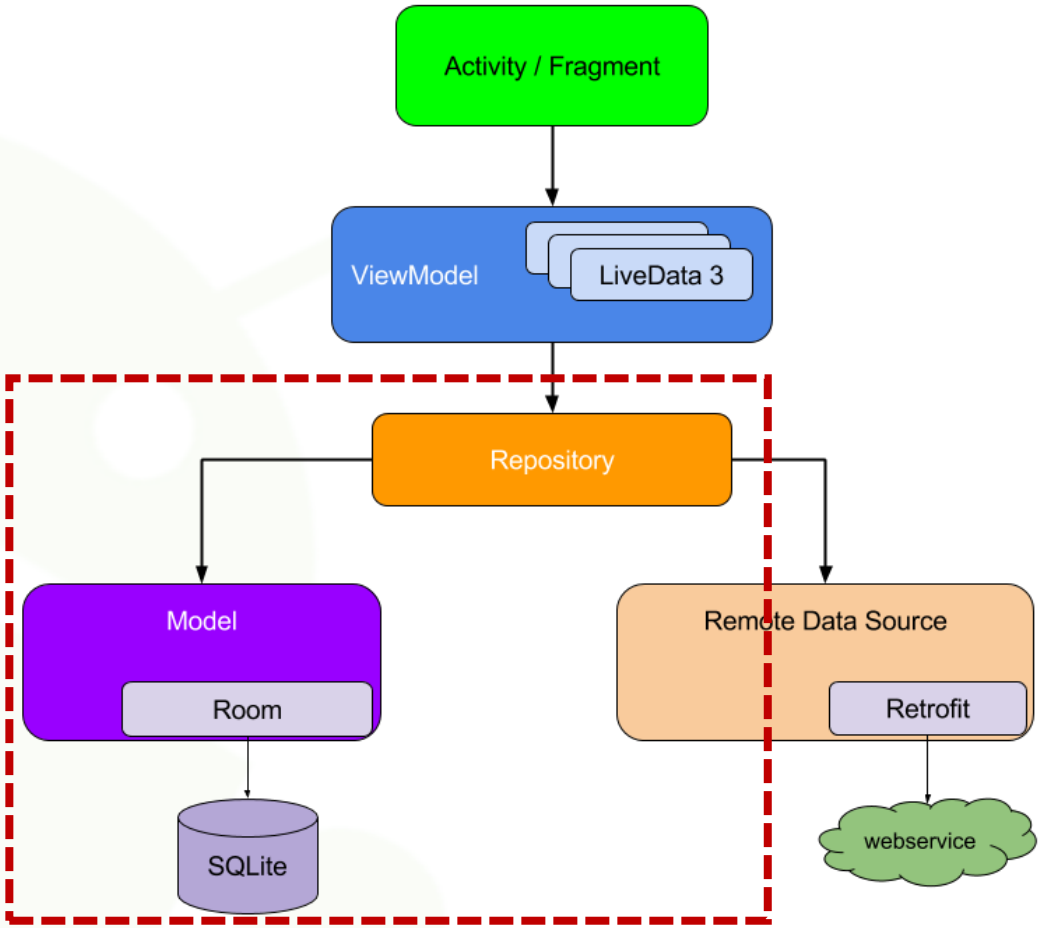
CSci 245
Mobile Software Development

Instructor:
Dr. Shuo Niu (shniu@clarku.edu)

Ten Core Modules

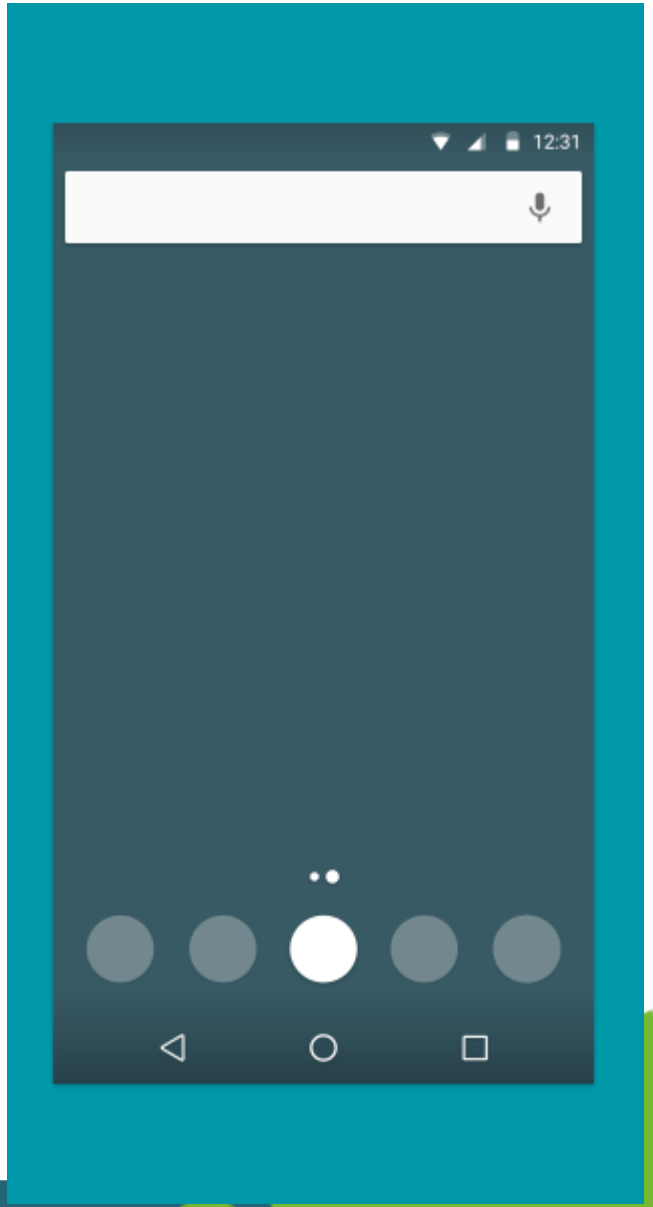
1. Intro to Android
2. Into to Kotlin
3. Mobile GUI
4. Activity and Fragment
5. Navigation
6. Architecture Components
7. Internet API
8. Database
9. Cloud Computing
10. Media and Animation
11. Sensors and Location
12. Background Processing

Architecture Components



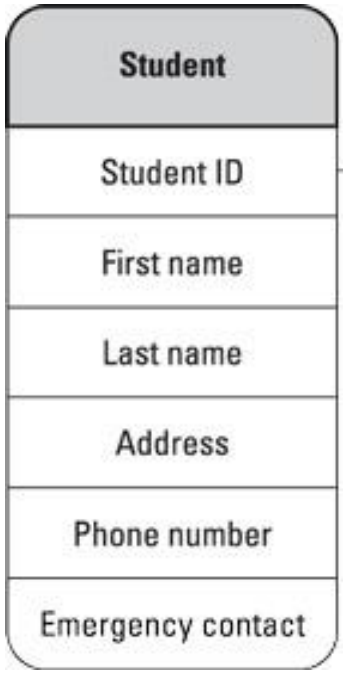
Why Database?

- Variables
  - Stored in Activities or Fragments.
  - Instance state lost
- ViewModel and Live Data
  - Preserve instance states and lifecycle aware
  - Data cleared when the user closed the app
- Database
  - Preserve the data even the app is closed



Abstract Concepts

- Entity – person, place, object or event
  - stored as a record or a table row
- Attribute – characteristic of an entity
  - stored as field or table column

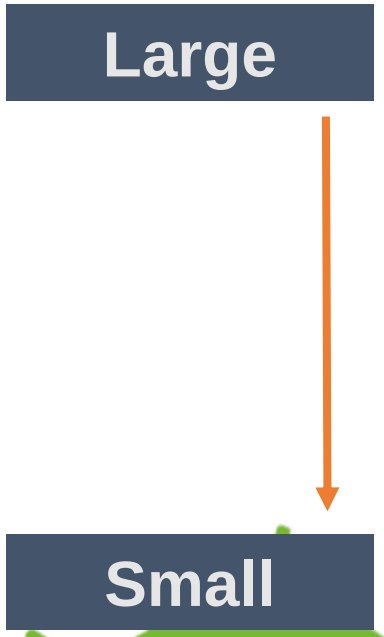



```
class Student(
    var studentID: String,
    var firstName: String,
    var lastName: String,
    var address: String,
    var phoneNumber: String,
    var emergencyContact: String
)
```

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

Database Concepts

- Database – a collection of related tables
- Tables – a collection of related records
  – collection of related entities
- Record – collection of fields (table row)
  – represents an entity
- Field – characters of a record (table column)
  – represents an attribute



student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

Fields and Records

- Fields
  - A field is an attribute of an entity
- Records
  - A record is an entity
  - A record consists of multiple fields

address
307 Westerfield Lane

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565

student_id	first_name	last_name	address	phone_number	emergency contact
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014

Tables

- A bunch of records form a table

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

- A table is a group of related entities

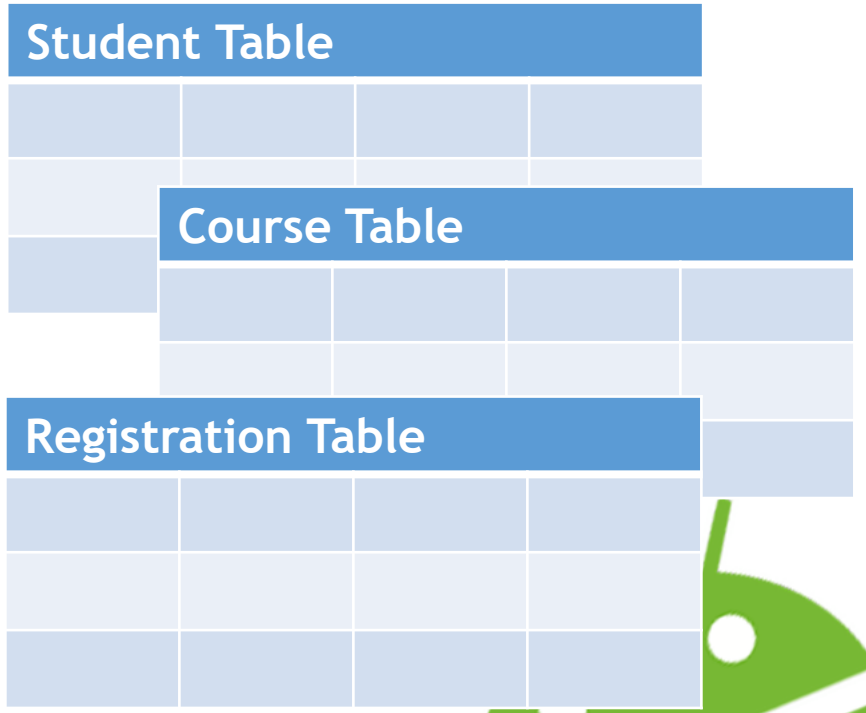
Database

- A bunch of tables form a database

Student
Student ID
First name
Last name
Address
Phone number
Emergency contact

Student_Courses
Student ID
Course ID
Final grade

Courses
Course ID
Course name
Professor name
Location
Meeting time



Properties of Tables

- Values are atomic
  - A cell must be string, numeric, Boolean, datetime, or time interval.
  - No compound data type. E.g. no field stores array data (unless convert to strings)
- Column values are of the same kind
  - All records in the same table have the same set of attributes, and an attribute of the different records have the same type.
- The sequence of columns is insignificant
  - Columns can be retrieved in any order
- The sequence of rows is insignificant
  - The order of rows in a table has no meaning
  - Rows can be retrieved in any order
- Columns have unique names

Properties of Tables

- Each row is unique
  - No two rows in a table are identical
  - There is at least one column serve as primary (column identifier, no duplicates)
  - Usually called "primary key" of a table

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

Local Databases

- Relational databases typically use Structured Query Language (**SQL**) to define, manage, and search data
- Most apps/sites do four general tasks with data in a database (CRUD):
  - Create new rows
  - Read existing data
  - Update / modify values in existing rows
  - Delete rows

SQL

- **Structured Query Language** (SQL): a language for searching and updating a database
  - a standard syntax that is used by all database software
  - Android database supports SQL queries
- `Create:` `INSERT`
- `Read:` `SELECT`
- `Update:` `UPDATE`
- `Delete:` `DELETE`

SQL

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

- Create: `INSERT INTO <table_name> VALUES <values>`

```
INSERT INTO student_table VALUES ('S106', 'Christopher', 'Nolan', '5 Main St', '508-254-1234', '508-254-4321');
```

- Read: `SELECT <column(s)> FROM <table_name> WHERE <condition>`

```
SELECT * FROM student_table
```
Select all fields

```
SELECT last_name FROM student_table
```
Select all last_names

```
SELECT last_name FROM student_table WHERE first_name = 'Nanci'
```

SQL

student_id	first_name	last_name	address	phone_number	emergency contact
S101	Renata	Wickmann	307 Westerfield Lane	949-916-2704	238-223-0565
S102	Nanci	Arnoult	9 Emmet Hill	984-104-1650	946-154-0014
S103	Trude	Wyllcocks	688 Canary Avenue	437-981-4778	441-147-9234
S104	Sylvester	Lemm	63320 Barby Hill	246-126-9827	172-463-3596
S105	Noell	Buckby	7 Mesta Park	719-785-0454	760-646-1652

- Delete: `DELETE FROM <table_name> WHERE <condition>`

```
DELETE FROM student_table WHERE first_name='Nanci'
DELETE FROM student_table
DELETE FROM class_table WHERE grade<60
DELETE FROM student_table WHERE student_id='S102' OR student_id='S105'
```

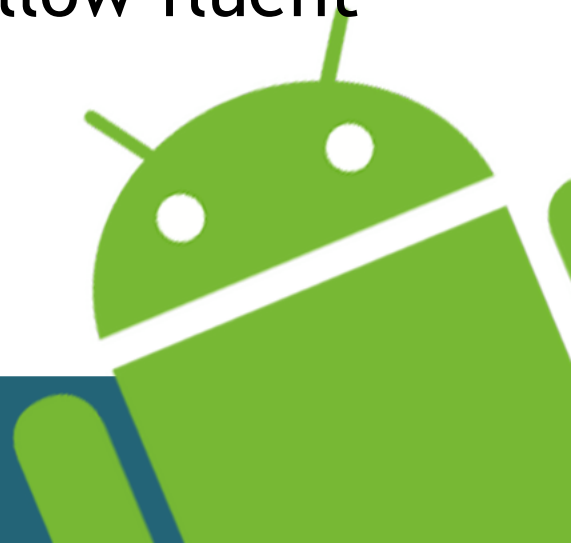
Select all records

- Update: `UPDATE <table_name> SET <field=value> WHERE <condition>`

```
UPDATE student_table SET address = '5 Main St.' WHERE student_id = 'S104'
```

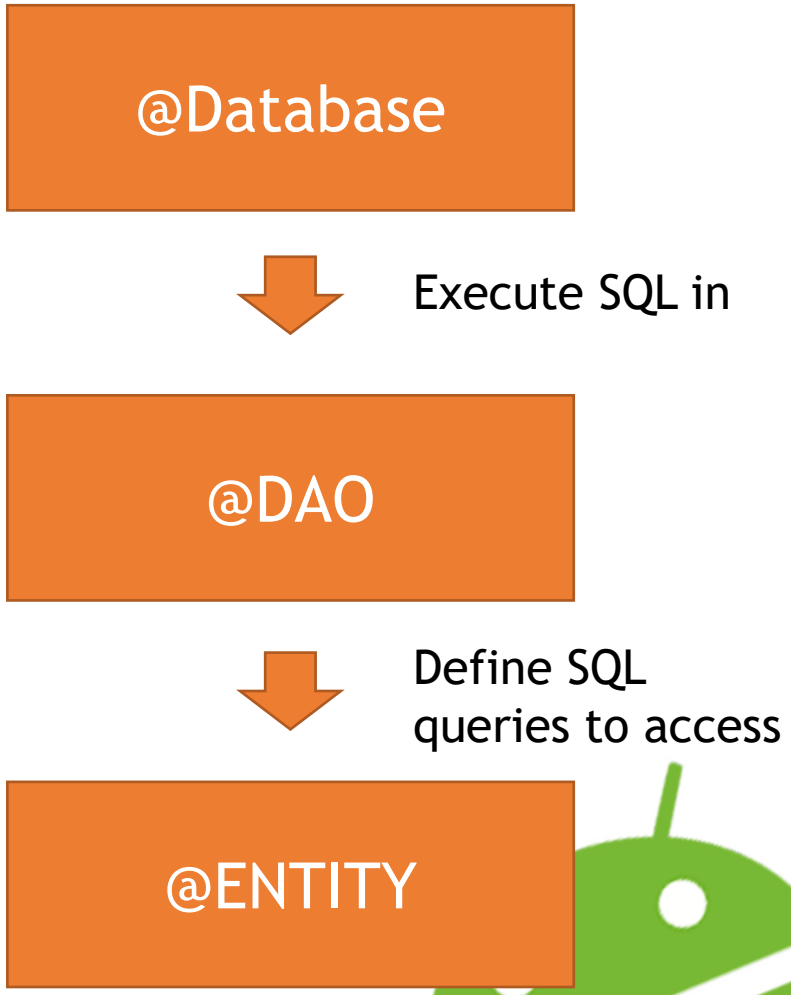
SQLite and Room Databases

- SQLite is a library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.
- A complete SQL database with multiple tables is contained in a single disk file.
- Room provides an abstraction layer over SQLite to allow fluent database access.



Room Database

- @Entity
  - Define entities of the database
- @DAO
  - Data Access Objects are the main classes where you define your database interactions.
  - They can include a variety of SQL query methods.
- @Database
  - Represent a database



@Entity

- @Entity: declare a class to be stored in the database
- @PrimaryKey: specify which property serves as primary key
  - Thus cannot have duplicated values

```
@Entity(tableName = "weatherTable")
class City {
    @PrimaryKey
    var name = "city name"
    var temperature = 0.0
    var pressure = 0
    var humidity = 0
}
```

name	temperature	pressure	humidity
Boston	79.6	1042	80
Worcester	76.3	1055	82

@DAO

- Data Access Object is the main classe where you define your database interactions.

```
@Dao
interface WeatherDAO {

    @Query("SELECT * FROM weatherTable")
    fun getAll(): List<City>

}
```

SQL query

Returned data type

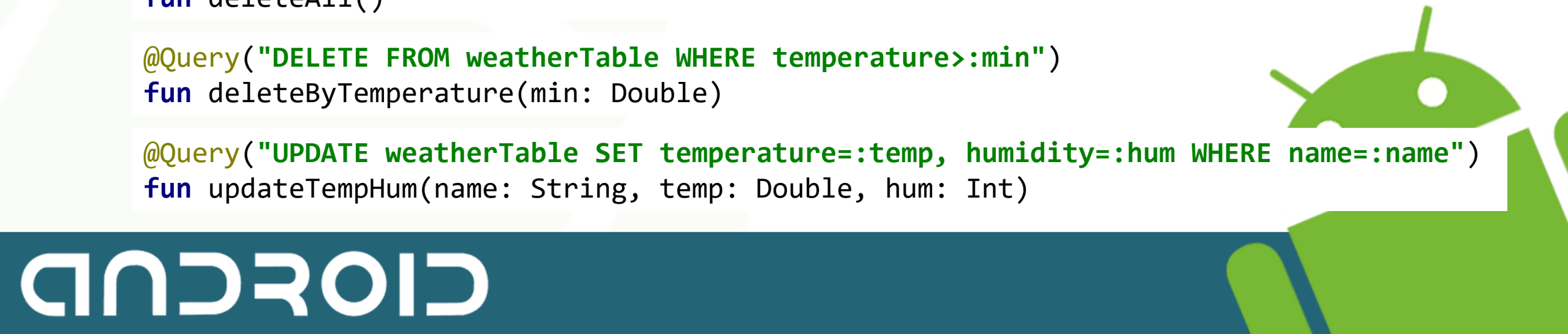
Function to execute

name	temperature	pressure	humidity
Boston	79	1042	80
Worcester	76	1055	82

@Query: any SQL query

Get the temperate by city name?

```
@Query("SELECT * FROM weatherTable")
fun getAll(): List<City>

@Query("SELECT * FROM weatherTable WHERE temperature>:min AND temperature<:max")
fun getCityByDegreeRange(min:Double, max:Double):List<City>

@Query("SELECT name FROM weatherTable WHERE humidity>:min")
fun getCityByHumidity(min: Int): List<String>

@Query("DELETE FROM weatherTable")
fun deleteAll()

@Query("DELETE FROM weatherTable WHERE temperature>:min")
fun deleteByTemperature(min: Double)

@Query("UPDATE weatherTable SET temperature=:temp, humidity=:hum WHERE name=:name")
fun updateTempHum(name: String, temp: Double, hum: Int)
```



@Insert and @Delete

- @Insert: insert a record into the table

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
fun insert(weather: City)
```

- Delete: delete a record from the database

```
@Delete
fun delete(city: City)
```

@Database

```
@Database(entities = [City::class], version = 1)
abstract class WeatherDB : RoomDatabase() {
    abstract fun weatherDAO(): WeatherDAO

    companion object {
        private var INSTANT: WeatherDB? = null

        fun getDBObject(context: Context): WeatherDB? {
            if (INSTANT == null) {
                synchronized(WeatherDB::class.java) {
                    INSTANT = Room.databaseBuilder(context, WeatherDB::class.java, "weatherDB")
                        .allowMainThreadQueries()
                        .fallbackToDestructiveMigration()
                        .build()
                }
            }
            return INSTANT
        }
    }
}
```

@Database

```
@Database(entities = [City::class], version = 1)
abstract class WeatherDB : RoomDatabase() {
    abstract fun weatherDAO(): WeatherDAO

    companion object {
        private var INSTANT: WeatherDB? = null

        fun getDBObject(context: Context): WeatherDB? {

            …
            return INSTANT
        }
    }
}
```

Control database version.
- If City class is changed, update the version number

```
@Dao
interface WeatherDAO {

    @Query("SELECT * FROM weatherTable")
    fun getAll(): List<City>

}
```

```
database.weatherDAO()?.getAll()
```

Questions?