CMPS 312

Read Chapter 11

Data Management

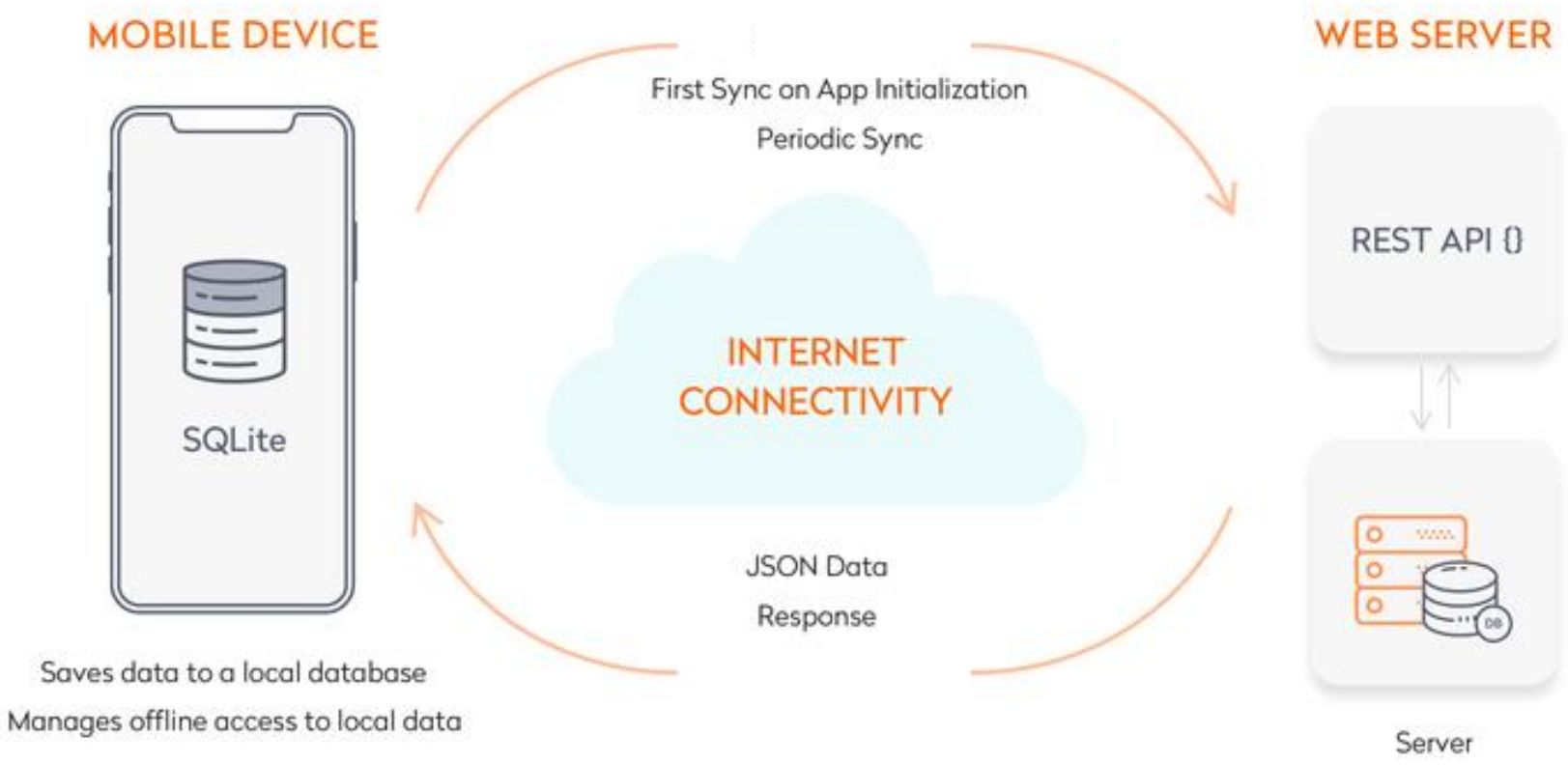
Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Data persistence options on Android
2. Room programming model
3. Relationships

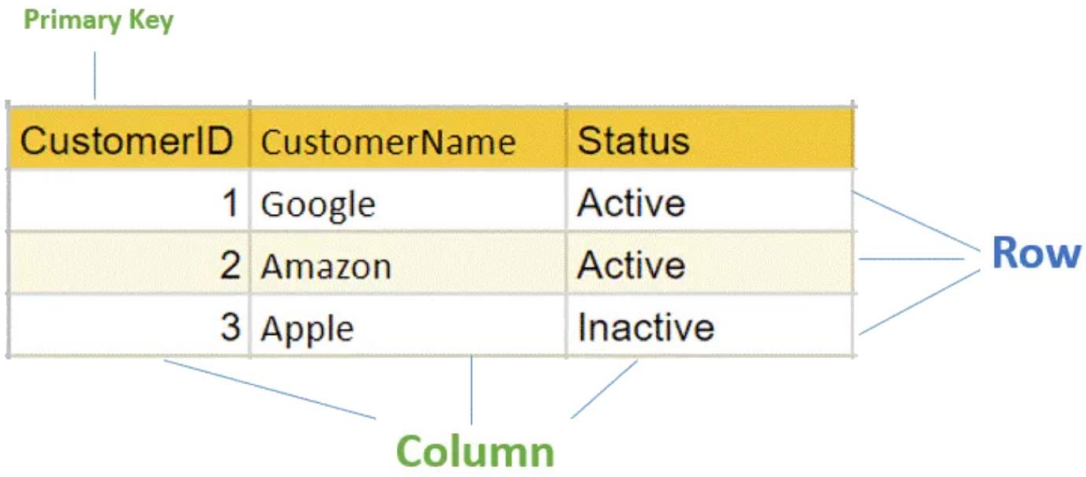
Offline app with Sync



- **Cache** relevant pieces of data on the device. App continues to work offline when a network connection is not available.
- When the network connection is back, the app's repository **syncs** the data with the server.

Relational Database

- Database allows **persisting structured data**
- A **relational** database organizes data into **tables**
  - A table has rows and columns
  - Tables can have relationships between them
- Tables could be queries and altered using SQL



SQL Statements

- Structured Query Language (**SQL**)
  - Language used to define, query and alter database tables
  - SQL is a language for interacting with a relational database
- **C**reating data:

```
INSERT into person (first_name, last_name)
  VALUES ("Ahmed", "Sayed")
```

- **R**eading data:

```
SELECT * FROM person WHERE last_name = "Sayed"
```

- **U**pdating data:

```
UPDATE person SET first_name = "Ali" where
     last_name = "Sayed"
```

- **D**eleting data:

```
DELETE from person where last_name = "Sayed"
```

Room Dependencies

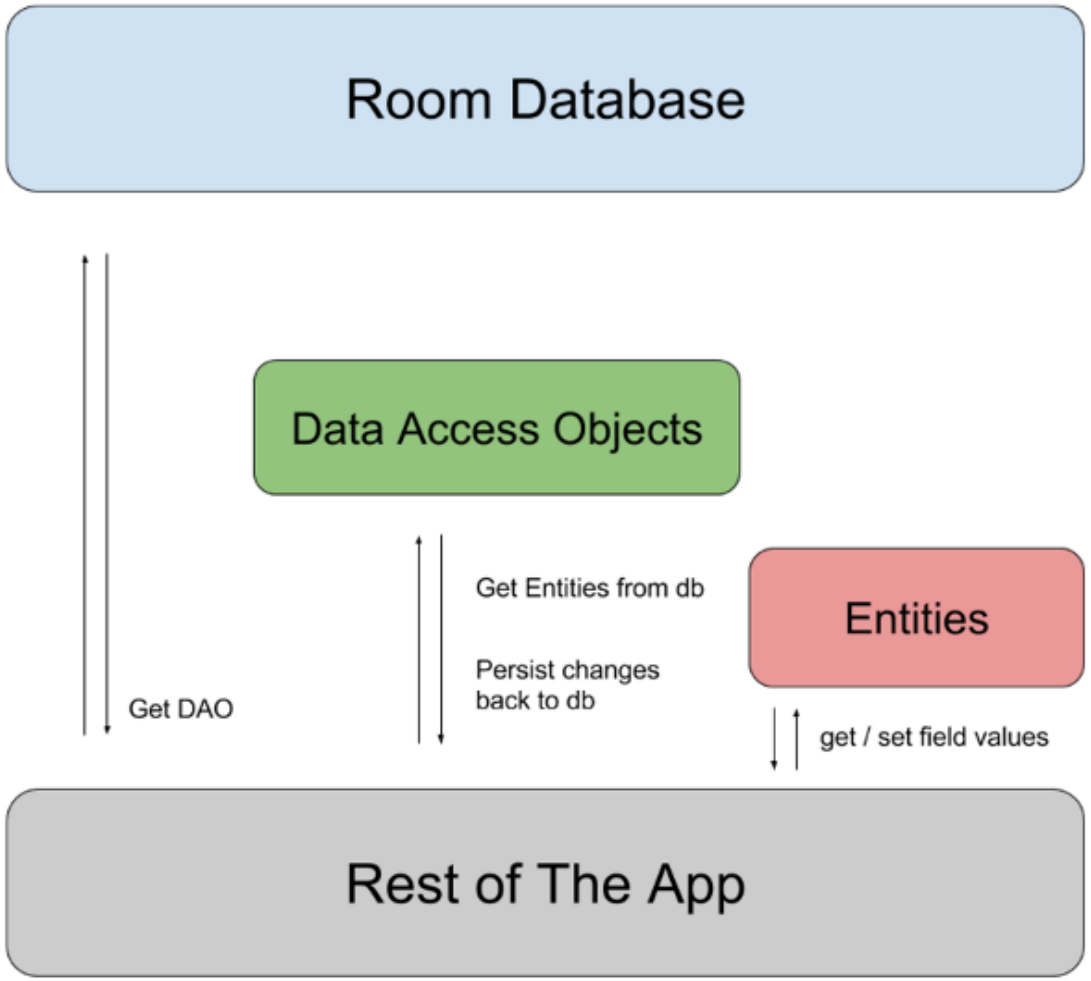
```
def room_version = "2.2.5"

implementation "androidx.room:room-runtime:$room_version"

kapt "androidx.room:room-compiler:$room_version"

// Kotlin Extensions and Coroutines support for Room

implementation "androidx.room:room-ktx:$room_version"
```

Room architecture diagram

Working with Room

- Model DB Tables as regular **entity classes**
- Define queries for Insert, Update and Delete in **DAO interfaces**. Implementation is auto-generated
- Interact with the database using DAOs

3 major components in Room



Room main components

- **Entity** → app class maps to a table within the database
  - Kotlin class annotated with **@Entity** to map it to a DB table
  - Must specify one of the entity properties as a **primary key**
  - Table representation (e.g., name and column names) is controlled by annotations
- Data Access Object (**DAO**) → how you persist and retrieve entities
  - Contains CRUD methods defining operations to be done on data
  - Interface or abstract class marked as **@Dao**
  - One or many DAOs per database
- **Database** → where data is persisted
  - abstract class that extends **RoomDatabase** and annotated with @Database

Entity

- Entity represents a table in a relational database, and each entity instance corresponds to a row in that table
- Each entity object has a Primary Key that **Uniquely identifies** the entity object in memory and in the DB
- The primary key can be generated in the database by specifying `autoGenerate = true`

```
@Entity
data class Item(
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0,
    val name: String, var quantity: Int)
```

Customizing Entity Annotation

- In most cases, the defaults are sufficient
- By default the table name corresponds to the name of the class

OPTIONAL Use @Entity (tableName = "…") to set the name of the table

OPTIONAL The columns can be customized using **@ColumnInfo**(name = "column_name") annotation

- If an entity has fields that you don't want to persist, you can annotate them using **@Ignore**
- If multiple constructors are available, add the **@Ignore** annotation to tell Room which should be used and which not

DAO @Query

- **@Query** used to annotate query methods
- Room ensures **compile time verification** of SQL queries

```
@Dao
interface UserDao {
    @Query("select * from User limit 1")
    suspend fun getFirstUser(): User
    @Query("select * from User")
    fun getAll(): List<User>
    @Query("select firstName from User")
    fun getFirstNames(): List<String>
    @Query("select * from User where firstName = :fn")
    fun getUsers(fn: String): List<User>
    @Query("delete from User where lastName = :ln")
    fun deleteUsers(ln: String): Int
}
```

DAO @Insert, @Update, @Delete

- Used to annotate insert, update and delete methods
- Suspend ensure that DB operations are not done on the main UI thread

```
@Dao
interface UserDao {
    @Insert
    suspend fun insert(user: User): Long
    @Insert
    suspend fun insertList(users: List<User>): List<Long>

    @Delete
    suspend fun delete(user: User)
    @Delete
    suspend fun deleteList(users: List<User>)

    @Update
    suspend fun update(user: User)
    @Update
    suspend fun updateList(users: List<User>)
}
```

Room database object

- Contains a singleton dbInstance created using **Room.databaseBuilder()**
  - Annotated with **@Database**
  - abstract class that extends RoomDatabase
- Serves as the **main access point** to get DAOs to interact with DB

```
@Database(entities = [Item::class], version = 1)
abstract class ShoppingDB : RoomDatabase() {
    abstract fun getShoppingDao(): ShoppingDao
    // Create a singleton dbInstance
    companion object {
        private var dbInstance: ShoppingDB? = null
        fun getInstance(context: Context): ShoppingDB {
            if (dbInstance == null) {
                dbInstance = Room.databaseBuilder(
                    context,
                    ShoppingDB::class.java, "shopping.db"
                ).build()
            }
            return dbInstance as ShoppingDB
        }
}
```

Observable queries

- Observable queries allow automatic notifications when data changes
  - Notifies the app with of any data updates
- We can accomplish this using **LiveData**, a lifecycle-aware observable value holder
  - We simply **wrap** the return type of our DAO methods with LiveData.

```
// App will be notified of any changes of the Item table data
// Whenever Room detects Item table data change our LiveData
observer will be called with the new list of items
// No need for suspend function as LiveData is already asynchronous
fun getAll() : LiveData<List<Item>>
```

TypeConverter

- Convert a field to column and vice versa
- SQLite has no support for some data types such as Date, enum, BigDecimal etc = TypeConverter is needed

```
class Converter{
    companion object{
        @TypeConverter
        fun fromBigDecimal(value: BigDecimal):String{
            return value.toString()
        }

        @TypeConverter
        fun toBigDecimal(value:String):BigDecimal{
            return value.toBigDecimal()
        }
    }
}
```

1-to-one relationship using @Embedded

- Can be used to model 1-to-1 relationship
- User table will have a houseNumber, street and city columns

```
data class Address(val houseNumber: String,
                   val street: String,
                   val city: String)
@Entity
data class User (
    val firstName: String,
    val lastName: String,
    @Embedded val address: Address
) {
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0
}
```

Relationships

- @Relation is used to model 1-to-many relationships

```
// Entity and its relations are  fetched by Room
@Entity
data class Pet(@PrimaryKey val catId: Long,
               val name: String, val ownerId: Long)

@Entity
data class Owner(@PrimaryKey val id: Long, val name: String) {
    @Relation(parentColumn = "id", entityColumn = "ownerId")
    val pets = listOf<Pet>()
}

@Dao
public interface OwnerDao {
    @Query("SELECT id, name FROM Owner")
    suspend fun getAll() : List<Owner>
}
```

Enforce constraints between entities with foreign keys

```
@Entity(foreignKeys = [
        ForeignKey(entity = Owner::class,
                parentColumns = ["userId"],
                childColumns = ["owner"])
        ])
data class Pet(@PrimaryKey val catId: Long,
               val name: String, val ownerId: Long)

@Entity
data class Owner(@PrimaryKey val id: Long, val name: String) {
    @Relation(parentColumn = "id", entityColumn = "ownerId")
    val pets = listOf<Pet>()
}
```

Summary

Major Components

- **@Entity** - Defines table structure
- **@DAO** - An interface with functions define how to access the database
- **@Database** - Connects all the pieces of Room together

Resources

- Save data in a local database using Room
  - https://developer.android.com/training/data-storage/room
- Room pro tips
  - https://medium.com/androiddevelopers/7-pro-tips-for-room-fbadea4bfbd1
- Room codelab
  - https://codelabs.developers.google.com/codelabs/android-room-with-a-view-kotlin/
  - https://developer.android.com/codelabs/kotlin-android-training-room-database