Библиотека Room (Google Inc.) – помощь в работе с SQLLite

@Entity – помечается объект, который мы хотим хранить в БД. Для этого сначала создается класс, описывающий объект БД. Такой объект – строка в таблице. Имя таблицы будет совпадать с именем класса (если не передать параметр tableName), а поля заполняться в соответствии со значениями полей класса.

@Dao – описание методов для работы с БД. (Что-то типа View (хранимых процедур)). Данной анотацией помечается интерфейс, отдельные методы имеют пометки: @Query, @Update, @Delete, @Insert смысл понятен из названий. В @Query(...) содержится SQL запрос.

@Database(…) – помечается основной класс по работе с БД. Класс **абстрактный и наследуется от RoomDatabase.** В параметрах указываем классы Entity, которые будут использоваться и версию базы. Для каждого класса в множестве entities будет создана таблица.

Практика:

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

Передаем контекст приложения, класс Database, имя файла для БД

Этот код будет каждый раз создавать новый экземпляр AppDatabase. Рекомендуется использовать один экземпляр для ВСЕХ операций. Поэтому вызывать данную конструкцию стоит лишь 1 раз. Хорошим решением будет обернуть его в Singleton или использовать в классе App.

Встала следующая проблема: как посмотреть текущее состояние БД. Для этого были использованы следующие зависимости:

// Sletho Library (для просмотра БД)  
// Stetho core  
implementation 'com.facebook.stetho:stetho:1.5.0'  
//If you want to add a network helper  
implementation 'com.facebook.stetho:stetho-okhttp:1.5.0'

И с помощью Google Chrome удалось просмотреть БД. (URL: “chrome://inspect”). И если телефон подключен, а приложение запущено, то можно посмотреть состояние БД (и еще кучу всего)

Правда информация обновляется не сразу.

Создание внешних ключей:

@Entity(foreignKeys = @ForeignKey(entity = Employee.class, parentColumns = "id", childColumns = "employeeID"))

Указываем класс и поле, на которое будем ссылаться, а затем уже поле самого FK.

Добавление прцоедуры onDelete:

@Entity(foreignKeys = @ForeignKey(entity = Employee.class, parentColumns = "id", childColumns = "employeeID", onDelete = *CASCADE*))

При удалении родителя, будут удалены связанные с ним внешние ключи. Т.е. при удалении сотрудника, удаляются все его автомобили.