[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" \o "Урок 7. Room. Insert, Update, Delete, Transaction)

09 января 2018

В этом уроке рассмотрим подробнее аннотации Insert, Update и Delete. А также узнаем, как использовать транзакции в Room.

## **Insert**

Аннотация Insert - это простой способ вставить объект в базу данных. Мы уже использовали ее в примерах прошлых уроков.

Использование этой аннотации выглядит так:

|  |
| --- |
| @Dao  public interface EmployeeDao {       @Insert     void insert(Employee employee);       // ...    } |

В Dao интерфейсе описываем метод, который на вход принимает Entity объект. К методу добавляем аннотацию Insert и Room сгенерирует необходимый код в реализации этого интерфейса.

Давайте посмотрим, какие еще возможности у нас есть.

### **Вставка нескольких объектов**

Мы можем передавать в метод не один, а несколько объектов, используя varargs

|  |  |
| --- | --- |
| 1  2 | @Insert  void insertMany(Employee... employees); |

Также, это может быть список:

|  |  |
| --- | --- |
| 1  2 | @Insert  void insert(List<Employee> employees); |

Или это вообще может быть любой Iterable:

|  |  |
| --- | --- |
| 1  2 | @Insert  void insert(Iterable<Employee> employees); |

При вызове этого метода вы можете использовать массив или коллекцию.

### **Получение id**

При вставке метод Insert может возвращать id только что добавленной записи. Для этого надо описать метод так, чтобы он возвращал long.

|  |  |
| --- | --- |
| 1  2 | @Insert  long insert(Employee employee); |

Если в Employee есть числовой первичный ключ, то именно его значение вы и получите.

В случае добавления нескольких записей, необходимо использовать long[]

|  |  |
| --- | --- |
| 1  2 | @Insert  long[] insert(List<Employee> employees); |

или List<Long>

|  |  |
| --- | --- |
| 1  2 | @Insert  List<Long> insert(List<Employee> employees); |

### **Режимы вставки**

Рассмотрим ситуацию, когда мы вставляем в таблицу запись, но обнаруживается, что запись с таким ключом там уже есть. По умолчанию мы получим ошибку: SQLiteConstraintException: UNIQUE constraint failed. И ничего в базу не запишется.

Но это можно поменять с помощью параметра onConflict.

|  |  |
| --- | --- |
| 1  2 | @Insert(onConflict = OnConflictStrategy.REPLACE)  void insert(Employee employee); |

В режиме REPLACE старая запись будет заменена новой. Этот режим хорошо подходит, если вам надо вставить запись, если ее еще нет в таблице или обновить запись, если она уже есть.

Также есть режим IGNORE. В этом режиме будет оставлена старая запись и операция вставки не будет выполнена.

Более подробно об этих режимах можно прочесть [здесь](https://sqlite.org/lang_conflict.html).

## **Update**

Эта аннотация аналогична Insert, но она не вставляет, а обновляет объекты в бд.

|  |  |
| --- | --- |
| 1  2 | @Update  void update(Employee employee); |

Так же, как и с Insert мы можем использовать коллекции и varargs, чтобы обновлять несколько объектов сразу.

Update ищет в бд запись по ключу. Если не найдет, то ничего не произойдет. Если найдет, то обновит все поля, а не только те, которые мы заполнили в Entity объекте.

Мы можем получить количество обновленных записей. Для этого опишите метод так, чтобы он возвращал int.

|  |  |
| --- | --- |
| 1  2 | @Update  int update(List<Employee> employee); |

Как и Insert, Update поддерживает параметр onConflict.

## **Delete**

Методы с аннотацией Delete будут удалять объекты.

|  |  |
| --- | --- |
| 1  2 | @Delete  void delete(Employee employee); |

В Delete методах мы также можем использовать коллекции и varargs, чтобы удалять несколько объектов сразу.

Delete ищет в бд запись по ключу.

Мы можем получить количество удаленных записей. Для этого необходимо описать метод так, чтобы он возвращал int.

|  |  |
| --- | --- |
| 1  2 | @Delete  int delete(List<Employee> employee); |

Аннотации Insert, Update и Delete позволяют выполнить простые операции. Для более сложных действий необходимо использовать SQL запросы: INSERT, UPDATE и DELETE. Это можно сделать с помощью аннотации Query. В следующем уроке мы рассмотрим эту возможность.

## **Транзакции**

Аннотация [@Transaction](https://developer.android.com/reference/android/arch/persistence/room/Transaction.html) позволяет выполнять несколько методов в рамках одной транзакции.

Рассмотрим пример, когда нам нужно добавить объекты Car и Employee:

|  |
| --- |
| @Dao  public abstract class EmployeeCarDao {       @Insert     public abstract void insertEmployee(Employee employee);       @Insert     public abstract void insertCar(Car car);         @Transaction     public void insertCarAndEmployee(Car car, Employee employee) {         insertCar(car);         insertEmployee(employee);     }    } |

EmployeeCarDao - отдельный Dao объект для работы с комбинацией Car и Employee. В нем описываем методы для вставки объектов по отдельности, а затем оба этих метода вызываем в одном методе с аннотацией Transaction. В итоге вставятся либо оба объекта, либо, в случае возникновения ошибки, ни один из них.

Обратите внимание, что в этом случае Dao - не интерфейс, а абстрактный класс.