

Trabalho sobre o Sequelize

Pedro Henrique Colares Ramos da Rosa

Vitor Cordioli Sandrini

Joédio Borges Júnior

Sombrio

2025

## Sumário

1 Introdução ............................................................................................. 2  
 2 Fundamentos dos ORMs ........................................................................ 2  
  2.1 O que é um ORM ............................................................................. 2  
  2.2 Paradigma da Impedância Objeto‑Relacional ................................. 2  
  2.3 Funcionamento de um ORM em Alto Nível .................................. 2  
  2.4 Papel da Camada de Abstração .................................................... 3  
 3 Sequelize em Detalhes .......................................................................... 3  
  3.1 O que é o Sequelize ........................................................................ 3  
  3.2 Models e Associações ................................................................. 4  
  3.3 Consultas e Abstração do SQL ..................................................... 5  
 4 Tópicos Avançados e Boas Práticas ....................................................... 5  
  4.1 Migrations ...................................................................................... 5  
  4.2 Transações ...................................................................................6  
 5 Análise Crítica e Comparativa ...........................................................6  
  5.1 Vantagens e Desvantagens de ORMs ............................................6  
  5.2 Quando NÃO usar um ORM ..........................................................7  
  5.3 Comparativo: Sequelize vs. Knex.js ..............................................7  
 6 Conclusão ...........................................................................................7  
 7 Referências Bibliográficas .................................................................8

## 1. Introdução

O desenvolvimento de aplicações modernas exige uma comunicação eficiente entre código orientado a objetos e bancos relacionais. O Mapeamento Objeto-Relacional (ORM) surge para facilitar essa integração, reduzindo a complexidade do acesso a dados e aumentando a produtividade dos desenvolvedores. Este trabalho explora os fundamentos dos ORMs, com foco no Sequelize, um ORM popular para Node.js, além de analisar boas práticas, tópicos avançados e uma comparação crítica com outras ferramentas.

## 2. Fundamentos dos ORMs

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

Um ORM (Object-Relational Mapping) é uma técnica que permite mapear objetos de uma linguagem de programação orientada a objetos (como JavaScript, Java, Python) para tabelas de um banco de dados relacional (como MySQL, PostgreSQL, SQLite). Isso significa que você pode interagir com o banco de dados usando código como se estivesse manipulando objetos, e não SQL diretamente.

### 2.2 Paradigma da Impedância Objeto-Relacional

Esse paradigma representa o conflito conceitual entre o modelo orientado a objetos (com herança, encapsulamento, objetos complexos) e o modelo relacional (baseado em tabelas com colunas e linhas). ORMs tentam reduzir esse atrito, traduzindo automaticamente as operações entre os dois mundos.

### 2.3 Funcionamento de um ORM em Alto Nível

Quando você executa um comando como Usuario.findAll():

1. O ORM converte essa instrução em uma query SQL (SELECT \* FROM Usuarios;).
2. Essa query é enviada para o banco de dados.
3. O banco retorna os resultados em formato tabular.
4. O ORM transforma esses dados em objetos JavaScript, como se fossem instâncias da classe Usuario.

### 2.4 Papel da Camada de Abstração

Essa camada oculta os detalhes do SQL, permitindo que o desenvolvedor se concentre na lógica de negócio. Ela também facilita a portabilidade entre bancos de dados diferentes, melhora a produtividade e reduz erros comuns de SQL.

## 3. Sequelize em Detalhes

### 3.1 O que é o Sequelize?

O Sequelize é um ORM baseado em JavaScript que funciona com o Node.js, permitindo o uso de modelos JS para manipular bancos relacionais. Ele é muito usado por sua simplicidade, integração com Promises/async-await e ampla documentação.

**Dialetos Suportados:**

* PostgreSQL
* MySQL
* MariaDB
* SQLite
* Microsoft SQL Server

### 3.2 Models e Associações: O Coração do Sequelize

**Exemplo de Model Produto:**

**Associações principais:**

**hasOne (Um-para-Um)**  
****

**hasMany (Um-para-Muitos)  
**

**belongsTo (Pertence a)  
**

**belongsToMany (Muitos-para-Muitos)  
**

**3.3 Consultas (Queries) e a Abstração do SQL**

Na seção de consultas e abstração do SQL, o Sequelize traduz operações comuns em SQL para métodos JavaScript equivalentes. Por exemplo, para buscar todos os registros, o SQL **SELECT \* FROM Produtos;** corresponde ao método **Produto.findAll();**. Para buscar um registro específico por ID, o SQL **SELECT \* FROM Produtos WHERE id = 1;** equivale a **Produto.findByPk(1);**. Para buscar registros com condição, como **SELECT \* FROM Produtos WHERE preco > 50;**, usa-se **Produto.findAll({ where: { preco: { [Op.gt]: 50 } } });** no Sequelize. Já para realizar um JOIN e buscar dados relacionados entre tabelas, como **SELECT \* FROM Produtos JOIN Categorias ON Produtos.categoria\_id = Categorias.id;**, a função Sequelize é **Produto.findAll({ include: Categoria });.**

## 4. Tópicos Avançados e Boas Práticas

### 4.1 Migrations: Evolução do Banco de Dados

**Por que evitar sync({ force: true }) em produção?** Esse comando apaga e recria todas as tabelas, o que pode causar perda total de dados em ambientes reais. Deve ser usado apenas em testes.

**O que são Migrations?**

Migrations são scripts versionados que permitem modificar o banco de dados de forma controlada (adicionar tabelas, campos, índices, etc.), mantendo o histórico das mudanças.

**Fluxo de migration:**

1. Criar:  
   npx sequelize-cli migration:generate --name create-produto
2. Aplicar(up):

npx sequelize-cli db:migrate

1. Reverter (down):

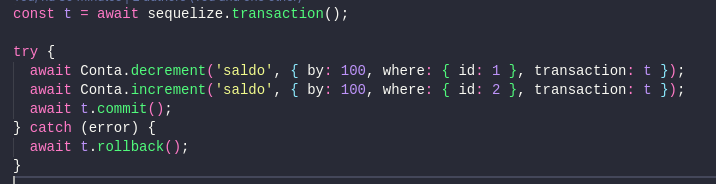
npx sequelize-cli db:migrate:undo

### 

### 

### 4.2 Transações (Transactions)

**O que é uma transação?** É um conjunto de operações que deve ser executado completamente ou não executado, garantindo a atomicidade. Exemplo clássico: transferir dinheiro entre contas.

**Exemplo com Sequelize:**

## 5. Análise Crítica e Comparativa

### 5.1 Vantagens e Desvantagens de ORMs

**Vantagens:**

1. **Produtividade:** Código mais simples, menos SQL manual.
2. **Abstração:** Facilita o uso de diferentes bancos sem grandes mudanças.
3. **Organização:** Facilita testes, modularização e manutenção.

**Desvantagens:**

1. **Performance:** Pode gerar SQL ineficiente.
2. **Curva de aprendizado:** Exige entender o funcionamento interno.
3. **Abstração pode vazar:** Algumas operações específicas exigem SQL puro.

### 5.2 Quando NÃO usar um ORM?

* Sistemas com grande volume de dados e performance crítica.
* Quando há uso intenso de stored procedures, triggers ou funcionalidades específicas do banco.
* Projetos onde se quer controle absoluto sobre as queries SQL.

### 5.3 Comparativo: Sequelize vs. [Knex.js](http://knex.js)

| **Critério** | **Sequelize** | **Knex.js** |
| --- | --- | --- |
| Tipo | ORM | Query Builder |
| Linguagem principal | JavaScript | JavaScript |
| Definição de schema | Models (classes) | Código JS estruturado |
| Facilidade de uso | Alta para CRUD simples | Mais flexível, mas mais código |
| Aprendizado | Mais fácil para iniciantes | Mais indicado para quem já entende SQL |

## 6. Conclusão

Este trabalho apresentou o conceito de ORM, destacando seu papel fundamental para facilitar a integração entre código orientado a objetos e bancos relacionais. O Sequelize, por sua simplicidade e popularidade no ecossistema Node.js, foi analisado em detalhes, incluindo suas funcionalidades, boas práticas e limitações. Também foram discutidas vantagens e desvantagens dos ORMs, além de situações onde seu uso pode não ser adequado. Por fim, um comparativo com o Knex.js ajudou a evidenciar as diferenças entre ORMs e query builders, auxiliando na escolha da ferramenta ideal conforme a necessidade do projeto.

**7. Referências Bibliográficas**

AMBELER, Scott W. The Object-Relational Impedance Mismatch. Agile Data, 2007. Disponível em: https://www.agiledata.org/essays/impedanceMismatch.html. Acesso em: 09 jul. 2025.

FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003.

SEQUELIZE. Sequelize Documentation. Disponível em: https://sequelize.org/master/. Acesso em: 09 jul. 2025.