📚 API Livraria com Express.js — Parte 6 (SQLite: sqlite3 e better-sqlite3)

Migrando de arquivo JSON para banco SQLite

👨‍🏫 **Professor:** Fabricio Bizotto

📘 **Disciplina:** Desenvolvimento Web I

🎓 **Curso:** Ciência da Computação

📅 **Fase:** 4ª fase

Roteiro

- Por que SQLite e `better-sqlite3`
- Instalação das dependências
- Variáveis de ambiente (.env)
- Módulo de banco centralizado (`src/database/sqlite.js`)
- Inicialização no `app`
- CRUD no Repositório (SQL)
- Seed opcional a partir do JSON
- Como rodar e testar
- Erros comuns e dicas

Por que SQLite + better-sqlite3?

- Substitui JSON local por um banco leve e confiável
- Não requer servidor externo (arquivo `.db`)
- `better-sqlite3` é simples e performático (API síncrona)
- Centralizamos acesso ao banco e facilitamos evolução futura

> Também instalamos `sqlite3` para referência/compatibilidade, mas a aplicação usa `better-sqlite3` no runtime.

Dependências e instalação

```
npm i better-sqlite3 sqlite3
```

Observações:

- O `better-sqlite3` geralmente oferece binários pré-compilados.
- Em alguns ambientes, pode compilar nativamente (precisa de `python`, `gcc`, `make`).

Variáveis de ambiente (opcional)

Arquivo `.env` (raiz):

```
SQLITE_DB_FILE=/caminho/absoluto/livraria.db
PORT=3000
NODE_ENV=development
```

Se não definir `SQLITE_DB_FILE`, usamos o padrão: `src/data/livraria.db`.

Estrutura atualizada do projeto

```
livraria_node_http/
├─ server.js
├─ package.json
└─ src/
   ├─ app.js
   ├─ database/
   │  └─ sqlite.js    <-- NOVO MÓDULO DE DB
   ├─ repositories/
   │  └─ livros.repository.js (CRUD SQL)
   ├─ models/
   │  └─ livro.model.js
   ├─ routes/
   └─ data/
      └─ livraria.db  (gerado)
```

src/database/sqlite.js

```
const path = require('path');
const fs = require('fs');
const Database = require('better-sqlite3');

const DB_FILE = process.env.SQLITE_DB_FILE || path.join(__dirname, '../data/livraria.db');
fs.mkdirSync(path.dirname(DB_FILE), { recursive: true });

let db; // singleton
function getDb() {
  if (!db) {
    db = new Database(DB_FILE);
    db.pragma('foreign_keys = ON');
  }
  return db;
}

function run(sql, params = []) { return getDb().prepare(sql).run(...params); }
function get(sql, params = []) { return getDb().prepare(sql).get(...params); }
function all(sql, params = []) { return getDb().prepare(sql).all(...params); }

function init() {
  run(`CREATE TABLE IF NOT EXISTS livros (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    titulo TEXT NOT NULL,
    autor TEXT NOT NULL,
    categoria TEXT NOT NULL,
    ano INTEGER NOT NULL
  )`);
  console.log('Banco de dados SQLite inicializado');
}
module.exports = { getDb, run, get, all, init };
```

Inicialização do banco no app

```
// src/app.js
const app = require("./config/express");
const db = require("./database/sqlite");
db.init(); // garante que a tabela exista antes das rotas

const routes = require("./routes");
const errorHandler = require("./middlewares/errorHandler");
app.use("/api", routes);
app.use(errorHandler);
```

Repositório: CRUD com SQL

```
// src/repositories/livros.repository.js
const Livro = require("../models/livro.model");
const db = require("../database/sqlite");

class LivrosRepository {
  async findAll() {
    const rows = await db.all("SELECT id, titulo, autor, categoria, ano FROM livros ORDER BY id ASC");
    return rows.map(r => Livro.fromJSON(r));
  }
  async findById(id) {
    const row = await db.get("SELECT id, titulo, autor, categoria, ano FROM livros WHERE id = ?", [id]);
    return row ? Livro.fromJSON(row) : null;
  }
  async create(data) {
    const novo = new Livro({ id: null, ...data });
    const res = await db.run(
      "INSERT INTO livros (titulo, autor, categoria, ano) VALUES (?, ?, ?, ?)",
      [novo.titulo, novo.autor, novo.categoria, novo.ano]
    );
    return this.findById(res.id);
  }
}
module.exports = LivrosRepository;
```

Repositório: CRUD com SQL

```
// src/repositories/livros.repository.js
const Livro = require("../models/livro.model");
const db = require("../database/sqlite");

class LivrosRepository {
  async update(id, dados) {
    const atual = new Livro({ id, ...dados });
    await db.run(
      "UPDATE livros SET titulo = ?, autor = ?, categoria = ?, ano = ? WHERE id = ?",
      [atual.titulo, atual.autor, atual.categoria, atual.ano, id]
    );
    return this.findById(id);
  }
  async delete(id) {
    const existente = await this.findById(id);
    if (!existente) {
      const e = new Error("Livro não encontrado");
      e.statusCode = 404; throw e;
    }
    await db.run("DELETE FROM livros WHERE id = ?", [id]);
    return existente;
  }
}
module.exports = LivrosRepository;
```

GitIgnore atualizado

```
# SQLite database file
src/data/*.db
```

> O arquivo `.db` gerado não deve ser versionado.

Rodando e testando

```
# desenvolvimento
npm run dev

# produção/local simples
npm start
```

Resumo das mudanças

- Módulo `src/database/sqlite.js` (singleton + helpers + `init()`)
- Inicialização no `app` (`db.init()`)
- Repositório usando SQL (CRUD completo)

Desafios extras

- Acrescentar as colunas faltantes (`editora`, `numeroPaginas`)
- Testar com Postman