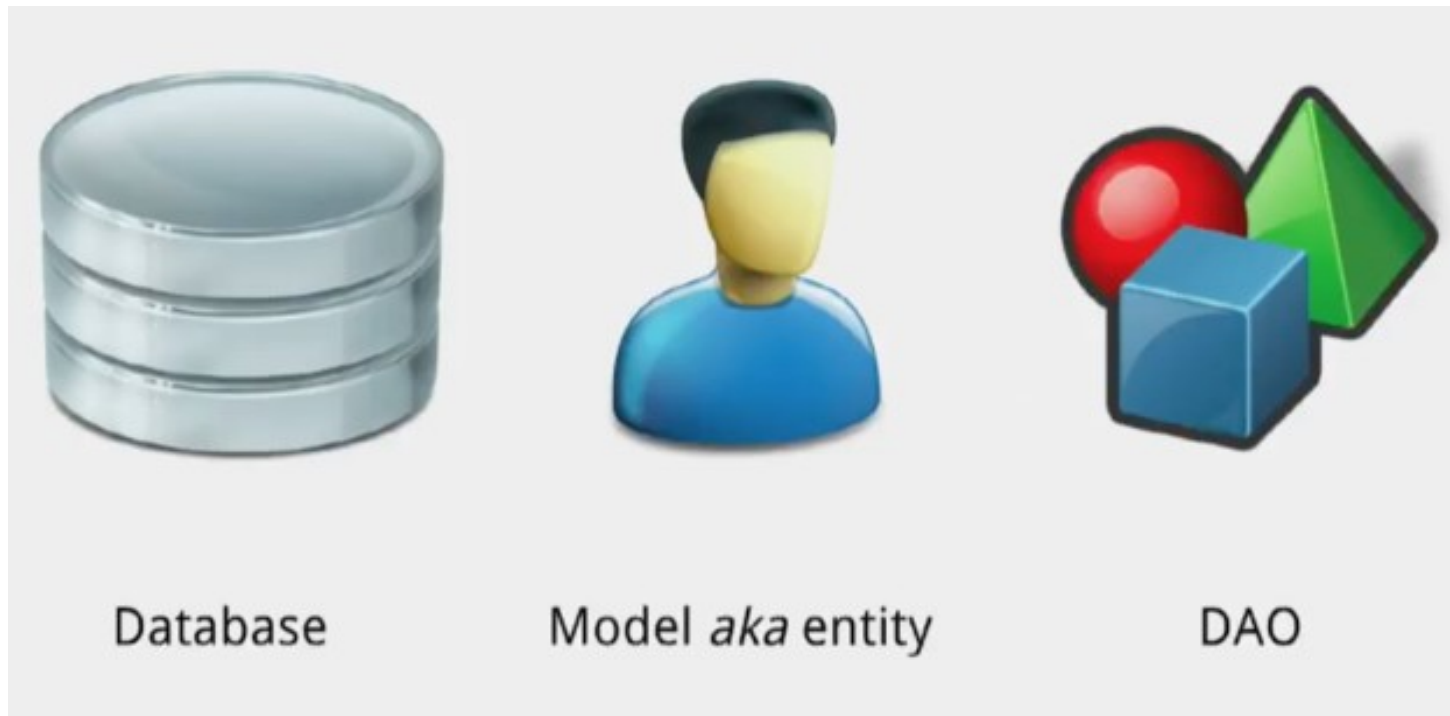


Adding to project

Release notes: 1.0.0 - November 6, 2017

- For [Room](#), add:
 - `implementation "android.arch.persistence.room:runtime:1.0.0"`
 - `annotationProcessor "android.arch.persistence.room:compiler:1.0.0"`
 - For [testing Room migrations](#), add:
 - `testImplementation "android.arch.persistence.room:testing:1.0.0"`
 - For [Room RxJava](#) support, add:
 - `implementation "android.arch.persistence.room:rxjava2:1.0.0"`

Room structure



Entity

```
public class Employee {  
    private String id;  
    private String lastName;  
}
```



Model *aka* entity

Entity

```
@Entity(tableName = "employees")
public class Employee {

    @NonNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "last_name", index = true)
    private String lastName;

    @Ignore
    private int someNotPersistedField;

    private transient int anotherSomeNotPersistedField;
```

Note: need getters & setters, not support for Lombok, AutoValue

Another Entity

```
@Entity(tableName = "departments")
public class Department {

    @NonNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "name")
    private String name;
}
```

Relation

Object references - not supported

```
@Entity(tableName = "departments")
public class Department {

    @NonNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "name")
    private String name;

    @OneToMany
    List<Employee> employees;

}
```

From doc - **Key takeaway:** Room disallows object references between entity classes. Instead, you must explicitly request the data that your app needs.

Relation

```
@Entity(tableName = "employees",
    foreignKeys = @ForeignKey(entity = Department.class,
        parentColumns = "id",
        childColumns = "department_id", onDelete = CASCADE))
public class Employee {

    @NotNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "department_id")
    private String departmentId;
```

Embedding

```
@Entity(tableName = "employees")
public class Employee {

    @NonNull
    @PrimaryKey
    private String id;

    @Embedded
    private EmployeeInfo employeeInfo;
```

```
public class EmployeeInfo {
    @ColumnInfo(name = "salary")
    private String salary;
```

Data types

```
@Entity(tableName = "employees")  
public class Employee {  
  
    @ColumnInfo(name = "hire_date")  
    private Date hireDate;  
}
```

Date - unknown type, requires converting

Data types

```
public class Converters {  
    @TypeConverter  
    public static Date fromTimestamp(Long value) {  
        return value == null ? null : new Date(value);  
    }  
  
    @TypeConverter  
    public static Long dateToTimestamp(Date date) {  
        return date == null ? null : date.getTime();  
    }  
}
```

Should be added to database object

```
@TypeConverters({Converters.class})  
public abstract class EmployeesDatabase extends RoomDatabase {
```

Database object

```
@Database(entities = {Employee.class, Department.class},  
          version = 1)  
@TypeConverters({Converters.class})  
public abstract class EmployeesDatabase extends RoomDatabase {
```



Database object

```
@Database(entities = {Employee.class, Department.class},
        version = 1)
@TypeConverters({Converters.class})
public abstract class EmployeesDatabase extends RoomDatabase {

    public abstract EmployeesDao employeesDao();

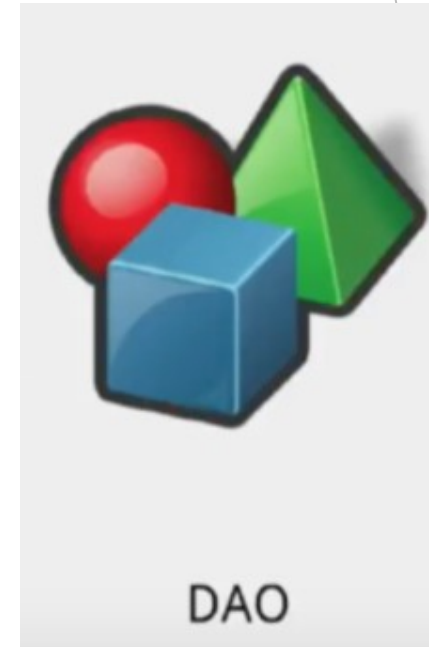
    private static volatile EmployeesDatabase INSTANCE;

    public static EmployeesDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            synchronized (EmployeesDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(
                        context.getApplicationContext(),
                        EmployeesDatabase.class,
                        "Employees.db")
                        .build();
                }
            }
        }
        return INSTANCE;
    }
}
```

DAO

```
public abstract EmployeesDao employeesDao();
```

- QUERY
- INSERT
- UPDATE
- DELETE



DAO

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees")
    List<Employee> getEmployees();

}
```

- Need to write SQL queries, not typical for ORMs
- Retrofit style

DAO

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employes")
    List<Employee> getEmployees();

}
```



! error: There is a problem with the query: [SQLITE_ERROR] SQL error or missing database (no such table: employes)

DAO - code generation

```
@Generated("android.arch.persistence.room.RoomProcessor")  
public class EmployeesDao_Impl implements EmployeesDao {  
    private final RoomDatabase __db;
```

DAO - code generation

```
@Generated("android.arch.persistence.room.RoomProcessor")  
public class EmployeesDao_Impl implements EmployeesDao {  
    private final RoomDatabase __db;
```

```
    @Override  
    public void insertEmployee(Employee employee) {  
        __db.beginTransaction();  
        try {  
            __insertionAdapterOfEmployee.insert(employee);  
            __db.setTransactionSuccessful();  
        } finally {  
            __db.endTransaction();  
        }  
    }  
}
```


DAO - Threading

- QUERY
- INSERT
- UPDATE
- DELETE

All Synchronous

DAO - Threading

- QUERY
- INSERT
- UPDATE
- DELETE

All Synchronous

Caused by: `java.lang.IllegalStateException:`
`Cannot` access database on the main thread since it may
potentially `lock` the UI for a `long` periods of time.

DAO - Threading

Quick, but not good fix

```
if (INSTANCE == null) {  
    INSTANCE = Room.databaseBuilder(  
        context.getApplicationContext(),  
        EmployeesDatabase.class,  
        name: "Employees.db")  
        .allowMainThreadQueries()  
        .build();  
}
```



DAO - LiveData

```
@dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees WHERE id = :id")
    LiveData<Employee> getEmployee(String id);
}
```

DAO - RxJava

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees WHERE id = :id")
    Maybe<Employee> getEmployee(String id);
}
```

DAO - RxJava

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees WHERE id = :id")
    Maybe<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Single<Employee> getEmployee(String id);
}
```

DAO - RxJava

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees WHERE id = :id")
    Maybe<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Single<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Flowable<Employee> getEmployee(String id);
}
```

DAO - RxJava

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM employees WHERE id = :id")
    Maybe<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Single<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Flowable<Employee> getEmployee(String id);

    @Query("SELECT * FROM employees WHERE id = :id")
    Flowable<List<Employee>> getEmployee(String id);
}
```


DAO - Query, Joins

From official documentation:

```
@Da
public interface MyDao
    @Query("SELECT * FROM book "
        + "INNER JOIN loan ON loan.book_id = book.id "
        + "INNER JOIN user ON user.id = loan.user_id "
        + "WHERE user.name LIKE :userName")
    public List<Book> findBooksBorrowedByNameSync(String userName);
}
```

Another way:

DAO -

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM departments WHERE id = :id")
    Flowable<DepartmentAndEmployeesInfo> getDepartment(String id);
```

Another way:

DAO -

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM departments WHERE id = :id")
    Flowable<DepartmentAndEmployeesInfo> getDepartment(String id);
```

```
public class DepartmentAndEmployeesInfo {

    @Embedded
    private Department department;

    @Relation(parentColumn = "id",entityColumn = "department_id")
    private List<Employee> employees;
```

Another way:

DAO -

```
@Dao
public interface EmployeesDao {

    @Query("SELECT * FROM departments WHERE id = :id")
    Flowable<DepartmentAndEmployeesInfo> getDepartment(String id);
```

```
public class DepartmentAndEmployeesInfo {

    @Embedded
    private Department department;

    @Relation(parentColumn = "id", entityColumn = "department_id")
    private List<Employee> employees;
```

Testing

Testing

```
@RunWith(AndroidJUnit4.class)
public class EmployeeDaoTest {

    private EmployeesDatabase database;

    private EmployeesDao employeesDao;
```

Testing

```
@RunWith(AndroidJUnit4.class)
public class EmployeeDaoTest {

    private EmployeesDatabase database;

    private EmployeesDao employeesDao;
```

```
@Rule
public InstantTaskExecutorRule instantTaskExecutorRule = new InstantTaskExecutorRule();
```

Testing

```
@RunWith(AndroidJUnit4.class)
public class EmployeeDaoTest {

    private EmployeesDatabase database;

    private EmployeeDao employeeDao;
```

```
@Rule
public InstantTaskExecutorRule instantTaskExecutorRule = new InstantTaskExecutorRule();
```

```
@Before
public void initDb() throws Exception {
    database = Room.inMemoryDatabaseBuilder(InstrumentationRegistry.getContext(),
        EmployeesDatabase.class)
        .allowMainThreadQueries()
        .build();
    employeeDao = database.employeeDao();
}
```

```
@After
public void closeDb() throws Exception {
    database.close();
}
```


Testing

RxJava's Flowable

```
private static final Employee EMPLOYEE
    = new Employee( id: "id", lastName: "Dogar");

@Test
public void insertAndGetUser() {
    database.employeesDao().insertEmployee(EMPLOYEE);

    database.employeesDao()
        .getEmployees()
        .test()
        .assertValue(employee -> {
            return employee != null
                && employee.getId().equals(EMPLOYEE.getId())
                && employee.getLastName().equals(EMPLOYEE.getLastName());
        });
}
```

Similar with LiveData

Testing

More real-life example UnitTest

```
public class LocalEmployeeRepository implements EmployeeRepository {  
    private final EmployeesDao employeesDao;  
  
    public LocalEmployeeRepository(EmployeesDao employeesDao) {  
        this.employeesDao = employeesDao;  
    }  
}
```

Testing

More real-life example UnitTest

```
public class EmployeeRepositoryTest {  
    @Mock  
    EmployeesDao mockedDao;  
  
    private EmployeeRepository employeeRepository;  
  
    @Before  
    public void setUp() throws Exception {  
        MockitoAnnotations.initMocks( testClass: this);  
  
        employeeRepository = new LocalEmployeeRepository(mockedDao);  
    }  
}
```

Migrations

Migrations

```
@Entity(tableName = "employees")
public class Employee {

    @NonNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "last_name", index = true)
    private String lastName;
```

Migrations

```
@Entity(tableName = "employees")
public class Employee {

    @NonNull
    @PrimaryKey
    private String id;

    @ColumnInfo(name = "last_name", index = true)
    private String lastName;
```

```
@ColumnInfo(name = "new_field")
private Integer newField;
```



Migrations

Update version

```
@Database(entities = {Employee.class, Department.class},  
          version = 1)  
@TypeConverters({Converters.class})  
public abstract class EmployeesDatabase extends RoomDatabase {
```

Migrations

Update version

```
@Database(entities = {Employee.class, Department.class},  
          version = 1)  
@TypeConverters({Converters.class})  
public abstract class EmployeesDatabase extends RoomDatabase {
```

```
@Database(entities = {Employee.class, Department.class},  
          version = 2)  
@TypeConverters({Converters.class})  
public abstract class EmployeesDatabase extends RoomDatabase {
```


Migrations

Provide migration

```
private static final Migration MIGRATION_1_2 = new Migration(1, 2) {  
    @Override  
    public void migrate(@NonNull SupportSQLiteDatabase database) {  
        database.execSQL("ALTER TABLE `employees`" +  
            " ADD COLUMN `new_field` INTEGER");  
    }  
};
```

Migrations

Provide migration

```
private static final Migration MIGRATION_1_2 = new Migration(1, 2) {  
    @Override  
    public void migrate(@NonNull SupportSQLiteDatabase database) {  
        database.execSQL("ALTER TABLE `employees` +  
            " ADD COLUMN `new_field` INTEGER");  
    }  
};
```

Add it to db object

```
INSTANCE = Room.databaseBuilder(  
    context.getApplicationContext(),  
    EmployeesDatabase.class,  
    name: "Employees.db")  
    .addMigrations(MIGRATION_1_2)  
    .build();
```

Migrations

Migrations execution

```
INSTANCE = Room.databaseBuilder(  
    context.getApplicationContext(),  
    EmployeesDatabase.class,  
    name: "Employees.db")  
    .addMigrations(  
        MIGRATION_1_2,  
        MIGRATION_2_3,  
        MIGRATION_3_4,  
        MIGRATION_2_4)  
    .build();
```

Migrations

Migrations execution

```
INSTANCE = Room.databaseBuilder(  
    context.getApplicationContext(),  
    EmployeesDatabase.class,  
    name: "Employees.db")  
    .addMigrations(  
        MIGRATION_1_2,  
        MIGRATION_2_3,  
        MIGRATION_3_4,  
        MIGRATION_2_4)  
    .build();
```

Shortest path

Migrations Testing

Migrations Testing

Setup

- Turn on schema export (as json file)

```
{  
  "formatVersion": 1,  
  "database": {  
    "version": 2,  
    "identityHash": "ed3c6326ed4ff343d28ae24912fc9316",  
    "entities": [  
      {  
        "tableName": "employees",  
        "createSql": "CREATE TABLE IF NOT EXISTS `${TAB  
        "fields": [  
          {  
            "fieldPath": "id",  
            "columnName": "id",
```

- In build.gradle: Set schema location to tests
- Export schema for annotation processor

Migrations Testing

```
@RunWith(AndroidJUnit4.class)
public class MigrationTest {
    private static final String TEST_DB_NAME = "test_employee_db";

    @Rule
    public MigrationTestHelper helper;

    public MigrationTest() {
        helper = new MigrationTestHelper(InstrumentationRegistry.getInstrumentation(),
            EmployeesDatabase.class.getCanonicalName(),
            new FrameworkSQLiteOpenHelperFactory());
    }
}
```

Migrations Testing

```
@Test
public void migrate1To2() throws IOException {
    SupportSQLiteDatabase db = helper.createDatabase(TEST_DB_NAME, 1);
    insertEmployee(db); // need to insert with ContentValues SQLite API
    helper.runMigrationsAndValidate(TEST_DB_NAME, 2, true, MIGRATION_1_2);
}
```

And then build db object, get Data from DAO and write asserts

Summary

- Removes ugly boilerplate low level APIs code
- Nice testing and RxJava2 support
- Google support
- Use it if you need it

Links

Florina Muntenescu blogs

- <https://medium.com/@florina.muntenescu>

Aleksander Piotrowski, Talk at GDGDevFest 2017 Is there a room for Room?

- <https://www.youtube.com/watch?v=BHiKSnOaoh4>

Florina Muntenescu talk at realm.io

- <https://goo.gl/dNYY3S>

And official doc

- <https://developer.android.com/training/data-storage/room/index.html>