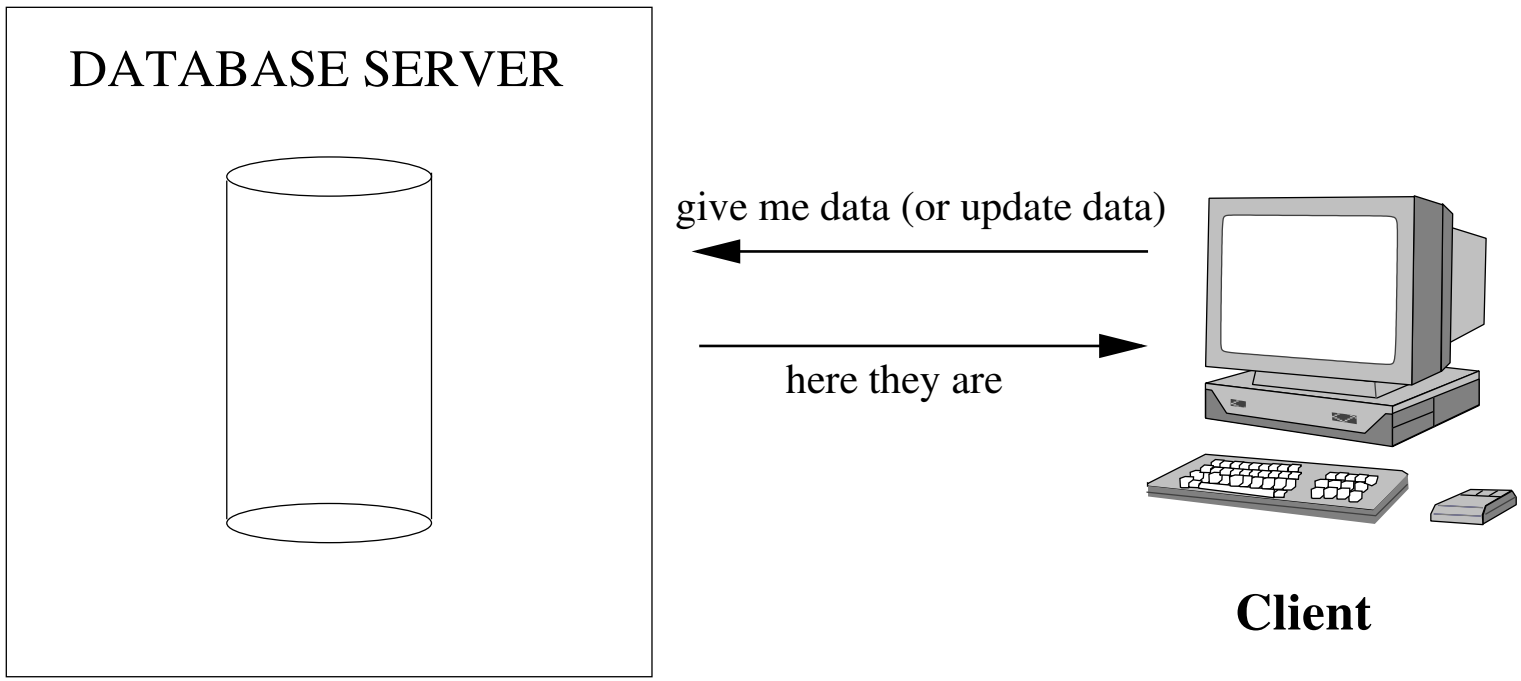
5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 6: Working with Databases
The Room Library

Dr Dimitris C. Dracopoulos

What is a Database Server

Just another server which receives requests from clients requiring access to data in a database (this could be read or write).



Relational Databases

Everything organised into tables.

Name	Age	Position	Salary
John Smith	35	Manager	40000
Robert Barclay	28	Developer	30000
George Deval	25	Administrator	32000
Tom Bubble	38	Head of Sales	45000

Accessing Databases

SQL (Structured Query Language) is used.

The main variations are:

- Transact SQL (T-SQL). Used by Microsoft SQL Server and Sybase. The two have very few differences.
- PL-SQL. Used in Oracle.
- ANSI SQL. Parts of it adopted by commercial and public domain products.

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

The CREATE Statement

Syntax:

```
CREATE TABLE tablename(
    colName  dataType
)
```

Example:

```
CREATE TABLE Person (
   name VARCHAR(100),
   age  INTEGER,
   address VARCHAR(100))
```

The INSERT Statement

Syntax:

```
INSERT INTO tablename
    (colName1, colName2, colName3 ...)
VALUES
    (value1, value2, value3, ...)
```

Example:

```
INSERT INTO Person (name, age, address)
VALUES ('John Smith', 26, 'London'),
       ('Tom Bubble', 34, 'New York')
```

The SELECT Statement

Syntax:

```
SELECT
   Name1, Name2, Name3 ...
FROM tablename1, tablename2, ...
WHERE
   conditions
ORDER BY colNames
```

Example:

```
SELECT Person.name, Person.address,
       ListensTo.music_group_name
FROM Person, ListensTo
WHERE ListensTo.music-group_name IN ('Beatles',
                                     'Popstars')
AND Person.name = ListensTo.person_name
AND Person.address = 'London'
```

The UPDATE Statement

Syntax:

```
UPDATE tablename
  SET colName1=value1, colName2=value2 ...
  WHERE colNamei someOperator valuei
```

Example:

```
UPDATE Person
  SET age = 25, address='Manchester'
  WHERE name = 'John Smith'
```

The DELETE and DROP Statements

Syntax:

```
DELETE FROM tablename
   WHERE colNamei someoperator valuei
```

Example:

```
DELETE FROM Person
   WHERE name = 'John Smith'
```

The rows corresponding to `John Smith` are deleted.

- To delete a whole table (not only the contents but the table itself) use the DROP statement. (after that the table needs to be created again).

Example:

```
DROP TABLE Person
```

The Room Library

It provides a layer on top of SQLite in an attempt to make things easier for the developer.

- Direct SQlite functionality still available.
- Room provides SQL queries check at compile time.
- Once you set it up it is straightforward!

Setting up Room in an Android Studio Project

1. Add the following in the module `build.gradle` file (make sure that you choose the appropriate sections to add the extra stuff):

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-kapt'
}

dependencies {
    implementation("androidx.room:room-runtime:2.4.2")
    annotationProcessor("androidx.room:room-compiler:2.4.2")

    // To use Kotlin annotation processing tool (kapt)
    kapt("androidx.room:room-compiler:2.4.2")

    // optional - Kotlin Extensions and Coroutines support for Room
    implementation("androidx.room:room-ktx:2.4.2")
}
```

2. In the compile options make sure that you specify the correct JDK version for your setup, e.g.:

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_17
    targetCompatibility JavaVersion.VERSION_17
}
kotlinOptions {
    jvmTarget = '17'
}
```

Room - How to Implement

1. Create an Entity class. Each instance represents a row in the corresponding table.
2. Create a DAO (data access object) typically an interface, defining methods corresponding to SQL statements.
3. Create the Database class.
4. Create an instance of the database.
5. Use a DAO object to call methods to execute equivalent SQL statements (instead of directly calling SQL statements)

Creating the Entity Class

File `User.kt`:

```
@Entity
data class User(
    @PrimaryKey val id: Int,

    val firstName: String?,
    val lastName: String?
)
```

Creating the DAO

File `UserDao.kt`:

```
@Dao
interface UserDao {
    @Query("Select * from user")
    suspend fun getAll(): List<User>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUsers(vararg user: User)

    @Insert
    suspend fun insertAll(vararg users: User)
}
```

Creating the Database Class

File AppDatabase.kt:

```
@Database(entities = [User::class], version=1)
abstract class AppDatabase: RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

Usage

In your code:

- Create an instance of the database:

```
val db = Room.databaseBuilder(this, AppDatabase::class.java,
                                    "mydatabase").build()
```

- Create an instance of the DAO object:

```
val userDao = db.userDao()
```

- Call the methods on the DAO object from inside a coroutine.

A Full Example

The layout file `activity_main.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/tv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
  </androidx.constraintlayout.widget.ConstraintLayout>
```

The Entity

The Entity file `User.kt`:

```
package uk.ac.westminster.roomdbexample

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class User(
    @PrimaryKey val id: Int,

    val firstName: String?,
    val lastName: String?
)
```

The DAO

File `UserDao.kt.kt`:

```
package uk.ac.westminster.roomdbexample

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface UserDao {
    @Query("Select * from user")
    suspend fun getAll(): List<User>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUsers(vararg user: User)

    @Insert
    suspend fun insertAll(vararg users: User)
}
```

The Database Class

File AppDatabase.kt:

```
package uk.ac.westminster.roomdbexample

import androidx.room.Database
import androidx.room.RoomDatabase

@Database(entities = [User::class], version=1)
abstract class AppDatabase: RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

The Activity

File `MainActivity.kt`:

```
package uk.ac.westminster.roomdbexample

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.TextView
import androidx.room.Room
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import org.w3c.dom.Text

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val tv = findViewById<TextView>(R.id.tv)
        tv.setText("")

        // create the database
        val db = Room.databaseBuilder(this, AppDatabase::class.java,
                                      "mydatabase").build()
        val userDao = db.userDao()
```

The Activity (cont'ed)

```
        runBlocking {
            launch {
                val user = User(1, "John", "Smith")
                val user2 = User(2, "Helen", "Jones")
                val user3 = User(3, "Mary", "Popkins")
                userDao.insertUsers(user, user2, user3)

                val users: List<User> = userDao.getAll()
                for (u in users) {
                    tv.append("\n ${u.firstName} ${u.lastName}")
                }
            }
        }
    }
```