



Reskilling 4Employment Software Developer

Acesso móvel a sistemas de informação

Bruno Santos

bruno.santos.mcv@msft.cesae.pt

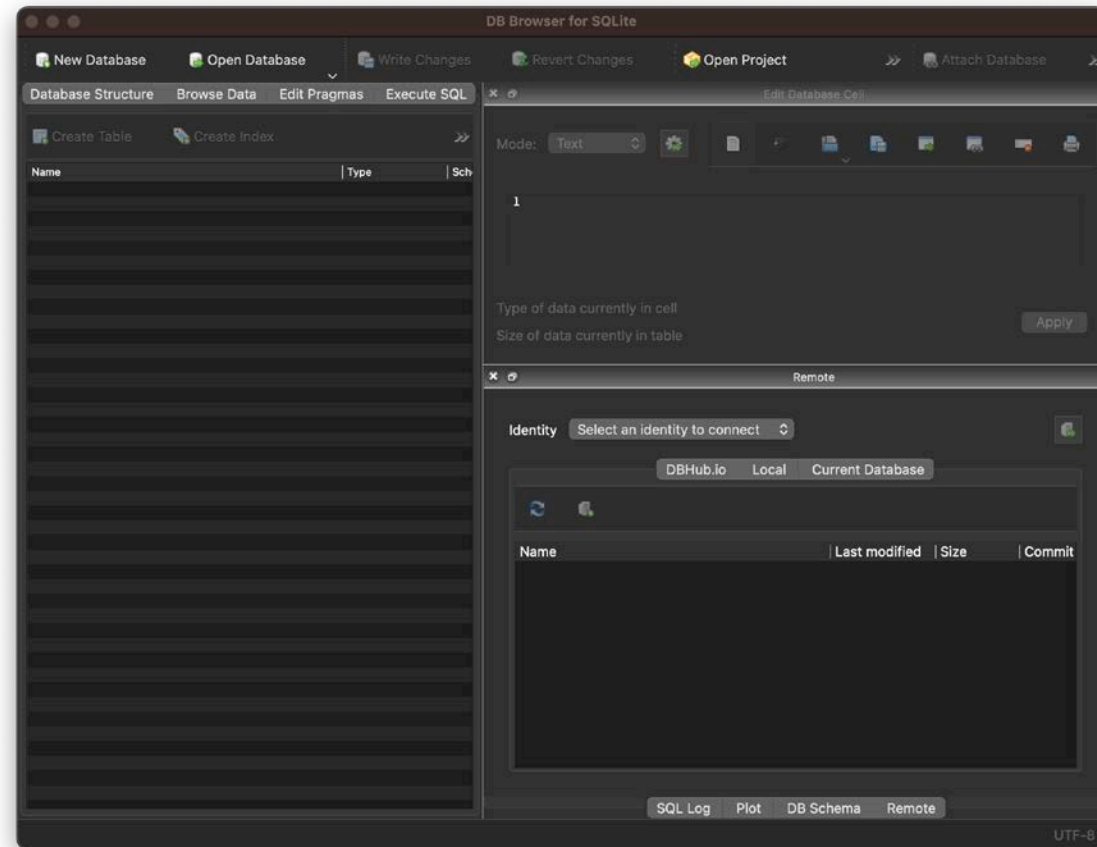
Tópicos

- Base de Dados SQLite

DB Browser

- A aplicação DB Browser (<https://sqlitebrowser.org/>) vai permitir abrir uma base de dados extraída do emulador/dispositivo onde está a ser testada a aplicação

DB Browser



SQLiteOpenHelper

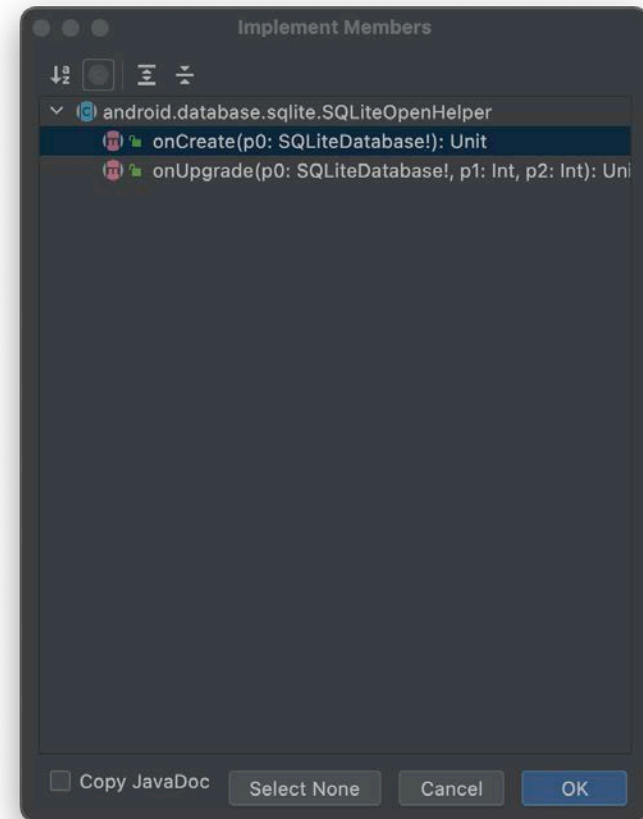
- Após criar um nova aplicação vamos criar um ficheiro Kotlin para manipular base de dados (DBHelper.kt)
- Este ficheiro deve ter a superclasse SQLiteOpenHelper

SQLiteOpenHelper

```
3  import ...
6
7  class DBHelper(context: Context) :
8      SQLiteOpenHelper(context, name: "database.db", factory: null, version: 1) {
9      }
10
```

SQLiteOpenHelper

- De seguida clicamos na lâmpada junto a class DBHelper e selecionamos a opção "Implement Methods", selecionando de seguida ambos os métodos sugeridos:
 - onCreate
 - onUpgrade



SQLiteOpenHelper

```
1 package com.example.a14_basedados
2
3 import ...
4
5
6
7 class DBHelper(context: Context) :
8     SQLiteOpenHelper(context, name: "database.db", factory: null, version: 1) {
9     override fun onCreate(p0: SQLiteDatabase?) {
10         TODO(reason: "Not yet implemented")
11     }
12
13     override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {
14         TODO(reason: "Not yet implemented")
15     }
16 }
```


SQLiteOpenHelper

- Implementamos agora os métodos onCreate e onUpgrade.
 - O método onCreate serve para criar as tabelas e registros iniciais;
 - O método onUpgrade serve para restaurar a base de dados para os valores iniciais ou atualizar para uma nova.
- Em ambos os métodos foi alterada a variável de acesso à base de dados (p0 para db) e retirado o ? que permite que a mesma seja passada nula.
- Foi ainda criado um array com os vários comandos SQL a executar.

SQLiteOpenHelper

```
val sql = arrayOf(
    "CREATE TABLE utilizador (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT)",
    "INSERT INTO utilizador (username, password) VALUES ('user','pass');",
    "INSERT INTO utilizador (username, password) VALUES ('admin','pwd');"
)

override fun onCreate(db: SQLiteDatabase) {
    sql.forEach { it: String
        db.execSQL(it)
    }
}

override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    db.execSQL( sql: "DROP TABLE utilizador")
    onCreate(db)
}
```

Objetos

- Vamos criar a classe Utilizador para permitir criar os objetos dos registos da base de dados

Objetos

```
class Utilizador(val id: Int = 0, val username: String = "", val password: String = "") {  
    override fun toString(): String {  
        return "Utilizador(id=$id, username='$username', password='$password')"  
    }  
}
```

SQLiteOpenHelper

- De seguida devemos criar métodos que nos permitam manipular os elementos constantes na base de dados:
 - Insert
 - Update
 - Delete
 - Select

Insert

- No caso do Insert vamos precisar de uma ligação de escrita à base de dados
- De seguida temos de agrupar os elementos da tabela num ContentValues
- Finalmente executamos a operação utilizando o comando insert que recebe como parâmetros o nome da tabela e o ContentValues (vamos desprezar o nullColumnHack).

Insert

```
fun utilizadorInsert(username: String, password: String): Long {  
    val db = this.writableDatabase  
    val contentValues = ContentValues()  
    contentValues.put("username", username)  
    contentValues.put("password", password)  
    val res = db.insert(table: "utilizador", nullColumnHack: null, contentValues)  
    db.close()  
    return res  
}
```

Update

- O update é semelhante ao Insert com a particularidade de ser necessário indicarmos o campo pelo qual vamos pesquisar o elemento a modificar, assim temos de indicar a whereClause
- Para além disso temos também de enviar o valor que irá substituir o ? que será o último elemento

Update

```
fun utilizadorUpdate(id: Int, username: String, password: String): Int {  
    val db = this.writableDatabase  
    val contentValues = ContentValues()  
    contentValues.put("username", username)  
    contentValues.put("password", password)  
    val res = db.update(table: "utilizador", contentValues, whereClause: "id=?", arrayOf(username))  
    db.close()  
    return res  
}
```

Delete

- O delete é o mais simples dos três uma vez que só temos de indicar qual o elemento a eliminar.

Delete

```
fun utilizadorDelete(id: Int): Int {  
    val db = this.writableDatabase  
    val res = db.delete(table: "utilizador", whereClause: "id=?", arrayOf(id.toString()))  
    db.close()  
    return res  
}
```

Select

- Quando queremos selecionar elementos de uma tabela podemos querer fazê-lo de duas formas:
 1. Selecionar todos os elementos;
 2. Selecionar apenas alguns elementos de acordo com uma ou mais condições;
- Nos dois casos iremos usar ligações à base de dados apenas de leitura pois não vamos alterar qualquer valor na mesma.

Select

- No primeiro caso vamos selecionar todos os registos da tabela Utilizador
- Quando executamos a chamada à base de dados o comando `rawQuery` devolve um objeto do tipo `Cursor` com o resultado. Vamos criar métodos para devolver à Activity o `Cursor` ou o objeto/lista de objetos

Select

```
fun utilizadorSelectAll(): Cursor {  
    val db = this.readableDatabase  
    val c = db.rawQuery( sql: "SELECT * FROM utilizador", selectionArgs: null)  
    db.close()  
    return c  
}
```

Select

```
fun utilizadorListSelectAll(): ArrayList<Utilizador> {  
    val db = this.readableDatabase  
    val c = db.rawQuery( sql: "SELECT * FROM utilizador", selectionArgs: null)  
    val listaUtilizador: ArrayList<Utilizador> = ArrayList()  
    if (c.count > 0) {  
        c.moveToFirst()  
        do {  
            val idIndex = c.getColumnIndex("id")  
            val usernameIndex = c.getColumnIndex("username")  
            val passwordIndex = c.getColumnIndex("password")  
            val id = c.getInt(idIndex)  
            val username = c.getString(usernameIndex)  
            val password = c.getString(passwordIndex)  
            listaUtilizador.add(Utilizador(id, username, password))  
        } while (c.moveToNext())  
    }  
    db.close()  
    return listaUtilizador  
}
```

Select

- No segundo caso vamos filtrar a seleção pelo username

Select

```
fun utilizadorSelectByID(id: Int): Cursor {  
    val db = this.readableDatabase  
    return db.rawQuery( sql: "SELECT * FROM utilizador WHERE id=?", arrayOf(id.toString()))  
}
```

Select

```
fun utilizadorObjectSelectByID(id: Int): Utilizador {  
    val db = this.readableDatabase  
    val c = db.rawQuery("SELECT * FROM utilizador WHERE id=?", arrayOf(id.toString()))  
    var utilizador = Utilizador()  
    if (c.count == 1) {  
        c.moveToFirst()  
        val idIndex = c.getColumnIndex("id")  
        val usernameIndex = c.getColumnIndex("username")  
        val passwordIndex = c.getColumnIndex("password")  
        val id = c.getInt(idIndex)  
        val username = c.getString(usernameIndex)  
        val password = c.getString(passwordIndex)  
        utilizador = Utilizador(id, username, password)  
    }  
    return utilizador  
}
```

Base de Dados

- Para recuperar os registos da base de dados e, por exemplo, apresentá-los numa ListView devemos criar uma Lista de elementos e convertê-la para a ListView, assim, na Activity vamos criar as variáveis para a Lista e Base de Dados.
- De seguida chamamos a função `utilizadorListSelectAll`

Base de Dados

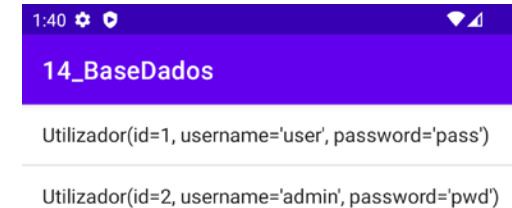
```
private lateinit var binding: ActivityMainBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val db = DBHelper(context: this)

    val listaUtilizadores = db.utilizadorListSelectAll()

    binding.listView.adapter =
        ArrayAdapter(context: this, android.R.layout.simple_list_item_1, listaUtilizadores)
}
```



1:40 [Settings] [Signal] [Battery]

14_BaseDados

Utilizador(id=1, username='user', password='pass')

Utilizador(id=2, username='admin', password='pwd')

Exercício 1

- Crie uma aplicação com três Activity:
 - LoginActivity;
 - RegistoActivity;
 - MinhaContaActivity.
- Nota: implemente uma classe DBHelper para fazer a ligação à base de dados. A classe Utilizador para criar os objetos é facultativa uma vez que não será apresentada qualquer listagem.

Exercício 1

- Na LoginActivity deve ser criado um formulário de login com username e password para o utilizador se autenticar, deve também existir um botão para o utilizador passar para a RegistoActivity e criar um novo registo.

Exercício 1

- Na RegistoActivity deve ser criado um formulário com username e password para o utilizador se registar.
- Nota: deve ser validado o username e/ou password vazios e username já registado.

Exercício 1

- Na `MinhaContaActivity` devem ser apresentados username e password do utilizador e permitir ao mesmo alterar a password e eliminar a conta.
- Nota: se quiser pode criar mais Activity para dividir as operações.

Exercício 2

- Crie uma nova aplicação que guarde um conjunto de alunos na base de dados.
- Deve criar uma classe Aluno com os atributos nome e email. Pode acrescentar outros atributos que considere necessários.
- Apresente todos os alunos numa ListView apresentando-os pelo nome.
- Apresente juntamente com a lista o número de alunos presentes na tabela da base de dados.
- Crie mecanismos para inserir, alterar e eliminar alunos.