IMD0509 - DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS

Professor Emerson Alencar
emerson@imd.ufrn.br

SQLITE

- Banco de Dados SQLite:
  - Site: www.sqlite.org
  - Biblioteca que implementa um banco de dados transacional, autocontido, sem necessidade de servidor e nem de configurações:
    - Suporte a transações: Permite executar um conjunto de operações e só efetivá-las se todas forem bem-sucedidas;
    - Autocontido: Dependências mínima do SO
    - Trabalha localmente, o aplicativo acessa diretamente o sistema de arquivos ;
    - Não precisa ser instalado ou configurado

SQLITE

- Possui diversos recursos existentes em SGBDs, tais como:
  - Tabelas,
  - Chaves primárias e estrangeiras,
  - Views,
  - Índices,
  - Suporte a transações e triggers.

SQLITE

Tipo	Descrição
INTEGER	Inteiro sinalizado
REAL	Ponto Flutuante de 64 bits
TEXT	Textos em UTF-8, UTF-16BE ou UTF-16LE
BLOB	Arrays de Bytes

DEFININDO O BANCO NA APLICAÇÃO

DBConstants.kt

```kotlin
package com.example.projetosqlite.repository.sqlite

const val DATABASE_NAME = "dbprodutos"
const val DATABASE_VERSION = 1
const val TABLE_NAME= "produto"
const val COLUMN_ID = "_id"
const val COLUMN_NAME = "nome"
const val COLUMN_VALUE = "valor"

```

- Arquivo com as constantes do banco

DEFININDO O BANCO NA APLICAÇÃO

ProdutoSqlHelper.kt

```kotlin
package com.example.projetosqlite.repository.sqlite

import ...

class ProdutoSqlHelper(context: Context):
    SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION){

    override fun onCreate(sqLiteDatabase: SQLiteDatabase) {

        sqLiteDatabase.execSQL(
            sql: "CREATE TABLE $TABLE_NAME("+
                    "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "+
                    "$COLUMN_NAME TEXT NOT NULL, "+
                    "$COLUMN_VALUE REAL)")
    }

    override fun onUpgrade(sqLiteDatabase: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        //Próximas versões
    }
}

```

- Define um classe que herde de SQLiteOpenHelper, responsável por criar o banco de dados

DEFININDO O BANCO NA APLICAÇÃO

ProdutoRepository.kt

```kotlin
package com.example.projetosqlite.repository.sqlite

import com.example.projetosqlitekotlin.model.Produto


interface ProdutoRepository {

    fun save(produto: Produto)
    fun remove(vararg produtos: Produto)
    fun produtoById(id: Long, callback: (Produto?) -> Unit)
    fun search(term: String, callback: (List<Produto>) -> Unit)

}
```

- Interface ProdutoRepository

DEFININDO O BANCO NA APLICAÇÃO

```
class SQLiteRepository(ctx: Context): ProdutoRepository {

    private  val helper: ProdutoSqlHelper = ProdutoSqlHelper(ctx)

    private fun insert(produto: Produto){

        val db = helper.writableDatabase

        val cv = ContentValues().apply { this: ContentValues
            put(COLUMN_NAME, produto.nome)
            put(COLUMN_VALUE, produto.valor)
        }

        val id = db.insert(TABLE_NAME,  nullColumnHack: null, cv)
        if (id != -1L){
            produto.id = id
        }
        db.close()
    }
```

- Método de inserir
- Chamadas a propriedade writetableDatabase obtém uma instância de SQLiteDatabase

DEFININDO O BANCO NA APLICAÇÃO

```
50
51      private fun update(produto: Produto){
52          val db = helper.writableDatabase
53
54          val cv = ContentValues().apply { this: ContentValues
55              put(COLUMN_NAME, produto.nome)
56              put(COLUMN_VALUE, produto.valor)
57          }
58
59          db.update(
60              TABLE_NAME,
61              cv,
62               whereClause: "$COLUMN_ID = ? ",
63              arrayOf(produto.id.toString())
64          )
65
66          db.close()
67
68      }
```

- Método de Atualizar

DEFININDO O BANCO NA APLICAÇÃO

```
70
71      override fun save(produto: Produto) {
72          if (produto.id == 0L){
73              insert(produto)
74          }else{
75              update(produto)
76          }
77      }
78
79      override fun remove(vararg produtos: Produto) {
80          val db = helper.writableDatabase
81          for (produto in produtos){
82              db.delete(
83                  TABLE_NAME,
84                   whereClause: "$COLUMN_ID = ? ",
85                  arrayOf(produto.id.toString())
86              )
87          }
88          db.close()
89      }
```

- Métodos Salvar e remover

DEFININDO O BANCO NA APLICAÇÃO

```
91    override fun produtoById(id: Long, callback: (Produto?) -> Unit) {
92        val sql = "SELECT * FROM $TABLE_NAME WHERE $COLUMN_ID = ? "
93        val db = helper.writableDatabase
94        val cursor = db.rawQuery(sql, arrayOf(id.toString()))
95        val produto = if (cursor.moveToNext())produtoFromCursor(cursor) else null
96
97        callback(produto)
98
99    }
```

- Buscar produto

DEFININDO O BANCO NA APLICAÇÃO

```
100
101     override fun search(term: String, callback: (List<Produto>) -> Unit) {
102         var sql = "SELECT * FROM $TABLE_NAME"
103         var args: Array<String>? = null
104
105         if (term.isNotEmpty()){
106             sql += " WHERE $COLUMN_NAME LIKE ?"
107             args = arrayOf("%$term%")
108         }
109
110         sql += " ORDER BY $COLUMN_NAME"
111         val db = helper.readableDatabase
112         val cursor = db.rawQuery(sql, args)
113         val produtos = ArrayList<Produto>()
114         while (cursor.moveToNext()){
115             val produto = produtoFromCursor(cursor)
116             produtos.add(produto)
117         }
118         cursor.close()
119         db.close()
120         callback(produtos)
121     }
```

- Buscar produtos

DEFININDO O BANCO NA APLICAÇÃO

```
122
123     private fun produtoFromCursor(cursor: Cursor):Produto{
124         val id = cursor.getLong(cursor.getColumnIndex(COLUMN_ID))
125         val name = cursor.getString(cursor.getColumnIndex(COLUMN_NAME))
126         val value = cursor.getFloat(cursor.getColumnIndex(COLUMN_VALUE))
127
128         return Produto(id, name, value)
129     }
130
```

- Função que recebe um curso e retorna um produto

Obrigado!

Dúvidas?

Professor Emerson Alencar
emerson@imd.ufrn.br