

# SQLITE

no mundo mobile

---



# GRUPO

**Scrum Master - Victor Oliveira Silva - 24007940**

**Gerente de configuração - Ravi Menezes Silva Leal - 24010140**

**Analista desenvolvedor - Hítallo Jean de Holanda Freitas - 01620638**

**Analista de dados e negócio - Matheus Soares Lemos - 01608078**



# O que é SQLite

- **SQLite é uma biblioteca escrita em C que implementa uma Engine SQL capaz de suportar SQL de forma completa, auto-contida, rápida e com mínima pegada de memória.**



# Principais Aplicações do SQLite

- SQLite, com baixa pegada de memória, suporte completo a SQL e alta performance, é a engine de banco de dados mais usada no mundo, presente em dispositivos móveis, sistemas operacionais, navegadores, apps desktop (Skype, iTunes, DropBox), TVs e sistemas automotivos



# Uso do SQLite em Dispositivos Móveis

- O SQLite é usado para persistência de dados diretamente no dispositivo, sem a necessidade de internet. Em sistemas móveis, ele armazena configurações do sistema operacional e registros de pacotes. Em aplicativos, pode salvar dados apenas localmente ou de forma sincronizada, armazenando localmente sem internet e sincronizando com um banco remoto quando a conexão for restabelecida.



# Funcionamento do SQLite

- O SQLite armazena todos os dados em um único arquivo, incluindo esquema e dados do usuário. Esse arquivo é estável, compatível com qualquer plataforma e pode ser facilmente comprimido e transferido para sincronizações e agregações de dados.



# Exemplos Práticos do Uso do SQLite

- Aplicativos como Minhas Finanças usam o SQLite para armazenar dados localmente, como contas, despesas e relatórios financeiros. No plano Sync, ele combina SQLite com um banco remoto para sincronizar dados entre dispositivos.
- Navegadores de internet (Chrome, Firefox): Utilizam o SQLite para armazenar histórico de navegação, cookies e cache de dados.
- Aplicativos de Mensagens (WhatsApp): Usam o SQLite para persistir localmente as conversas, mídias e contatos.
- Jogos Mobile: Armazenaam o progresso dos jogadores, configurações e rankings locais no SQLite.
- Aplicativos de Mídia (Spotify): Utilizam o SQLite para gerenciar playlists, histórico de músicas ou podcasts baixados para uso offline.



# scan/index.js

```
1 const fs = require('fs-extra');
2 const path = require('path');
3 const sqlite3 = require('sqlite3').verbose();
4
5 let dirPath = '.'
6
7 const args = process.argv.slice(2);
8 if (args.length === 1) {
9 | dirPath = args[0];
10 }
11
12 // Database setup
13 const db = new sqlite3.Database('./files.db');
14
15 // Create a table if it doesn't exist
16 db.serialize(() => {
17 | db.run(`
18 |   CREATE TABLE IF NOT EXISTS files (
19 |     id INTEGER PRIMARY KEY AUTOINCREMENT,
20 |     name TEXT NOT NULL,
21 |     path TEXT NOT NULL,
22 |     size INTEGER NOT NULL,
23 |     extension TEXT,
24 |     created_at TEXT NOT NULL
25 |   )
26 | );
27
28 db.run(`
29 |   CREATE INDEX IF NOT EXISTS idx_extension ON files (extension)
30 | );
31 });
32
```

- **fs-extra (manipulação de arquivos)**
- **path (gerenciamento de caminhos)**
- **sqlite3 (interação com SQLite)**
- **let dirPath = '.';**  
(define o diretório padrão como o diretório atual)
- **const args**  
(Se o usuário passar um diretório na linha de comando, ele será usado como dirPath. Se não, dirPath mantém o valor original).
- **const db**  
(cria ou abre um banco de dados SQLite chamado files.db)
- **db.serialize(() => { ... }):**  
(Garante que as operações dentro da função sejam executadas de forma sequencial)

# scan/index.js

```
33 // Function to insert file metadata into the database
34 const insertFileMetadata = (name, path, size, extension, createdAt) => {
35   db.run(`|
36     INSERT INTO files (name, path, size, extension, created_at)
37     VALUES (?, ?, ?, ?, ?)`
38   , [name, path, size, extension, createdAt], function (err) {
39     if (err) {
40       console.error('Error inserting file data:', err.message);
41     } else {
42       console.log(`File ${name} inserted with id ${this.lastID}`);
43     }
44   );
45 };
46
47 const getFileExtension = (name) => {
48   const splitFileName = name.split('.');
49   if (splitFileName.length === 1 || (splitFileName[0] === "" && splitFileName.length === 2)) {
50     return "";
51   }
52   return splitFileName.pop().toLowerCase();
53 }
```

## • const processDirectory

função chamada processDirectory, que processa um diretório e seus arquivos, e insere os metadados de cada arquivo no banco de dados.

## • processDirectory(dirPath)

executa a função processDirectory e lida com o fechamento do banco de dados, garantindo que ele seja fechado corretamente, independentemente do sucesso ou falha ao processar o diretório.

## • const insertFileMetadata

(função chamada insertFileMetadata que insere os metadados de um arquivo na tabela files)

### • DB.RUN

(Executa uma instrução SQL para inserir os dados do arquivo na tabela files.)

### • const getFileExtension

(função chamada getFileExtension, que extrai a extensão de um nome de arquivo.)

# scan/index.js

```
55 // Function to process files in a directory
56 const processDirectory = async (dir) => {
57   try {
58     const files = await fs.readdir(dir);
59     for (const fileName of files) {
60       const filePath = path.join(dir, fileName);
61       const stats = await fs.stat(filePath);
62       if (stats.isFile()) {
63         insertFileMetadata(fileName, path.resolve(fileName), stats.size, getFileExtension(fileName), stats.birthtime.toISOString());
64       } else if (stats.isDirectory()) {
65         console.log(filePath)
66         await processDirectory(filePath);
67       }
68     }
69   } catch (err) {
70     console.error('Error reading directory:', err.message);
71   }
72 };
73
74 processDirectory(dirPath)
75   .then(() => {
76     db.close();
77   })
78   .catch(err => {
79     console.error('Error processing directory:', err.message);
80     db.close();
81   });
82
```

# read/index.js

```
1 const sqlite3 = require('sqlite3').verbose();
2
3 // Open the database
4 const db = new sqlite3.Database('./files.db');
5
6 // Function to read all data from the files table
7 const readData = () => {
8     return new Promise((resolve, reject) => {
9         db.all('SELECT * FROM files', [], (err, rows) => {
10            if (err) {
11                reject(err);
12            } else {
13                resolve(rows);
14            }
15        });
16    });
17}
18
19 // Read data and display it
20 readData()
21 .then(rows => {
22     for (const row of rows) {
23         console.table(row);
24     }
25 })
26 .catch(err => {
27     console.error('Error reading data from database:', err.message);
28 })
29 .finally(() => {
30     db.close();
31 })
32
```

- **sqlite3 = require('sqlite3').verbose();**

(Importa o módulo SQLite com o modo "verbose", o que fornece mensagens detalhadas para debug).

- **const db = new sqlite3.Database('./files.db')**

(Abre ou cria o banco de dados files.db.)

- **readData():**

(Retorna uma promessa que busca todos os dados da tabela files usando a query SELECT \* FROM files.)