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

FELIPE FRASSETTO FERNANDES
GABRIEL DE SOUZA TRAJANO

Desvendando o Mundo dos ORMs com Sequelize

SOMBRIO
2025

1. Introdução

Com o avanço das tecnologias web e a crescente demanda por aplicações mais dinâmicas e escaláveis, ferramentas que facilitam a integração entre o código da aplicação e o banco de dados tornaram-se essenciais. Entre elas, os ORMs (Object-Relational Mappers) ganham destaque por simplificar esse processo. Este trabalho tem como objetivo apresentar os fundamentos dos ORMs, com ênfase no Sequelize, abordando seu funcionamento, recursos, benefícios e limitações.

2. Fundamentos dos ORMs

2.1. O que é um ORM (Mapeamento Objeto-Relacional)?

ORM é uma forma de conectar o código do sistema com o banco de dados. Com ele, dá pra usar objetos do código para acessar e salvar dados, sem precisar escrever comandos SQL direto. Isso deixa tudo mais fácil e organizado.

Os programas usam objetos e os bancos de dados usam tabelas, que funcionam de formas diferentes. Isso torna difícil fazer o código **se comunicar** com o banco.

O Paradigma da Impedância Objeto-Relacional é o problema central que os ORMs (Object-Relational Mappers) se propõem a resolver. Ele surge da dificuldade fundamental em conciliar a forma como os dados são estruturados e manipulados em linguagens de programação orientadas a objetos e em bancos de dados relacionais.

2.2. Como um ORM funciona em alto nível?

O ORM funciona em alto nível mapeando classes do código para tabelas do banco de dados. Quando o desenvolvedor realiza operações como criar, buscar ou atualizar objetos, o ORM traduz essas ações em comandos SQL equivalentes, envia ao banco de dados e retorna os resultados como objetos manipuláveis no código. Dessa forma, ele automatiza e simplifica a comunicação entre a aplicação e o banco de dados.

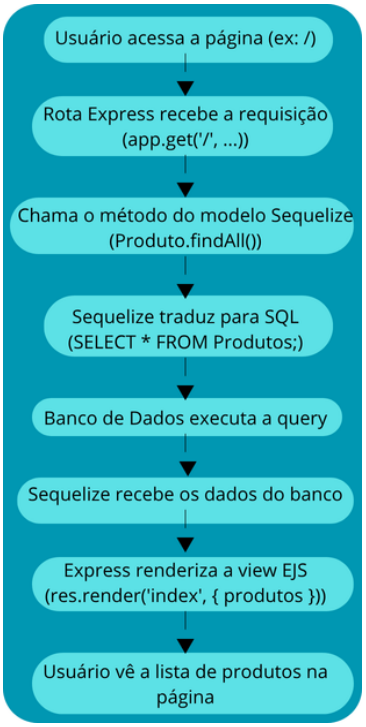


Figura 1. Fluxograma que ilustra o caminho de uma requisição[Autores 2025].

2.3 Explique o papel da camada de abstração que o ORM cria.

A camada de abstração criada por um ORM (Object-Relational Mapping) funciona como uma ponte entre a aplicação e o banco de dados. Ela permite que os desenvolvedores manipulem os dados como objetos na linguagem de programação, sem a necessidade de escrever comandos SQL diretamente.

3. Sequelize em detalhes

3.1 O que é o Sequelize?

O Sequelize é um ORM (Mapeador Objeto-Relacional) moderno desenvolvido com TypeScript e Node.js. Ele facilita a interação entre aplicações e bancos de dados relacionais, permitindo que você trabalhe com dados usando objetos em vez de escrever SQL puro. Sua popularidade no ecossistema Node.js vem justamente dessa praticidade, além de contar com recursos avançados como suporte a transações, relacionamentos entre tabelas, carregamento rápido ou sob demanda, replicação de leitura, entre outros.

Os principais dialetos suportados são: PostgreSQL, MySQL, MariaDB, SQLite e MSSQL.

3.2 Models e Associações: O Coração do Sequelize

No Sequelize, um Model é a representação de uma tabela do banco de dados no código. Ele define os campos, seus tipos, regras como chaves primárias e validações, e também os relacionamentos com outros Models. É a base para que o Sequelize traduza comandos em JavaScript para operações no banco de dados.

models > JS Produto.js > ...

```
1   // models/Produto.js
2
3   const { DataTypes } = require('sequelize');
4   const sequelize = require('../config/database'); // Importa a instância do Sequelize que configuramos
5
6   const Produto = sequelize.define('Produto', {
7       // Define os atributos do modelo (colunas da tabela)
8       id: {
9           type: DataTypes.INTEGER,
10          autoIncrement: true,
11          primaryKey: true,
12          allowNull: false
13      },
14      nome: {
15          type: DataTypes.STRING,
16          allowNull: false // O nome não pode ser nulo
17      },
18      preco: {
19          type: DataTypes.DECIMAL(10, 2), // DECIMAL(10, 2) para números com 2 casas decimais (ex: 123.45)
20          allowNull: false // O preço não pode ser nulo
21      }
22  }, {
23      // Opções adicionais do modelo
24      tableName: 'produtos', // Nome da tabela no banco de dados (o Sequelize pluraliza por padrão, mas é bo
25      timestamps: true // Adiciona automaticamente as colunas createdAt e updatedAt
26  });
27
28  module.exports = Produto; // Exporta o modelo Produto
```

Figura 2. Exemplo de código de um model 'Produto'[Autores 2025].

3.3 Associações

- hasOne(B)associação significa que existe um relacionamento Um-Para-Um entre Ae B, com a chave estrangeira sendo definida no modelo de destino (B).

-belongsTo(B)associação significa que existe um relacionamento Um-Para-Um entre Ae B, com a chave estrangeira sendo definida no modelo de origem (A).

-hasMany(B)associação significa que existe um relacionamento Um-Para-Muitos entre Ae B, com a chave estrangeira sendo definida no modelo de destino (B).

-belongsToMany(B, { through: 'C' })associação significa que existe um relacionamento Muitos-Para-Muitos entre Ae B, usando a tabela Como tabela de junção , que terá as chaves estrangeiras (aIde bId, por exemplo). O Sequelize criará automaticamente este modelo C(a menos que ele já exista) e definirá as chaves estrangeiras apropriadas nele.

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const Usuario = sequelize.define('Usuario', {
  nome: DataTypes.STRING,
});

const Perfil = sequelize.define('Perfil', {
  bio: DataTypes.TEXT,
});

const Produto = sequelize.define('Produto', {
  nome: DataTypes.STRING,
});

const Aluno = sequelize.define('Aluno', { nome: DataTypes.STRING });
const Curso = sequelize.define('Curso', { titulo: DataTypes.STRING });

Usuario.hasOne(Perfil);
Perfil.belongsTo(Usuario);

Usuario.hasMany(Produto);
Produto.belongsTo(Usuario);

Aluno.belongsToMany(Curso, { through: 'AlunoCurso' });
Curso.belongsToMany(Aluno, { through: 'AlunoCurso' });

(async () => {
  await sequelize.sync({ force: true });
  console.log("Banco sincronizado com sucesso!");
})();

```

Figura 3. Exemplo de código para cada relação [Autores 2025].

3.4 Consultas (Queries) e a Abstração do SQL

A tabela 1 apresenta um comparativo entre comandos SQL tradicionais e suas equivalentes implementações usando o Sequelize em JavaScript. O objetivo é demonstrar como operações comuns no banco de dados, como buscar todos os registros, buscar por ID, aplicar condições com WHERE e realizar JOINs que podem ser abstraídas por métodos do Sequelize, facilitando o desenvolvimento. Enquanto o SQL puro exige a escrita direta da consulta, o Sequelize oferece uma abordagem orientada a objetos, mais integrada ao código da aplicação, mantendo a expressividade e legibilidade.

Tabela 1. Comparativo de SQL Puro com Sequelize [Autores 2025].

Operação	SQL Puro	Sequelize (JavaScript)
Buscar todos os registros	SELECT * FROM usuarios;	Usuario.findAll();
Buscar um registro por ID	SELECT * FROM usuarios WHERE id = 1;	Usuario.findByPk(1);
Buscar registros com condição (WHERE)	SELECT * FROM usuarios WHERE nome = 'Felipe';	Usuario.findAll({ where: { nome: 'Felipe' } });
Buscar com JOIN (INNER JOIN)	SELECT * FROM usuarios INNER JOIN perfis ON perfis.UsuarioId = usuarios.id;	Usuario.findAll({ include: Perfil });

4. Tópicos Avançados e Boas Práticas

4.1. Migrations: Gerenciamento da Evolução do Banco de Dados

O comando `sequelize.sync({ force: true })` é perigoso em produção porque ele apaga todas as tabelas do banco de dados e as recria do zero, resultando na perda total de dados. Esse comando deve ser usado apenas em ambientes de teste ou desenvolvimento.

Migrations são arquivos usados para registrar e aplicar mudanças no banco de dados de forma organizada. Elas ajudam a manter um histórico das alterações feitas na estrutura do banco, como criação ou modificação de tabelas e colunas.

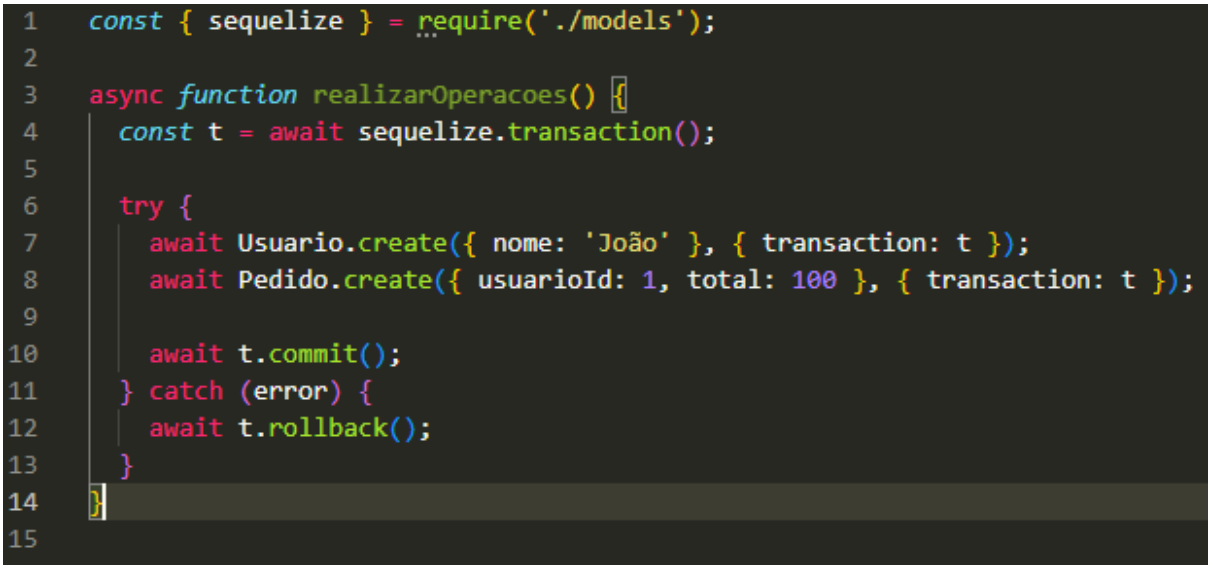
O uso de migrations resolve o problema de controlar a evolução do banco de dados, permitindo que a estrutura seja replicada de forma segura em diferentes ambientes (como desenvolvimento, testes e produção), além de possibilitar a reversão de alterações em caso de erros.

O fluxo de uso básico é: criar uma migration, aplicar as mudanças com o comando de migração (`migrate`) e, se necessário, reverter com o comando de `undo`.

4.2. Transações (Transactions)

Uma transação em um banco de dados é um conjunto de operações que são executadas como uma unidade única. Isso significa que ou todas as operações são concluídas com sucesso, ou nenhuma delas é aplicada. Isso garante a integridade dos dados e está relacionado ao conceito de atomicidade que assegura que uma operação composta seja "tudo ou nada".

Transações são importantes, por exemplo, quando se realiza um débito e um crédito em contas bancárias: não faz sentido debitar um valor sem creditar o outro lado. Se algo falhar no meio do processo, os dados devem voltar ao estado anterior.



```
const { sequelize } = require('./models');

async function realizarOperacoes() {
  const t = await sequelize.transaction();

  try {
    await Usuario.create({ nome: 'João' }, { transaction: t });
    await Pedido.create({ usuarioId: 1, total: 100 }, { transaction: t });

    await t.commit();
  } catch (error) {
    await t.rollback();
  }
}
```

Figura 4. Exemplo de código de Como o Sequelize permite que você execute múltiplas operações dentro de uma única transação [Autores 2025].

5. Análise Crítica e Comparativa

5.1 Vantagens e Desvantagens de Usar um ORM

Usar um ORM como o Sequelize traz diversas vantagens. A primeira é o ganho de produtividade, pois ele permite escrever consultas ao banco de dados usando JavaScript, o que simplifica o desenvolvimento. A segunda vantagem é a portabilidade, já que facilita a troca de banco de dados sem grandes alterações no código. Por fim, há o reforço na segurança, pois o Sequelize ajuda a prevenir ataques como SQL Injection de forma automática.

Por outro lado, também existem desvantagens. A primeira é a curva de aprendizado, especialmente para quem está começando e precisa entender conceitos como models, migrations e associações. A segunda é a possível perda de performance em consultas complexas, já que o ORM pode gerar instruções SQL menos otimizadas. E, por fim, em alguns cenários, a abstração do ORM pode ser limitada, exigindo o uso direto de SQL, o que pode complicar o desenvolvimento.

5.2 Quando NÃO usar um ORM?

Um ORM pode não ser a melhor opção em situações onde as consultas ao banco de dados são muito complexas e demandam um controle preciso sobre a execução do SQL, para garantir desempenho otimizado. Também pode ser inadequado para sistemas que lidam com grandes volumes de dados e operações de alta escala, pois a camada adicional do ORM pode introduzir lentidão. Além disso, em projetos mais simples ou em scripts pontuais, utilizar SQL puro ou um Query Builder pode ser mais eficiente, já que essas abordagens oferecem maior flexibilidade e menos complexidade do que o uso completo de um ORM.

5.3 Comparativo: Sequelize vs. Outras Ferramentas

A Tabela 2 compara o Sequelize com o Prisma, duas ferramentas amplamente utilizadas para o gerenciamento de bancos de dados em aplicações JavaScript/TypeScript. O comparativo abrange critérios como o tipo de ferramenta, linguagem principal, forma de definição de schema e facilidade de uso. Enquanto o Sequelize é um ORM tradicional e mais flexível, com maior suporte à personalização, o Prisma se destaca como uma solução moderna, voltada ao TypeScript e com foco em simplicidade e segurança de tipos. A tabela evidencia como cada ferramenta atende a diferentes perfis de projeto e preferências de desenvolvimento.

Tabela 2. Comparativo de Sequelize com Prisma [Autores 2025].

Critério	Sequelize	Prisma
Tipo de ferramenta	ORM tradicional	ORM moderno + Query Builder
Linguagem principal	JavaScript com suporte opcional a TypeScript	Projetado para TypeScript
Forma de definir schema	Definição via código JS/TS	Arquivo declarativo separado
Facilidade de uso	Flexível, mas verboso; curva de aprendizado maior, mais configuração manual	API mais intuitiva, menos boilerplate, melhor suporte a Type Safety

6. Conclusão

O Sequelize representa uma solução prática e robusta para o mapeamento objeto-relacional em aplicações Node.js, ao proporcionar uma camada de abstração eficiente e segura. Ainda que ofereça muitas facilidades, é necessário compreender bem suas características e limitações para utilizá-lo de forma adequada, respeitando as particularidades de cada projeto e garantindo a qualidade do desenvolvimento.

7. Referências

ALURA. *ORM*. Recuperado de: https://www.alura.com.br/artigos/orm. Acessado em: 16 jul. 2025.

ALURA. *ORM?*. Recuperado de: https://www.alura.com.br/artigos/orm?srsltid=AfmBOoqqwuObvVxb6MnMVBT1cOtRkyaIhzRvnehspmrRVvPSWL4IpXgQ. Acessado em: 16 jul. 2025.

DEVIMEDIA. *ORM - Object Relational Mapper*. Recuperado de: https://www.devmedia.com.br/orm-object-relational-mapper/19056. Acessado em: 16 jul. 2025.

FREECODECAMP. *O que é um ORM? O significado das ferramentas de mapeamento relacional de objetos de banco de dados*. Recuperado de: https://www.freecodecamp.org/portuguese/news/o-que-e-um-orm-o-significado-das-ferramentas-de-mapeamento-relacional-de-objetos-de-banco-de-dados/. Acessado em: 16 jul. 2025.

MERCADO ONLINE DIGITAL. *ORM*. Recuperado de: https://mercadoonlinedigital.com/blog/orm/#:~:text=Camada%20de%20Abstra%C3%A7%C3%A3o%20do%20Banco,continuaremos%20nos%20t%C3%B3picos%20a%20seguir. Acessado em: 16 jul. 2025.

PRISMA. *Prisma and Sequelize*. Recuperado de: https://www.prisma.io/docs/orm/more/comparisons/prisma-and-sequelize. Acessado em: 16 jul. 2025.

PROGRAMAE. *O que é ORM (Object-Relational Mapping) no Desenvolvimento Web?*. Recuperado de: https://programae.org.br/web/glossario/o-que-e-orm-object-relational-mapping-no-desenvolvimento-web/. Acessado em: 16 jul. 2025.

SEQUELIZE. *Sequelize*. Recuperado de: https://sequelize.org/. Acessado em: 16 jul. 2025.

SEQUELIZE. *Associações*. Recuperado de: https://sequelize-org.translate.goog/docs/v6/core-concepts/assocs/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc. Acessado em: 16 jul. 2025.

TREINAWEB. *O que é ORM*. Recuperado de: https://www.treinaweb.com.br/blog/o-que-e-orm#google_vignette. Acessado em: 16 jul. 2025.