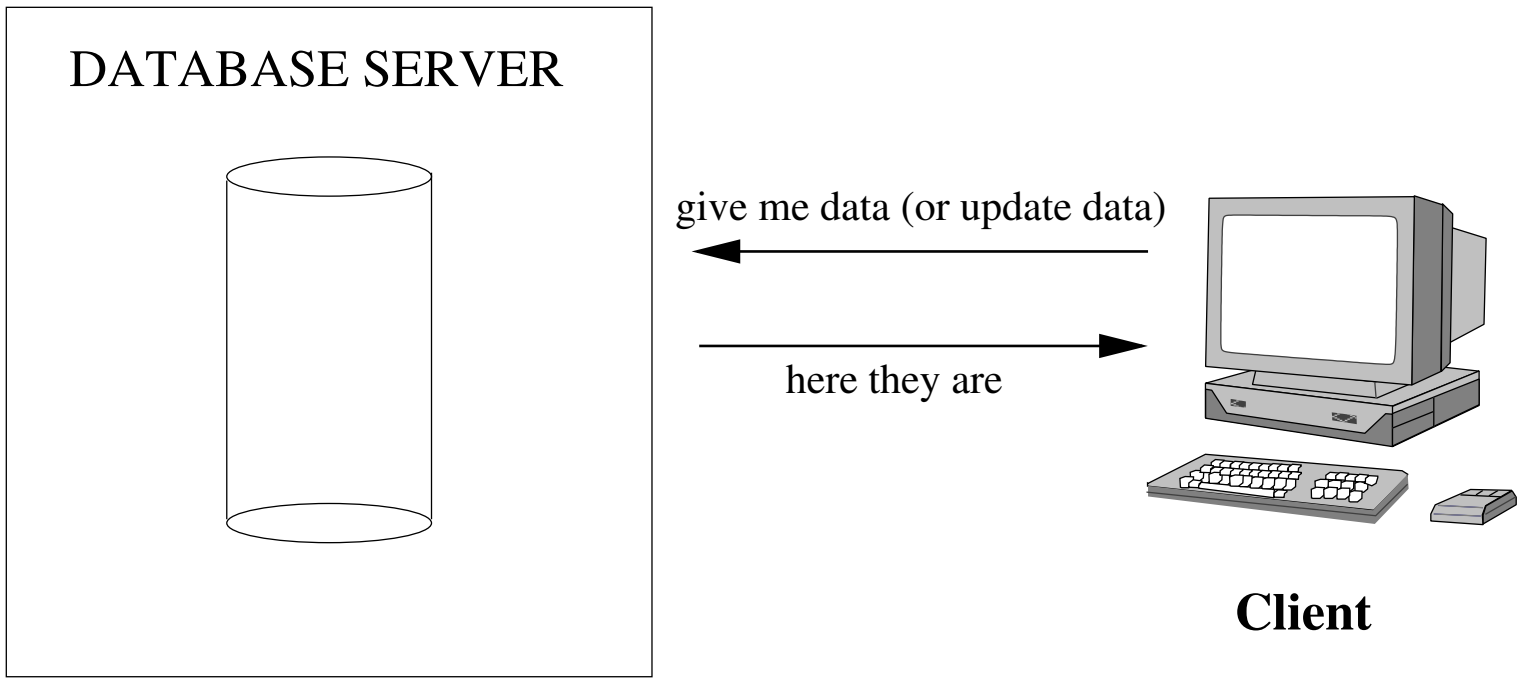
5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 8: Working with Databases
The Room Library

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 `Project build.gradle` file (choose the appropriate versions for your system from `https://github.com/google/ksp/releases`):

```
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.kotlin.compose) apply false

    // declare the KSP plugin
    id("com.google.devtools.ksp") version "2.3.6" apply false
}
```

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

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.compose)

    // enable ksp
    id("com.google.devtools.ksp")
}
```

Setting up Room in an Android Studio Project (cont'd)

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

```
dependencies {
    val room_version = "2.8.4"

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

    // use Kotlin Symbol Processing (KSP)
    ksp("androidx.room:room-compiler:$room_version")

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

4. In the compile options make sure that you specify the correct JDK version and the compose version for your setup (see `https://developer.android.com/jetpack/androidx/releases/compose-kotlin`), e.g.:

```
compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}
buildFeatures {
    compose = true
}
```

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 (read, update, insert, delete).
3. Create the Database class.
4. Create an instance of the database.
5. Obtain a DAO object from the database.
6. Use the DAO object to call methods to execute equivalent SQL statements (instead of directly calling SQL statements)

Creating the Entity Class

File `User.kt` (make sure that you include the correct imports):

```
@Entity
data class User(
    @PrimaryKey(autoGenerate = true) var id: Int = 0,

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

Creating the DAO (Data Access Object)

File `UserDao.kt`:

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

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(vararg users: User)

    // insert one user without replacing an identical one - duplicates allowed
    @Insert
    suspend fun insertUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)

    @Query("select * from user where lastName LIKE :name")
    fun findByLastName(name: String): User

    @Query("delete from user")
    suspend fun deleteAll()
}
```

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:

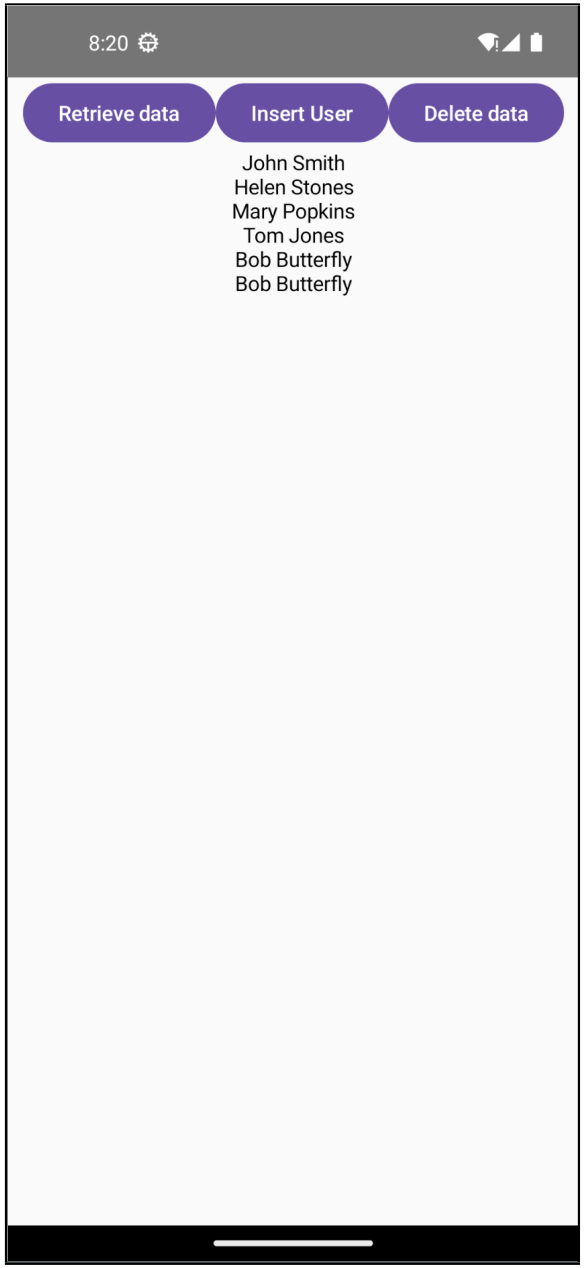
```
val db = Room.databaseBuilder(context,
                              AppDatabase::class.java,
                              "mydatabase").build()
```

- 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



A Full Example (cont'd)

Besides the previously mentioned files, the following is the main activity (imports are omitted):

```
package uk.ac.westminster.roomdbcomposableexampleupdated

/* imports omitted */

lateinit var db: AppDatabase
lateinit var userDao: UserDao

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

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

        setContent {
            GUI()
        }
    }
}
```

A Full Example (cont'd)

```
@Composable
fun GUI() {
    var usersString by remember { mutableStateOf("") }

    /* reinserts the same 4 users every time the "userString"
       changes - just for illustration */
    LaunchedEffect(usersString){
            userDao.insertAll(
                User(1, "John", "Smith"),
                User(2, "Helen", "Stones"),
                User(3, "Mary", "Popkins"),
                User(4, "Tom", "Jones")
            )
    }

    /* obtain the scope of the composable to start a coroutine
       outside the composable (e.g. a button click) */
    val scope = rememberCoroutineScope()
```

A Full Example (cont'd)

```
        Column(modifier = Modifier.fillMaxSize(), horizontalAlignment = Alignment.CenterHorizontally) {
            Row {
                Button(onClick = {
                    scope.launch {
                        usersString = retrieveData(userDao)
                    }
                })
                {
                    Text("Retrieve data")
                }
                Button(onClick = {
                        scope.launch {
                            userDao.insertUser(User(firstName = "Bob", lastName = "Butterfly"))
                            usersString = retrieveData(userDao)
                        }
                }) {
                    Text("Insert User")
                }
                Button(onClick = {
                        scope.launch {
                            userDao.deleteAll()
                            usersString = retrieveData(userDao)
                        }
                }) {
                    Text("Delete data")
                }
            }
            Text(
                modifier = Modifier
                    .fillMaxWidth(),
                textAlign = TextAlign.Center,
                text = usersString
            )
        }
    }
```

A Full Example (cont'd)

```
suspend fun retrieveData(userDao: UserDao): String {
    var allUsers = ""

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

    return allUsers
}
```