[Урок 10 Room Запрос из нескольких таблиц Relation](https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/534-urok-10-room-zapros-iz-neskolkih-tablic-relation.html" \o "Урок 10. Room. Запрос из нескольких таблиц. Relation)

В этом уроке рассмотрим, как получать данные из нескольких таблиц. А также разберемся, как использовать аннотацию Relation.

Для примера будем использовать две таблицы: сотрудники и отделы. Каждый сотрудник прикреплен к какому-либо отделу.

Entity объект для отделов:

|  |
| --- |
| @Entity  public class Department {       @PrimaryKey     public int id;       public String name;    } |

Entity объект для сотрудников:

|  |
| --- |
| @Entity  public class Employee {       @PrimaryKey     public long id;       public String name;       public int salary;       @ColumnInfo(name = "department\_id")     public int departmentId;    } |

В поле departmentId хранится id отдела, к которому прикреплен сотрудник.

Мы хотим получить список работников, в котором будет следующая информация: имя работника, его зарплата, наименование его отдела. Для этого нам надо будет написать запрос, который вытащит данные из двух таблиц.

Описываем метод в Dao объекте

@Dao

public interface EmployeeDao {

   @Query("SELECT employee.name, employee.salary, department.name AS department\_name " +

       "FROM employee, department " +

       "WHERE department.id == employee.department\_id")

   public List<EmployeeDepartment> getEmployeeWithDepartment();

   // ...

}

Т.к. поле name есть в обоих таблицах, то для отдела переименовываем его в department\_name

Обратите внимание на тип объектов, который мы будем получать от этого метода. Это EmployeeDepartment. Нам нужно создать этот объект, и указать в нем все поля, которые мы ожидаем получить от запроса.

|  |
| --- |
| public class EmployeeDepartment {       public String name;       public int salary;       @ColumnInfo(name = "department\_name")     public String departmentName;    } |

Это не Entity объект, а обычный класс. Поля этого класса должны совпадать с полями результата, который вернет запрос. Room конвертирует результаты запроса в список этих объектов, и мы получим то, что хотели.

Relation

Аннотация [Relation](https://developer.android.com/reference/android/arch/persistence/room/Relation.html" \t "_blank) также позволяет делать запросы из нескольких таблиц, но структура результата будет немного другой. И нам самим не придется писать сложные запросы. Room все сделает за нас.

Давайте представим, что нам надо получить список отделов. И к каждому отделу должен прилагаться список сотрудников.

Структура для этих данных будет выглядеть так:

|  |
| --- |
| public class DepartmentWithEmployees {       public int id;       public String name;       @Relation(parentColumn = "id", entityColumn = "department\_id")     public List<Employee> employees;    } |

Это не Entity, а обычный класс. В полях id и name будут данные отдела.

В employees будет список сотрудников этого отдела. Для этого мы помечаем список аннотацией Relation, и Room сам заполнит его для нас. Давайте разбираться, как именно Room поймет, что он должен поместить в этот список. Откуда он будет брать данные и по какому условию?

Тип данных списка - это Employee. Это Entity объект, для него в базе данных создана таблица. Из этой таблицы Room и будет читать данные по сотрудникам. В параметрах parentColumn и entityColumn указываем названия полей, которые участвуют в условии выборки данных. В результате, Room будет искать сотрудников, у которых entityColumn (т.е. department\_id) равен parentColumn (т.е. id) отдела. Все найденные сотрудники окажутся в employees.

По требованиям Room, тип employees должен быть List или Set.

Осталось описать метод в Dao:

|  |
| --- |
| @Dao  public interface DepartmentDao {       @Query("SELECT id, name from department")     List<DepartmentWithEmployees> getDepartmentsWithEmployees();       // ...    } |

Это простой запрос, который вытащит необходимые данные по отделу. А запрос по сотрудникам для каждого отдела сделает за нас Room.

В классе DepartmentWithEmployees мы используем поля id и name для данных по отделу. Но класс Department имеет точно такую же структуру - id и name. Поэтому мы в DepartmentWithEmployees можем заменить эти поля на одно поле с типом Department и аннотацией Embedded:

|  |
| --- |
| public class DepartmentWithEmployees {       @Embedded     public Department department;       @Relation(parentColumn = "id", entityColumn = "department\_id")     public List<Employee> employees;    } |

Предположим, что нам нужны не все данные по сотрудникам, а только некоторые поля. Например, name и salary. Создаем под них класс:

|  |
| --- |
| public class EmployeeNameAndSalary {       public String name;       public int salary;    } |

И используем его, как тип в Relation-списке

|  |
| --- |
| public class DepartmentWithEmployees {       public int id;       public String name;       @Relation(parentColumn = "id", entityColumn = "department\_id", entity = Employee.class)     public List<EmployeeNameAndSalary> employees;    } |

А чтобы Room знал, откуда брать данные по сотрудникам, указываем Entity класс Employee в параметре entity.

Relation может быть вложенным. Т.е. в нашем примере класс EmployeeNameAndSalary также может содержать в себе Relation, который будет для каждого сотрудника собирать, например, список техники, записанной на него.

Relation не может быть использован в Entity классах, только в обычных. Relation поле не может задаваться через конструктор. Оно должно быть public или иметь public set-метод.

Relation + Transaction

При использовании Relation, Room выполняет несколько запросов, чтобы собрать все данные. Имеет смысл выполнять все эти запросы в одной транзакции, чтобы получить корректные данные. Для этого можно использовать аннотацию Transaction

|  |  |
| --- | --- |
| 1  2  3 | @Transaction  @Query("SELECT id, name from department")  List<DepartmentWithEmployees> getDepartmentsWithEmployees(); |