5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 9: Working with Databases - Part II

Dr Dimitris C. Dracopoulos

Another Database Example using the Room Library

The application will be used to store products that the user needs to purchase in their next visit to the supermarket.

1. The user interacts with a 2 textboxes entering a product name (e.g. “Cheese”) and its price respectively. There are also 2 buttons displayed in the same screen, labelled `Save to DB` and `Retrieve all`.
2. Every time that the user presses the `Save to DB` button the corresponding product and its price are saved in a table in the database using the Room library.
3. When the user presses the `Retrieve all` button, all the products saved in the database (and the corresponding prices) are displayed.
   The total sum of all (to be purchased) products is also displayed.

The `Product` Class

File `Product.kt`:

```
package uk.ac.westminster.fooddbcomposableexample

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

@Entity
data class Product (
    @PrimaryKey(autoGenerate = true) var id: Int= 0,
    var name: String?,
    var price: Int
)
```

The DAO

File `ProductDao.kt`:

```
package uk.ac.westminster.fooddbcomposableexample

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

@Dao
interface ProductDao {
    @Query("select * from product")
    suspend fun getAll(): List<Product>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(vararg products: Product)
}
```

The AppDatabase Class

File AppDatabase.kt:

```kotlin
package uk.ac.westminster.fooddbcomposableexample

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

@Database(entities = [Product::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun getDao(): ProductDao
}
```

The Main Activity Class

The imports are omitted.

```kotlin
lateinit var db: AppDatabase
lateinit var productDao: ProductDao

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

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

        setContent {
            GUI()
        }
    }
}
```

The Main Activity Class (cont'd)

```
@Composable
fun GUI() {
    var product_name by remember { mutableStateOf("")}
    var product_price by remember{ mutableStateOf("") }
    var results by remember{ mutableStateOf("") }

    var context = LocalContext.current
    val scope = rememberCoroutineScope()

    Column {
        Row (verticalAlignment = Alignment.CenterVertically)
        {
            TextField(modifier = Modifier.padding(5.dp),
                label = { Text("Product name") },
                value = product_name,
                onValueChange = { newText ->
                    product_name = newText
                }
            )

            TextField(placeholder = { Text("Price") },
                value = product_price,
                onValueChange = { newText ->
                        product_price = newText
                }
            )
        }

        Spacer(modifier = Modifier.padding(10.dp))
```

The Main Activity Class (cont'd)

```
Row (modifier = Modifier.fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.Center){

    Button(modifier = Modifier.padding(10.dp),
        onClick = {
            try {
                val price = product_price.toInt()
                scope.launch {
                    productDao.insert(
                        Product(
                            name = product_name,
                            price = price
                        )
                    )
                }
            }
            catch (e: NumberFormatException) {
                Toast.makeText(context,
                               "The price should be a number",
                               Toast.LENGTH_SHORT).show()
           }
        }) {
        Text("Save to DB")
    }
```

The Main Activity Class (cont'd)

```
            Button(onClick = {
                // create a new coroutine to handle the database request
                scope.launch {
                    val products: List<Product> = productDao.getAll()

                    // string that will be displayed, containing all the products
                    results = ""
                    var cost = 0  // total cost of all products

                    for (p in products) {
                        results += "${p.name}, price: ${p.price}\n"
                        cost += p.price
                    }

                    results += "\nTotal cost is: $cost"
                }
            }) {
                Text(text = "Retrieve all")
            }
        }

        Text(modifier = Modifier.fillMaxWidth(), text = results, textAlign = TextAlign.Center)
    }
}
```

LaunchedEffect and rememberCoroutineScope

Android will not allow to make a database request from the main UI thread, as such a request may block the thread.

- Normal Kotlin coroutines may be used.
- However, in Compose, the lifecycle of the composables requires specialised mechanisms for creating the coroutines.
- To call suspend functions safely within a composable the `LaunchedEffect` should be used. This will launch a new coroutine, which will be cancelled when the `LaunchedEffect` leaves the composition. If `LaunchedEffect` is recomposed a new coroutine will be created.
- To call suspend functions outside a composable scope (e.g. the `onClick` of a button) the `rememberCoroutineScope` is used to create a new coroutine bound with a composable function.

Examples of LaunchedEffect and rememberCoroutineScope

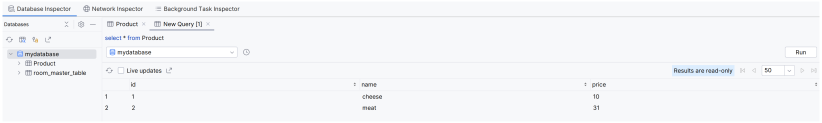
```
@Composable
fun GUI() {
    var usersString by remember { mutableStateOf("") }
    LaunchedEffect(usersString){
            userDao.insertAll(
                User(1, "John", "Smith"),
                User(2, "Helen", "Stones"),
                User(3, "Mary", "Popkins"),
                User(4, "Tom", "Jones")
            )
    }

    val scope = rememberCoroutineScope()
    Button(onClick = {
                    scope.launch {
                        userDao.insertUser(User(firstName = "Bob",
                                                lastName = "Butterfly"))
                    }
    })
}
```

The Database Inspector Tool in Android Studio

Accessible from `View -> Tool Windows -> App Inspection`

- View data in tables
- Run queries on the database



```
select * from Product
```

	id	name	price
1	1	cheese	10
2	2	meat	31

Extending the Products DB Example

1. Display the products retrieved from the DB with a checkbox.
2. The user selects the products wishing to purchase and sees the total cost of the selected products.
3. Add a button which sends an email of the selected products with the total score.

Extending the Products DB Example (cont'd)

Replace the `GUI()` composable function with the following function:

```
@Composable
/* Additional functionality with checkboxes for selected food */
fun GUIExtendedFunctionality() {
    // list of all products in the database and currently being displayed
    var product_name by remember { mutableStateOf("")}
    // list of prices for all products
    var product_price by remember{ mutableStateOf("") }

    var cost by remember{ mutableStateOf(0) }  // total cost

    var displayed_products = remember{ mutableListOf<Product>().toMutableStateList()}
    // whether each checkbox is ticked or not
    var checkStates = remember { mutableListOf<Boolean>().toMutableStateList()}


    var context = LocalContext.current
    val scope = rememberCoroutineScope()

    Column {
        Row (verticalAlignment = Alignment.CenterVertically)
        {
            TextField(modifier = Modifier.padding(5.dp),
                label = { Text("Product name") },
                value = product_name,
                onValueChange = { newText ->
                    product_name = newText
                }
            )
```

Extending the Products DB Example (cont'd)

```
            TextField(placeholder = { Text("Price") },
                value = product_price,
                onValueChange = { newText ->
                    product_price = newText
                }
            )
        }

        Spacer(modifier = Modifier.padding(10.dp))
        Row (modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center){
            Button(modifier = Modifier.padding(10.dp),
                onClick = {
                    try {
                        val price = product_price.toInt()
                        scope.launch {
                            productDao.insert(
                                Product(
                                    name = product_name,
                                    price = price
                                )
                            )
                        }
                    }
                    catch (e: NumberFormatException) {
                      Toast.makeText(context, "The price should be a number",
                                     Toast.LENGTH_SHORT).show()
                    }
                }) {
                Text("Save to DB")
            }
```

Extending the Products DB Example (cont'd)

```
Button(onClick = {
    // create a new coroutine to handle the database request
    scope.launch {
        val products: List<Product> = productDao.getAll()

        // reset the list containing the previously displayed products
        displayed_products.clear()
        checkStates.clear()  // reset the list of checkboxes
        cost = 0  // reset the total cost

        for (i in 0..< products.size) {
            var p = products[i]  // retrieve the actual product

            displayed_products.add(p)
            checkStates.add(false)
        }
    }
}) {
    Text(text = "Retrieve all")
}
```

Extending the Products DB Example (cont'd)

```
    Button(onClick = {
        var message = "Shopping list:\n" +
                      "--------------\n"
        for (i in 0..<checkStates.size) {
            if (checkStates[i])
            message += "${displayed_products[i].name},
                          price: ${displayed_products[i].price}\n"
        }
        // include the cost in the message
        message += "\n\nTotal cost: $cost"

        var i = Intent(Intent.ACTION_SEND)
        i.setType("text/plain")
        i.putExtra(Intent.EXTRA_EMAIL, arrayOf<String>("recipient@gmail.com"))
        i.putExtra(Intent.EXTRA_SUBJECT, "Shopping List")
        i.putExtra(Intent.EXTRA_TEXT, message)
        context.startActivity(i)

    }) {
        Text("Email selected")
    }
}
```

Extending the Products DB Example (cont'd)

```
        for (i in 0..<displayed_products.size) {
            Row (verticalAlignment = Alignment.CenterVertically)
            {
                Checkbox(checked = checkStates[i],
                    onCheckedChange = {
                        checkStates[i] = !checkStates[i]
                        // update the total cost
                        if (checkStates[i])
                            cost += displayed_products[i].price
                        else
                            cost -= displayed_products[i].price

                    })
                    Text(text = ""+displayed_products[i].name +
                                 ", price: ${displayed_products[i].price}")
            }
        }

        // display the total cost of selected products
        Text(modifier = Modifier.fillMaxWidth(),
            text = "Total cost is: $cost",
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center)
    }
}
```