[Урок 8 Room Query](https://startandroid.ru/ru/blog/27-course/architecture-components/532-urok-8-room-query.html" \o "Урок 8. Room. Query)

09 января 2018

В этом уроке поговорим подробнее про Query. В каком виде мы можем получать данные: List, массив, Cursor, LiveData. Как передавать параметры. Как получать только некоторые поля. Как с помощью Query выполнять update и delete запросы в Room.

В качестве примера будем работать с таким Entity классом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @Entity()  public class Employee {       @PrimaryKey()     public long id;       @ColumnInfo(name = "first\_name")     public String firstName;       @ColumnInfo(name = "last\_name")     public String lastName;       public int salary;    } |

List, массив, Cursor

Чтобы запросить из базы Employee-объекты, необходимо в Dao создать метод с аннотацией Query

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Dao  public interface EmployeeDao {       @Query("SELECT \* FROM employee")     List<Employee> getAll();       // ...    } |

В Query прописываем запрос, который должен вернуть данные. А в качестве возвращаемого типа указываем List<Employee>.

При вызове этого метода, Room сделает запрос в таблицу employee, конвертирует полученные данные в Employee объекты и упакует их в List.

Запрос, который вы указываете в Query проверяется на правильность синтаксиса во время компиляции. Если в нем будет ошибка, система вам сразу подскажет это.

Вместо List, мы также можем использовать массив:

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee")  Employee[] getAll(); |

и даже Cursor, если это необходимо по каким-то причинам:

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee")  Cursor getAll(); |

LiveData

Room умеет возвращать данные в [LiveData](https://startandroid.ru/ru/blog/27-course/architecture-components/525-urok-2-livedata.html) обертке.

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee")  LiveData<List<Employee>> getAll(); |

Получение данных в коде Activity выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | LiveData<List<Employee>> employeesLiveData = db.employeeDao().getAll();    employeesLiveData.observe(this, new Observer<List<Employee>>() {     @Override     public void onChanged(@Nullable List<Employee> employees) {         log("onChanged " + employees);     }  }); |

Получаем LiveData и подписываемся на него.

Использование LiveData имеет огромное преимущество перед использование списка или массива. Подписавшись на LiveData, вы будете получать свежие данные при их изменении в базе. Т.е. при добавлении новых, удалении старых или обновлении текущих данных в таблице employee, Room снова выполнит ваш Query запрос, и вы получите в onChanged методе актуальные данные с учетом последних изменений. Вам больше не надо самим запрашивать эти данные каждый раз. И все это будет приходить вам в UI поток.

Передача параметров

В Query можно передавать параметры, чтобы сделать запросы более конкретными.

Например, запрос данных по id

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee WHERE id = :employeeId")  Employee getById(long employeeId); |

Перед параметром employeeId в запросе должно стоять двоеточие. Room возьмет значение этого параметра из метода и подставит его в запрос.

Рассмотрим еще несколько примеров:

Поиск сотрудников с зарплатой больше заданного значения

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee WHERE salary > :minSalary")  List<Employee> getAllWithSalaryMoreThan(int minSalary); |

Поиск сотрудников с зарплатой в заданном диапазоне

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee WHERE salary BETWEEN :minSalary AND :maxSalary")  List<Employee> getAllWithSalaryBetween(int minSalary, int maxSalary); |

Поиск сотрудников по имени или фамилии

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee WHERE first\_name LIKE :search OR last\_name LIKE :search")  List<Employee> getAllWithNameLike(String search); |

Поиск сотрудников по списку id.

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT \* FROM employee WHERE id IN (:idList)")  List<Employee> getByIdList(List<Long> idList); |

Subsets

Часто при запросе данных нам нужно получить из таблицы не все поля, а только некоторые. Такие запросы быстрее и легче, чем тянуть все поля.

Допустим нам надо получать только имя и фамилию сотрудника. Если сделать так:

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT first\_name, last\_name FROM employee")  List<Employee> getNames(); |

то уже при компиляции получим ошибку: *The columns returned by the query does not have the fields [id,salary] in Employee even though they are annotated as non-null or primitive. Columns returned by the query: [first\_name,last\_name].*

Room сообщает, что в данных, которые вернет этот запрос, не хватает полей, чтобы заполнить все поля объекта Employee.

В этом случае мы можем использовать отдельный объект.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Name {       @ColumnInfo(name = "first\_name")     public String firstName;       @ColumnInfo(name = "last\_name")     public String lastName;    } |

Обратите внимание, что он не Entity. Это обычный класс. С помощью ColumnInfo мы настраиваем имена полей, чтобы они совпадали с полями таблицы.

Используем этот класс в методе запроса:

|  |  |
| --- | --- |
| 1  2 | @Query("SELECT first\_name, last\_name FROM employee")  List<Name> getNames(); |

Теперь все ок, и мы получим список Name объектов.

Вы также можете в этих не Entity классах использовать вложенные классы с аннотацией @Embedded. Подробно об этой аннотации мы говорили в [Уроке 6](https://startandroid.ru/ru/blog/27-course/architecture-components/530-urok-6-room-entity.html).

insert, update и delete запросы

Аннотации Insert, Update и Delete позволяют нам модифицировать данные, но их возможности слишком ограниченны. Часто возникает необходимость обновить только некоторые поля или удалить записи по определенному условию. Это можно сделать запросами с помощью Query.

Давайте рассмотрим пару примеров.

Обновление зарплат у сотрудников по списку id.

|  |  |
| --- | --- |
| 1  2 | @Query("UPDATE employee SET salary = :newSalary WHERE id IN (:idList)")  int updateSalaryByIdList(List<Long> idList, int newSalary); |

Опционально метод может возвращать int значение, в котором мы получим количество обновленных строк. Если вам это не нужно, то делайте метод void.

Вызов метода будет выглядеть так:

|  |  |
| --- | --- |
| 1 | int updatedCount = db.employeeDao().updateSalaryByIdList(Arrays.asList(1L, 3L, 4L), 10000); |

Удаление сотрудников по списку id

|  |  |
| --- | --- |
| 1  2 | @Query("DELETE from employee WHERE id IN (:idList)")  int deleteByIdList(List<Long> idList); |

Запросы удаления также могут возвращать int значение, в котором мы получим количество удаленных строк.

Вызов метода:

|  |  |
| --- | --- |
| 1 | int deletedCount = db.employeeDao().deleteByIdList(Arrays.asList(1L, 3L, 4L)); |