[Урок 5 Room. Основы](https://startandroid.ru/ru/blog/27-course/architecture-components/529-urok-5-room-osnovy.html" \o "Урок 5. Room. Основы)

09 января 2018

Библиотека Room предоставляет нам удобную обертку для работы с базой данных SQLite. В этом уроке рассмотрим основы. Как подключить к проекту. Как получать, вставлять, обновлять и удалять данные.

Полный список уроков курса:

* [Урок 1. Lifecycle](https://startandroid.ru/ru/blog/27-course/architecture-components/524-urok-1.html)
* [Урок 2. LiveData](https://startandroid.ru/ru/blog/27-course/architecture-components/525-urok-2-livedata.html)
* [Урок 3. LiveData. Дополнительные возможности](https://startandroid.ru/ru/blog/27-course/architecture-components/526-urok-3-livedata.html)
* [Урок 4. ViewModel](https://startandroid.ru/ru/blog/27-course/architecture-components/527-urok-4-viewmodel.html)
* [Урок 5. Room. Основы](https://startandroid.ru/ru/blog/27-course/architecture-components/529-urok-5-room-osnovy.html)
* [Урок 6. Room. Entity](https://startandroid.ru/ru/blog/27-course/architecture-components/530-urok-6-room-entity.html)
* [Урок 7. Room. Insert, Update, Delete, Transaction](https://startandroid.ru/ru/blog/27-course/architecture-components/531-urok-7-room-insert-update-delete-transaction.html)
* [Урок 8. Room. Query](https://startandroid.ru/ru/blog/27-course/architecture-components/532-urok-8-room-query.html)
* [Урок 9. Room. RxJava](https://startandroid.ru/ru/blog/27-course/architecture-components/533-urok-9-room-rxjava.html)
* [Урок 10. Room. Запрос из нескольких таблиц. Relation](https://startandroid.ru/ru/blog/27-course/architecture-components/534-urok-10-room-zapros-iz-neskolkih-tablic-relation.html)
* [Урок 11. Room. Type converter](https://startandroid.ru/ru/blog/27-course/architecture-components/539-urok-11-room-type-converter.html)
* [Урок 12. Room. Миграция версий базы данных](https://startandroid.ru/ru/blog/27-course/architecture-components/540-urok-12-migracija-versij-bazy-dannyh.html)
* [Урок 13. Room. Тестирование](https://startandroid.ru/ru/blog/27-course/architecture-components/541-urok-13-room-testirovanie.html)
* [Урок 14. Paging Library. Основы](https://startandroid.ru/ru/blog/27-course/architecture-components/542-urok-14-paging-library-chast-1.html)
* [Урок 15. Paging Library. PagedList и DataSource. Placeholders.](https://startandroid.ru/ru/blog/27-course/architecture-components/543-urok-15-paging-library-chast-2.html)
* [Урок 16. Paging Library. LivePagedListBuilder. BoundaryCallback.](https://startandroid.ru/ru/blog/27-course/architecture-components/544-urok-15-paging-library-chast-2-pagedlist-i-datasource.html)
* [Урок 17. Paging Library. Виды DataSource](https://startandroid.ru/ru/blog/27-course/architecture-components/545-urok-17-paging-library-vidy-datasource.html)
* [Урок 18. Android Data Binding. Основы](https://startandroid.ru/ru/blog/27-course/architecture-components/551-urok-18-data-binding-osnovy.html)
* [Урок 19. Android Data Binding. Код в layout. Доступ к View](https://startandroid.ru/ru/blog/27-course/architecture-components/552-urok-19-android-data-binding-vozmozhnosti.html)
* [Урок 20. Android Data Binding. Обработка событий](https://startandroid.ru/ru/blog/27-course/architecture-components/553-urok-20-android-data-binding-obrabotka-sobytiy.html)
* [Урок 21. Android Data Binding. Observable поля. Двусторонний биндинг.](https://startandroid.ru/ru/blog/27-course/architecture-components/554-urok-21-android-data-binding-observable.html)
* [Урок 22. Android Data Binding. Adapter. Conversion.](https://startandroid.ru/ru/blog/27-course/architecture-components/555-urok-22-android-data-binding-include-viewstub-holders.html)
* [Урок 23. Android Data Binding. Использование с include, ViewStub и RecyclerView.](https://startandroid.ru/ru/blog/27-course/architecture-components/556-urok-23-android-data-binding.html)
* [Урок 24. Navigation Architecture Component. Введение](https://startandroid.ru/ru/blog/27-course/architecture-components/557-urok-24-android-navigation-component-vvedenie.html)
* [Урок 25. Navigation. Передача данных. Type-safe аргументы.](https://startandroid.ru/ru/blog/27-course/architecture-components/558-urok-25-navigation-peredacha-dannyh-typesafe-argumenty.html)
* [Урок 26. Navigation. Параметры навигации](https://startandroid.ru/ru/blog/27-course/architecture-components/559-urok-26-navigation-parametry-navigacii.html)
* [Урок 27. Navigation. NavigationUI.](https://startandroid.ru/ru/blog/27-course/architecture-components/560-urok-27-navigation-navigationui.html)
* [Урок 28. Navigation. Вложенный граф. Global Action. Deep Link.](https://startandroid.ru/ru/blog/27-course/architecture-components/561-urok-28-navigation-vlozhennyj-graf.html)
* [Урок 29. WorkManager. Введение](https://startandroid.ru/ru/blog/27-course/architecture-components/562-urok-29-workmanager-vvedenie.html)
* [Урок 30. WorkManager. Критерии запуска задачи.](https://startandroid.ru/ru/blog/27-course/architecture-components/563-urok-30-workmanager-constraints.html)
* [Урок 31. WorkManager. Последовательность выполнения задач.](https://startandroid.ru/ru/blog/27-course/architecture-components/564-urok-31-workmanager-posledovatelnosti-vypolnenija-zadach.html)
* [Урок 32. WorkManager. Передача и получение данных](https://startandroid.ru/ru/blog/27-course/architecture-components/565-urok-32-workmanager-peredacha-i-poluchenie-dannyh.html)
* [Урок 33. Практика. О чем это будет.](https://startandroid.ru/ru/blog/27-course/architecture-components/566-urok-33-praktika-o-chem-eto-budet.html)
* [Урок 34. Практика. TodoApp. Список задач.](https://startandroid.ru/ru/blog/27-course/architecture-components/567-urok-34-praktika-todoapp-tasks.html)
* [Урок 35. Практика. TodoApp. Просмотр задачи](https://startandroid.ru/ru/blog/27-course/architecture-components/569-urok-35-praktika-todoapp-taskdetail.html)

Подключение к проекту

В build.gradle файл проекта добавьте репозитарий google()

|  |
| --- |
| allprojects {      repositories {          jcenter()          google()      }      ...  } |

В build.gradle файле модуля добавьте dependencies:

|  |
| --- |
| dependencies {      implementation "android.arch.persistence.room:runtime:1.0.0"      annotationProcessor "android.arch.persistence.room:compiler:1.0.0"      ...  } |

Если у вас студия ниже 3.0 и старые версии Gradle и Android Plugin, то подключение будет выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | buildscript {      repositories {          jcenter()          maven { url 'https://maven.google.com' }      }      ...  } |

и так:

|  |  |
| --- | --- |
| 1  2  3  4  5 | dependencies {      compile "android.arch.persistence.room:runtime:1.0.0"      annotationProcessor "android.arch.persistence.room:compiler:1.0.0"      ...  } |

Room

Room имеет три основных компонента: Entity, Dao и Database. Рассмотрим их на небольшом примере, в котором будем создавать базу данных для хранения данных по сотрудникам (англ. - employee).

При работе с Room нам необходимо будет писать SQL запросы. Если вы не знакомы с ними, то имеет смысл прочесть хотя бы основы.

Entity

Аннотацией [Entity](https://developer.android.com/reference/android/arch/persistence/room/Entity.html" \t "_blank) нам необходимо пометить объект, который мы хотим хранить в базе данных. Для этого создаем класс Employee, который будет представлять собой данные сотрудника: id, имя, зарплата:

|  |
| --- |
| @Entity  public class Employee {       @PrimaryKey     public long id;       public String name;       public int salary;  } |

Класс помечается аннотацией Entity. Объекты класса Employee будут использоваться при работе с базой данных. Например, мы будем получать их от базы при запросах данных и отправлять их в базу при вставке данных.

Этот же класс Employee будет использован для создания таблицы в базе. В качестве имени таблицы будет использовано имя класса. А поля таблицы будут созданы в соответствии с полями класса.

Аннотацией [PrimaryKey](https://developer.android.com/reference/android/arch/persistence/room/PrimaryKey.html" \t "_blank) мы помечаем поле, которое будет ключом в таблице.

В следующих уроках мы рассмотрим возможности Entity более подробно.

Dao

В объекте [Dao](https://developer.android.com/reference/android/arch/persistence/room/Dao.html" \t "_blank) мы будем описывать методы для работы с базой данных. Нам нужны будут методы для получения списка сотрудников и для добавления/изменения/удаления сотрудников.

Описываем их в интерфейсе с аннотацией Dao.

|  |
| --- |
| @Dao  public interface EmployeeDao {       @Query("SELECT \* FROM employee")     List<Employee> getAll();       @Query("SELECT \* FROM employee WHERE id = :id")     Employee getById(long id);       @Insert     void insert(Employee employee);       @Update     void update(Employee employee);       @Delete     void delete(Employee employee);    } |

Методы getAll и getById позволяют получить полный список сотрудников или конкретного сотрудника по id. В аннотации [Query](https://developer.android.com/reference/android/arch/persistence/room/Query.html" \t "_blank)нам необходимо прописать соответствующие SQL-запросы, которые будут использованы для получения данных.

Обратите внимание, что в качестве имени таблицы мы используем employee. Напомню, что имя таблицы равно имени Entity класса, т.е. Employee, но в SQLite не важен регистр в именах таблиц, поэтому можем писать employee.

Для вставки/обновления/удаления используются методы insert/update/delete с соответствующими аннотациями. Тут никакие запросы указывать не нужно. Названия методов могут быть любыми. Главное - аннотации.

В следующих уроках мы рассмотрим возможности Dao и его аннотаций более подробно.

Database

Аннотацией [Database](https://developer.android.com/reference/android/arch/persistence/room/Database.html" \t "_blank) помечаем основной класс по работе с базой данных. Этот класс должен быть абстрактным и наследовать RoomDatabase.

|  |  |
| --- | --- |
| 1  2  3  4 | @Database(entities = {Employee.class}, version = 1)  public abstract class AppDatabase extends RoomDatabase {     public abstract EmployeeDao employeeDao();  } |

В параметрах аннотации Database указываем, какие Entity будут использоваться, и версию базы. Для каждого Entity класса из списка entities будет создана таблица.

В Database классе необходимо описать абстрактные методы для получения Dao объектов, которые вам понадобятся.

Практика

Все необходимые для работы объекты созданы. Давайте посмотрим, как использовать их для работы с базой данных.

Database объект - это стартовая точка. Его создание выглядит так:

|  |
| --- |
| AppDatabase db =  Room.databaseBuilder(getApplicationContext(),         AppDatabase.class, "database").build(); |

Используем Application Context, а также указываем AppDatabase класс и имя файла для базы.

Учитывайте, что при вызове этого кода Room каждый раз будет создавать новый экземпляр AppDatabase. Эти экземпляры очень тяжелые и рекомендуется использовать один экземпляр для всех ваших операций. Поэтому вам необходимо позаботиться о синглтоне для этого объекта. Это можно сделать с помощью [Dagger](https://startandroid.ru/ru/blog/16-course/dagger2/424-urok-1.html), например.

Если вы не используете Dagger (или другой DI механизм), то можно использовать Application класс для создания и хранения AppDatabase:

|  |
| --- |
| public class App extends Application {        public static App instance;        private AppDatabase database;        @Override      public void onCreate() {          super.onCreate();          instance = this;          database = Room.databaseBuilder(this, AppDatabase.class, "database")                  .build();      }        public static App getInstance() {          return instance;      }        public AppDatabase getDatabase() {          return database;      }  } |

*Не забудьте добавить App класс в манифест*

В коде получение базы будет выглядеть так:

|  |  |
| --- | --- |
|  | AppDatabase db = App.getInstance().getDatabase(); |

Из Database объекта получаем Dao.

|  |
| --- |
| EmployeeDao employeeDao = db.employeeDao(); |

Теперь мы можем работать с Employee объектами. Но эти операции должны выполняться не в UI потоке. Иначе мы получим Exception.

Добавление нового сотрудника в базу будет выглядеть так:

|  |
| --- |
| Employee employee = new Employee();  employee.id = 1;  employee.name = "John Smith";  employee.salary = 10000;    employeeDao.insert(employee); |

Метод getAll вернет нам всех сотрудников в List<Employee>

|  |  |
| --- | --- |
| 1 | List<Employee> employees = employeeDao.getAll(); |

Получение сотрудника по id:

|  |  |
| --- | --- |
| 1 | Employee employee = employeeDao.getById(1); |

Обновление данных по сотруднику.

|  |  |
| --- | --- |
| 1  2 | employee.salary = 20000;  employeeDao.update(employee); |

Room будет искать в таблице запись по ключевому полю, т.е. по id. Если в объекте employee не заполнено поле id, то по умолчанию в нашем примере оно будет равно нулю и Room просто не найдет такого сотрудника (если, конечно, у вас нет записи с id = 0).

Удаление сотрудника

|  |  |
| --- | --- |
| 1 | employeeDao.delete(employee); |

Аналогично обновлению, Room будет искать запись по ключевому полю, т.е. по id

Давайте для примера добавим еще один тип объекта - Car.

Описываем Entity объект

|  |
| --- |
| @Entity  public class Car {       @PrimaryKey     public long id;       public String model;       public int year;    } |

Теперь Dao для работы с Car объектом

|  |
| --- |
| @Dao  public interface CarDao {       @Query("SELECT \* FROM car")     List<Car> getAll();       @Insert     void insert(Car car);       @Delete     void delete(Car car);    } |

Будем считать, что нам надо только читать все записи, добавлять новые и удалять старые.

В Database необходимо добавить Car в список entities и новый метод для получения CarDao

|  |
| --- |
| @Database(entities = {Employee.class, Car.class}, version = 1)  public abstract class AppDatabase extends RoomDatabase {     public abstract EmployeeDao employeeDao();     public abstract CarDao carDao();  } |

Т.к. мы добавили новую таблицу, изменилась структура базы данных. И нам необходимо поднять версию базы данных до 2. Но об этом мы подробно поговорим в Уроке 12. А пока можно оставить версию равной 1, удалить старую версию приложения и поставить новую.

UI поток

Повторюсь, операции по работе с базой данных - синхронные, и должны выполняться не в UI потоке.

В случае с Query операциями мы можем сделать их асинхронными используя [LiveData](https://startandroid.ru/ru/blog/27-course/architecture-components/525-urok-2-livedata.html) или [RxJava](https://startandroid.ru/ru/blog/19-course/rxjava/435-urok-1.html). Об этом еще поговорим в следующих уроках.

В случае insert/update/delete вы можете обернуть эти методы в асинхронный RxJava. В моем блоге есть [статья](https://startandroid.ru/ru/blog/492-kak-obernut-metod-v-rxjava.html) на эту тему.

Также, вы можете использовать allowMainThreadQueries в билдере создания AppDatabase

|  |
| --- |
| AppDatabase db =  Room.databaseBuilder(getApplicationContext(),         AppDatabase.class, "database")         .allowMainThreadQueries()         .build(); |

В этом случае вы не будете получать Exception при работе в UI потоке. Но вы должны понимать, что это плохая практика, и может добавить ощутимых тормозов вашему приложению.

Переход на Room

Если вы надумали с SQLite мигрировать на Room, то вот пара полезных ссылок по этой теме:

<https://medium.com/google-developers/incrementally-migrate-from-sqlite-to-room-66c2f655b377>

<https://medium.com/@price.yvonne.86/quick-and-easy-migration-to-room-d40dbb142b51>