INSTITUTO FEDERAL CATARINENSE - CAMPUS SOMBRIO
TÉCNICO INTEGRADO EM INFORMÁTICA PARA INTERNET

Lucas Macedo de Jesus
Vinícius Schmoller de Limas

Trabalho sobre ORM e Sequelize

Sombrio,
2025

Introdução

Com o avanço do desenvolvimento web, a forma como lidamos com dados armazenados em bancos relacionais precisou evoluir. No passado, desenvolvedores escreviam consultas SQL manualmente para interagir com o banco de dados. Embora isso oferecesse controle total, também exigia muito conhecimento específico e aumentava a chance de erros. Foi nesse cenário que surgiram os ORMs (Object-Relational Mappers), ferramentas que atuam como uma ponte entre o mundo orientado a objetos da programação e o modelo relacional dos bancos de dados tradicionais. Este trabalho propõe-se a explorar essa tecnologia por meio do Sequelize, um dos ORMs mais usados no ecossistema Node.js, abordando desde seus fundamentos até a análise crítica do seu uso.

Fundamentos dos ORMs

ORM é a sigla para Object-Relational Mapping, ou Mapeamento Objeto-Relacional, e trata-se de uma técnica que permite representar tabelas e registros de um banco de dados como objetos em uma linguagem de programação. Isso significa que o desenvolvedor pode interagir com o banco usando código JavaScript, por exemplo, sem precisar escrever SQL diretamente. Em vez de escrever SELECT * FROM usuarios, o programador escreve Usuario.findAll().

Mas o surgimento dos ORMs também está relacionado a um problema clássico da computação: o chamado “Paradigma da Impedância Objeto-Relacional”. Esse nome complicado representa algo bem prático. Linguagens de programação usam estruturas orientadas a objetos (com classes, heranças, métodos etc.), enquanto bancos relacionais organizam dados em tabelas com colunas e linhas. Traduzir uma estrutura para a outra não é simples nem direto. O ORM tenta resolver justamente esse desafio, criando uma camada de abstração que converte os dados automaticamente entre esses dois mundos. Dessa forma, o código fica mais limpo, organizado e fácil de manter.

Sequelize em Detalhes

Entre as diversas ferramentas ORM disponíveis para JavaScript, o Sequelize se destaca por sua simplicidade e ampla adoção. Ele é construído sobre Promises, o que facilita sua integração com aplicações modernas baseadas em Node.js. O Sequelize suporta diversos bancos de dados como PostgreSQL, MySQL, SQLite, MariaDB e MSSQL. Ele permite que desenvolvedores definem models que representam tabelas, realizem associações entre essas entidades e realizem operações como inserções, buscas, atualizações e exclusões usando apenas JavaScript.

Por exemplo, podemos definir um model chamado Produto com os campos id, nome e preco da seguinte forma:

```
const Produto = sequelize.define('Produto', {

  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  nome: { type: DataTypes.STRING },
  preco: { type: DataTypes.DECIMAL }
});
```

Além de representar uma tabela, o Sequelize também permite relacionar os models uns com os outros. Uma empresa pode ter um funcionário, um funcionário pode pertencer a uma empresa, e dois usuários podem participar de vários grupos. O Sequelize traduz essas relações em código com métodos como hasOne, hasMany, belongsTo e belongsToMany, sempre respeitando as lógicas do banco de dados relacional.

Outro ponto importante é que o Sequelize abstrai as consultas SQL. Por exemplo, ao invés de escrever uma query SQL para buscar todos os produtos, usamos Produto.findAll(). Para buscar um item pelo ID, usamos Produto.findByPk(1), e, se quisermos aplicar uma condição, como preco < 10, fazemos isso com um where. A busca de dados relacionados também pode ser feita com o método include, que funciona como um JOIN. Assim, a sintaxe fica muito mais intuitiva e próxima do JavaScript puro.

Tópicos Avançados e Boas Práticas

Ao começar a usar ORMs, muitos desenvolvedores recorrem ao método sequelize.sync({ force: true }) para criar ou sincronizar as tabelas automaticamente. No entanto, esse método remove todas as tabelas e as recria do zero. Em produção, isso pode significar perda total de dados. Por isso, uma alternativa mais segura são as migrations.

Migrations são arquivos de versionamento que registram todas as alterações feitas nas tabelas: criação, remoção ou modificação de colunas, chaves estrangeiras, entre outros. O Sequelize CLI permite gerar migrations com comandos como npx sequelize-cli migration:generate, aplicar com db:migrate e desfazer com db:migrate:undo. Esse controle dá segurança e previsibilidade à evolução do banco de dados.

Outro recurso avançado e fundamental são as transações. Elas garantem que um conjunto de operações no banco aconteça como um bloco único e atômico: ou tudo acontece, ou

nada acontece. Imagine uma compra online. Se o produto for debitado do estoque, mas o pagamento falhar, a operação precisa ser revertida. Com Sequelize, é possível criar transações que agrupam múltiplas operações usando sequelize.transaction().

```
const t = await sequelize.transaction();

try {

  await Produto.create({ nome: 'Pão', preco: 2 }, { transaction: t });

  await Estoque.create({ produto_id: 1, quantidade: 100 }, { transaction: t });

  await t.commit();

} catch (err) {

  await t.rollback();

}
```

Esse tipo de controle é essencial para garantir a integridade dos dados.

Análise Crítica e Comparativa

O uso de ORMs como o Sequelize traz muitas vantagens. A primeira delas é a produtividade. Com menos código SQL para escrever e manter, o desenvolvedor pode se concentrar mais na lógica da aplicação. Outro benefício é a portabilidade. Como o Sequelize suporta vários tipos de bancos de dados, é possível trocar de banco com poucas alterações no código. Além disso, ele permite manter um padrão de código mais organizado e orientado a objetos.

Por outro lado, nem tudo são flores. Um dos principais problemas dos ORMs é a performance. Consultas simples funcionam bem, mas operações complexas podem gerar SQL ineficientes ou até mesmo difíceis de depurar. Outro ponto é a curva de aprendizado: entender como definir corretamente associações, migrations, tipos e restrições pode levar tempo. E, em projetos muito exigentes em performance, a camada de abstração criada pelo ORM pode "vazar", ou seja, esconder problemas que só aparecem quando o sistema já está em produção.

Existem cenários em que o uso de ORM não é recomendado. Em aplicações que exigem consultas altamente otimizadas ou que trabalham com lógica muito específica de banco de dados, como análises estatísticas complexas, é preferível usar SQL puro ou ferramentas como Knex.js, um query builder que oferece mais controle sobre a geração das consultas. Nessas situações, a simplicidade do Sequelize pode se tornar uma limitação.

Comparando com outras ferramentas, por exemplo o Knex.js, percebe-se uma diferença essencial: o Sequelize é um ORM completo, enquanto o Knex é um construtor de queries. Com Sequelize, você define models e relações; com Knex, você escreve as consultas com JavaScript, mas sem abstração orientada a objetos. Isso dá mais controle ao desenvolvedor, mas exige mais responsabilidade. A escolha entre um e outro depende do tipo e do tamanho do projeto, da equipe e das necessidades específicas de cada caso.

Referências

- Wikipedia - Mapeamento objeto-relacional: https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional
- Medium - Sequelize ORM: https://medium.com/@ogustavorichard/sequelize-orm-ccc3a54a5f05
- Reddit - O que é Sequelize: https://www.reddit.com/r/brdev/comments/11dhypv/sequelize_%C3%A9_uma_ferramenta_amplamente_utilizada/