

INSTITUTO FEDERAL DE MATO GROSSO DO SUL
CAMPUS NOVA ANDRADINA

NOTAS DE AULA

PROGRAMAÇÃO E TECNOLOGIAS
PARA APLICAÇÕES SERVIDOR 2

Prof. Me. Luiz F. Picolo

NOVA ANDRADINA - MS

Atualizado em 11 de abril de 2022

1 Introdução

*Persistência é a irmã gêmea da excelência.
Uma é a mãe da qualidade, a outra é a mãe do tempo.*

Marabel Morgan

O armazenamento de dados é uma importante parte de qualquer aplicativo de computador. Os dados coletados precisam ser armazenados para análise posterior, ou precisam ser recuperados para gerar as saídas desejadas. Existem muitas maneiras de armazenar dados, sendo que, o armazenamento de arquivos (incluindo opções de texto e/ou binárias), bancos de dados relacionais, sistemas NoSQL, entre outros.

Contudo, da mesma havendo diversas forma para o armazenamento de dados, o objetivo aqui é demonstrar o uso de um banco de dados relacional junto ao NodeJS. Para isso, será usada a linguagem padrão para consulta, a SQL (Structured Query Language).

A linguagem de consulta SQL é um padrão compartilhado em muitos sistemas de gerenciamento de banco de dados relacional (também conhecido como SGDBs). O SQL implementado às vezes varia do padrão oficial. Essas variações não importam muito se você permanecer na mesma plataforma de banco de dados. No entanto, no momento em que você começa a alternar entre diferentes fornecedores de banco de dados ou até mesmo versões diferentes no mesmo fornecedor, as diferenças podem se tornar importantes. Alternar entre sistemas relacionais e não relacionais traz ainda mais trabalho. Assim, é importante analisar o projeto antes de decidir qual banco de dados será utilizado.

Já o banco escolhido, inicialmente, será o Postgresql. Esse SGDB, segundo seus desenvolvedores, é poderoso sistema de banco de dados objeto-relacional de código aberto com mais de 30 anos de desenvolvimento ativo que lhe rendeu uma forte reputação de confiabilidade, robustez de recursos e desempenho¹.

As notas de aula, não se preocupam com detalhes de instalação e/ou configuração, pois se acredita que, nesse momento do curso, o estudante já está capacitado para tal tarefa. Contudo, caso o leitor queira montar seu ambiente de trabalho, segue as tecnologias base que serão utilizadas nesse primeiro capítulo.

- NodeJS - versão 16+
- Express - versão 4+
- Postgres - 14+ (será usado a versão online disponível em: <<https://www.elephantsql.com>>)

¹ Para mais detalhes, acesse: <<https://www.postgresql.org>>

1.1 Conceitos básicos sobre Banco de dados Relacionais

Segundo Elmasri et al. 2005, um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito. Por exemplo, considere nomes, números telefônicos e endereços de pessoas que você conhece. Esses dados podem ter sido escritos em uma agenda de telefones ou armazenados em um computador. Essas informações são uma coleção de dados com um significado implícito, consequentemente, um banco de dados.

Ainda, segundo os autores:

Um banco de dados pode ser gerado e mantido manualmente ou pode ser automatizado (computadorizado). Por exemplo, um catálogo de cartões bibliotecários é um banco de dados que oferece a possibilidade de ser criado e mantido manualmente. Um banco de dados computadorizado pode ser criado e mantido tanto por um grupo de aplicativos escritos especialmente para essa tarefa como por um sistema gerenciador de banco de dados [Elmasri et al. 2005, p. 04].

Já, um Sistema Gerenciador de Banco de dados, (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações [Elmasri et al. 2005].

Assim, a base de dados e o software de gerenciamento da base de dados compõem o chamado Sistema de Base de Dados. A Figura 1 apresenta um esquema genérico de um Sistema de Banco de Dados em sua interação com seus usuários [Takai, Italiano e Ferreira 2005].

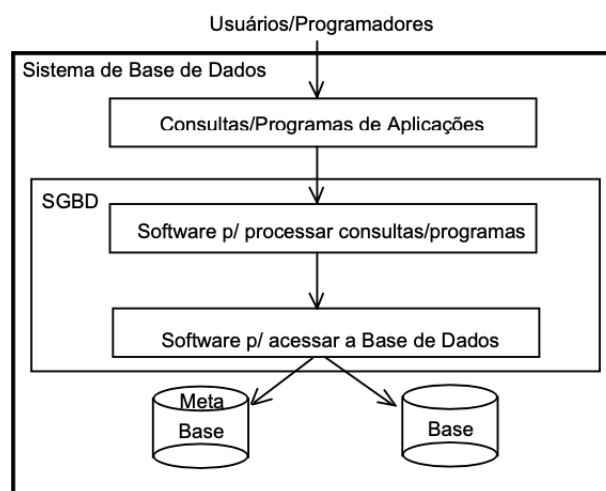


Figura 1 – Sistema de Banco de Dados Fonte: [Takai, Italiano e Ferreira 2005]

1.1.1 Modelo Entidade-Relacionamento: Entidades e atributos

O objeto básico que o Modelo Entidade-Relacionamento (MER) representa é a **entidade**. Uma entidade é algo do mundo real que possui uma existência independente. Uma entidade pode ser um objeto com uma existência física - uma pessoa, carro ou empregado - ou pode ser um objeto com existência conceitual - uma companhia, um trabalho ou um curso universitário. Já, cada entidade tem propriedades particulares, chamadas atributos, que a descrevem [Takai, Italiano e Ferreira 2005].

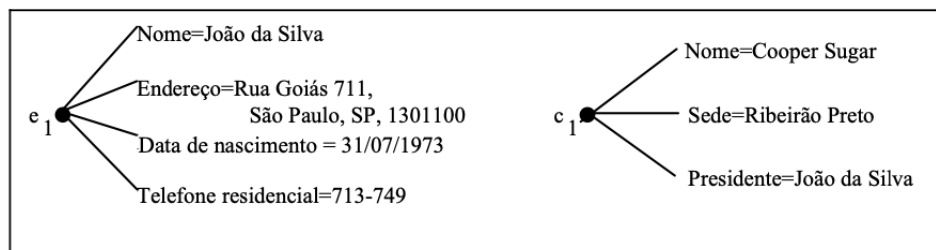


Figura 2 – Exemplo de entidades e seus respectivos atributos Fonte: [Takai, Italiano e Ferreira 2005]

Alguns atributos podem ser divididos em subpartes com significados independentes. Por exemplo, **Endereço**, como visto na Figura 3, pode ser dividido em **Endereço da Rua**, **Cidade**, **Estado** e **CEP**. Um atributo que é composto de outros atributos mais básicos é chamado **composto**. Já, atributos que não são divisíveis são chamados **simples** ou **atômicos**.

Tipos de atributos:

- **Compostos**: Podem ser divididos em partes menores;
- **Simples**: Eles não são divisíveis;
- **Monovalorados**: Possuem apenas um valor;
- **Multivalorado**: Um ou mais valores para o mesmo;
- **Armazenado**: Em geral todos os atributos são armazenados;
- **Derivado**: Alguns atributos podem ter uma relação entre si;
- **Nulo**: Quando valores podem não ser aplicados.

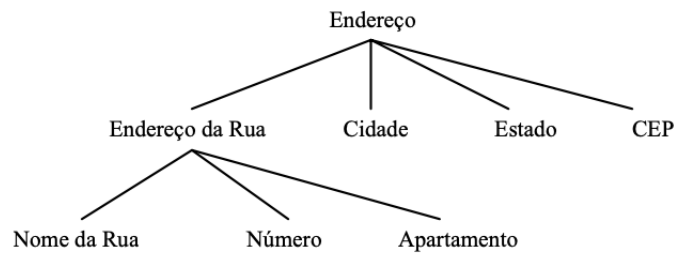


Figura 3 – Exemplo de atributos compostos Fonte: [Takai, Italiano e Ferreira 2005]

Um banco de dados inclui uma coleção de conjuntos de entidades, cada qual contendo um número de entidades do mesmo tipo. A figura abaixo mostra parte de um banco de dados que consiste em dois conjuntos de entidades: clientes e contas [Sanches 2005]

cliente				conta	
Oliver	645-32-1098	Main	Austin	259	1000
Harris	890-12-3456	North	Georgetown	630	2000
Marsh	456-78-9012	Main	Austin	401	1500
Pepper	369-12-1518	North	Georgetown	700	1500
Ratliff	246-80-1214	Park	Round Rock	199	500
Brill	121-21-2121	Putnam	San Marcos	467	900
Evers	135-79-1357	Nassau	Austin	115	1200
				183	1300
				118	2000
				225	2500
				210	2200

Figura 4 – Conjuntos de entidades cliente e conta. Fonte: [Sanches 2005]

1.1.2 Relacionamentos e conjunto de relacionamentos

Um relacionamento, segundo Sanches 2005, é uma associação entre diversas entidades. Por exemplo, podemos definir um relacionamento que associa o **cliente Harris** à **conta 401**. Isto especifica que Harris é um cliente com conta bancária número 401. Assim, o relacionamento ContaCliente é um exemplo de um conjunto de relacionamentos **binários** - isto é, ele envolve dois conjuntos de entidades.

cliente			
Oliver	645-32-1098	Main	Austin
Harris	890-12-3456	North	Georgetown
Marsh	456-78-9012	Main	Austin
Pepper	369-12-1518	North	Georgetown
Ratliff	246-80-1214	Park	Round Rock
Brill	121-21-2121	Putnam	San Marcos
Evers	135-79-1357	Nassau	Austin

conta	
259	1000
630	2000
401	1500
700	1500
199	500
467	900
115	1200
183	1300
118	2000
225	2500
210	2200

Figura 5 – Relacionamento envolvendo entidades: **Cliente** e **Conta**. Fonte: [Sanches 2005]

Os relacionamentos podem ser:

- **Um-para-um**: uma entidade A está associada no máximo a uma entidade B e uma entidade B está associada no máximo a entidade de A;
- **Um-para-muitos**: uma entidade A está associada a qualquer número de entidades de B. Uma entidade de B, entretanto, pode estar associada no máximo a uma entidade de A;
- **Muitos-para-um**: uma entidade A está associada no máximo a uma entidades de B. Uma entidade de B, entretanto, pode estar associada a qualquer número de entidades de A;
- **Muitos-para-muitos**: uma entidade A está associada a qualquer número de entidades de B e uma entidade de B está associada a qualquer número de entidades de A.

Abaixo, na Figura 6, pode-se visualizar as formas de relacionamento, como as entidades podem se relacionar para mais dar sentido a informação desejada.

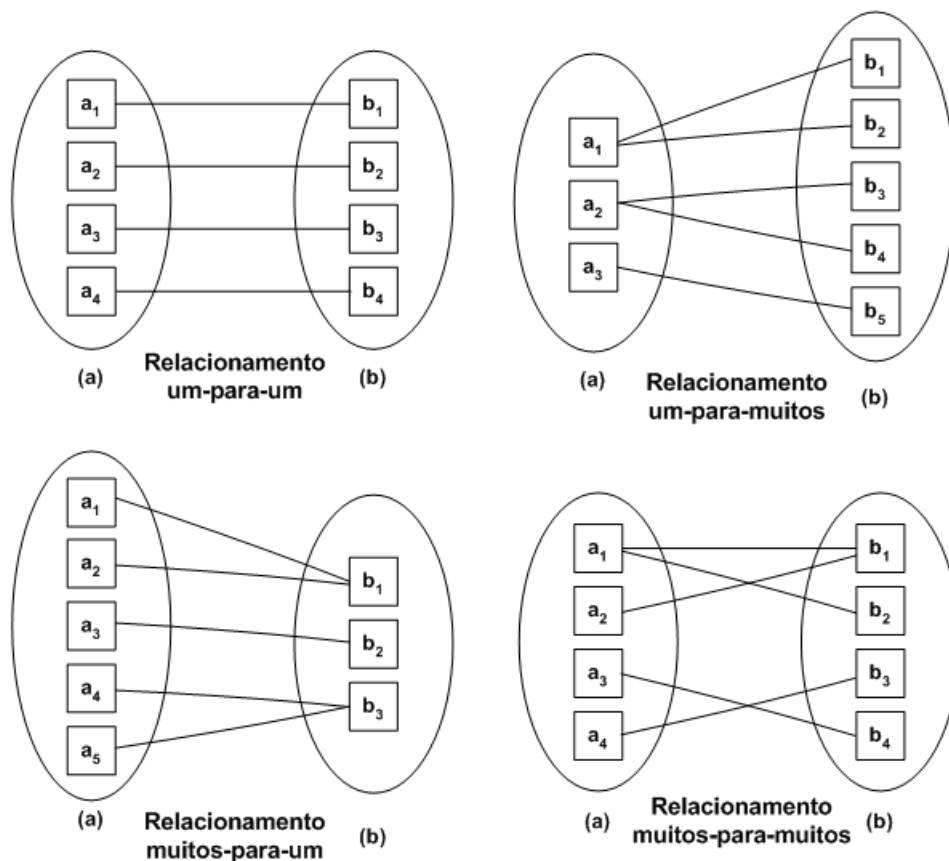


Figura 6 – Tipos de relacionamento. Fonte: [Sanches 2005]

1.1.3 Linguagens de SGBD's relacionais

A linguagem de SGBD's é essencialmente SQL, que no que lhe concerne é subdividido em grupos de comandos, conforme a função de cada comando.

e, resumindo,

- **DDL** = Data Definition Language - Linguagem de definição de dados, possibilita a definição ou descrição dos objetos do BD.
- **DML** = Data Manipulation Language - Linguagem de manipulação de dados, que suporta manipulação (acesso e alteração).
- **DCL** = Data Control Language - Linguagem de controle de dados, possibilita a determinação das permissões que cada usuário terá sobre os objetos do BD (permissão de criação, consulta, alteração, etc.).

Para este estudo, consideraremos apenas as DML que serão utilizadas no próximo capítulo. A Linguagem de Manipulação de Dados (DML - **Data Manipulation Language**) é um conjunto de comandos que determinam quais valores estão presentes nas tabelas em qualquer momento e como serão manipulados. No SQL, por exemplo, são instruções DML:

- SELECT - consulta (seleção)
- UPDATE - atualização
- DELETE - exclusão
- INSERT - inclusão

De forma simples, para ser realizada uma seleção de alguns dados no banco, pode-se fazer da seguinte forma:

```
1  select * from ENTIDADE
2  select * from ENTIDADE where ATRIBUTO = VALOR DESEJADO
3  select NOME_DA_COLUNA FROM ENTIDADE_1 inner join ENTIDADE_2
4  ON ENTIDADE_1.coluna_nome = ENTIDADE_2.coluna_nome;
```

Existem varias formas de se fazer uma seleção, atualização, entre outros. Contudo, em nossas práticas diárias, faremos algumas delas considerando o conhecimento adquirido pelo estudante no decorrer de várias disciplinas relacionadas a Banco de dados.

1.1.4 Exercícios de fixação

1. Defina os seguintes termos: dados, informação, banco de dados, SGBD e sistema de banco de dados.
2. Quais podem ser as diferentes categorias de usuários finais de banco de dados? Discuta as atividades principais de cada um.
3. Cite e explique algumas vantagens do uso de Banco de dados.
4. Quais são as categorias de linguagens de SGBD? Para que serve cada uma?
5. Uma escola possui várias turmas. Uma turma contém vários professores na qual um professor pode lecionar aulas em mais de uma turma. Uma turma possui sempre aulas na mesma sala, mas uma sala pode estar associada a várias turmas.
 - a) Liste as possíveis entidades que podem ser encontradas
 - b) Liste os possíveis atributos das entidades encontradas
 - c) Relacione, de forma simples, as entidades.

2 Criação e acesso a banco de dados relacional usando NodeJS

Hoje você acha cansativo, porém mais tarde receberá a recompensa por todo esse tempo que passou estudando.

Anônimo

Em nossa aula vamos aprender como podemos usar Node.js com PostgreSQL. Apesar de MongoDB e outras bases não relacionais serem uma das escolhas mais comuns com Node, muitos desenvolvedores, conhecem e usam PostgreSQL e não querer abrir mão de seu conhecimento a respeito. Também é importante deixar claro que partimos do pressuposto que você já saiba usar, minimamente, um banco de dados relacional como o Mysql por exemplo, e que também já saiba o básico de Node.js. Assim, vamos focar nossa aula em apenas ligar os pontos, ou seja, o Node.js com nosso banco dados PostgreSQL.

2.1 Usando o ElephantSQL

Para evitar o passo de instalação do banco em nossa maquina local, vamos usar um serviço online que poderá se comunicar com nossas aplicações desenvolvidas no Replit. Para tanto, vamos usar o plano *free* da <elephantsql.com> chamado de **Tiny Turtle**.

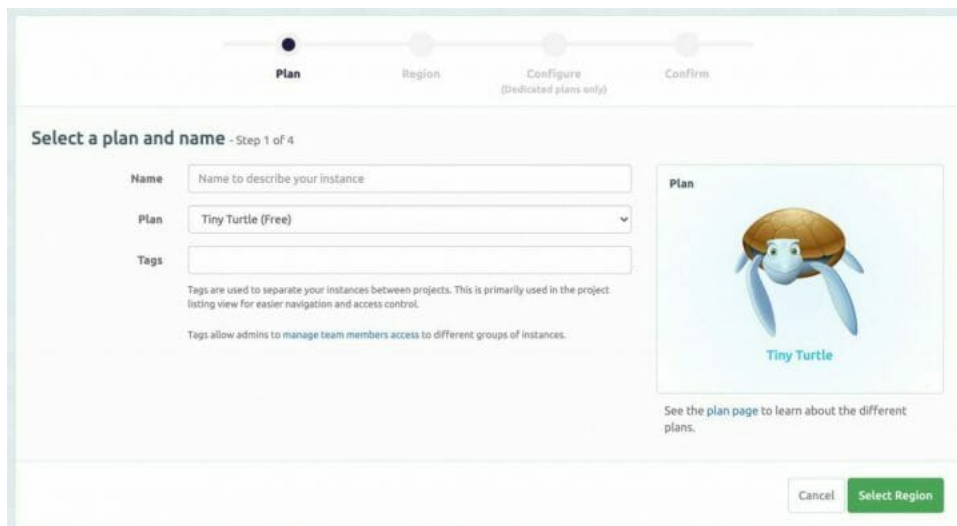
The image shows a web form for creating a new instance on ElephantSQL. At the top, there is a progress bar with four steps: 'Plan' (selected), 'Region', 'Configure (Dedicated plans only)', and 'Confirm'. The main heading is 'Select a plan and name - Step 1 of 4'. Below this, there are three input fields: 'Name' with a placeholder 'Name to describe your instance', 'Plan' with a dropdown menu showing 'Tiny Turtle (Free)', and 'Tags' with a placeholder. Below the 'Tags' field, there is a small text block explaining that tags are used for separating instances between projects and for easier navigation. To the right of the input fields, there is a box titled 'Plan' featuring a cartoon turtle and the text 'Tiny Turtle'. Below this box, there is a link to 'See the plan page to learn about the different plans.' At the bottom right of the form, there are two buttons: 'Cancel' and 'Select Region'.

Figura 7 – Criação da primeira instância

Após adicionar um nome, selecione a região (não é necessário alterar), revise os dados e clique em *Criar Instância*. Quando terminar, clique no nome da instância criada

você terá acesso às credenciais de acesso e todas as seções que utilizaremos para manipular nosso banco.

Para criar nossa primeira tabela, clique em *Browser* e será aberto o *SQL Browser*. Nele vamos digitar nossos comandos SQL.

2.1.1 Comando CREATE TABLE no PostgreSQL

O comando **SQL CREATE TABLE** é utilizado para definir uma nova tabela, inicialmente vazia (sem nenhum registro), no esquema de banco de dados atual. A tabela criada pertence ao usuário que executa o comando. Vejamos a sintaxe básica para a criação de uma tabela no postgres:

```
1 CREATE TABLE [IF NOT EXISTS] nome_tabela (  
2     nome_coluna tipo_dados [COLLATE colecao] constraint,  
3     nome_coluna tipo_dados constraint,  
4     nome_coluna tipo_dados constraint,  
5     ...,  
6     [FOREIGN KEY chave_estrangeira REFERENCES coluna]  
7     [ON DELETE acao ] [ ON UPDATE acao ]  
8 )
```

Vamos criar uma tabela **autores** que irá conter os campos **id nome**, **sobrenome** e **datanascimento**

```
1 CREATE TABLE autores (  
2     id SERIAL CONSTRAINT pk_id_autor PRIMARY KEY,  
3     nome varchar(30) NOT NULL,  
4     sobrenome varchar(40) NOT NULL,  
5     datanascimento date  
6 );
```

Agora, vamos criar a tabela de livros, incluindo os relacionamentos com as demais tabelas por meio do uso de chaves estrangeiras.

```
1 CREATE TABLE livros (  
2     id SERIAL CONSTRAINT pk_id_livro PRIMARY KEY,  
3     nome varchar(50) NOT NULL,
```

```
4      autor integer NOT NULL,  
5      editora integer NOT NULL,  
6      datapublicacao date,  
7      preco money,  
8      FOREIGN KEY (autor) REFERENCES autores (id) ON DELETE CASCADE  
9  );
```

Outra forma de estabelecer esse relacionamento é simplesmente indicar a referência de uma coluna em sua própria declaração, o que a torna uma chave primária. Neste exemplo poderíamos escrever simplesmente:

```
1  CREATE TABLE livros (  
2      id SERIAL CONSTRAINT pk_id_livro PRIMARY KEY,  
3      nome varchar(50) NOT NULL,  
4      autor integer REFERENCES autores(id) NOT NULL,  
5      editora integer NOT NULL,  
6      datapublicacao date,  
7      preco money  
8  );
```

2.1.2 Exercícios de fixação

Crie as seguintes tabelas com seus respectivos relacionamentos:

Empregado (id:Serial, matricula:integer, cpf:varchar, nome:varchar, endereço:varchar, cep:integer)

Projeto (id:Serial, nom:varchar, verba:money)

Alocação (id:Serial, projeto:foreingkey, empregado:foreingkey)

1. Escreva o código antes de adicionar ao banco
2. Valide o código com o professor
3. Adicione o código ao banco

2.2 Usando Node.js com o PostgreSQL

Para iniciarmos nossa jornada com Node.js e PostgreSQL, vamos criar nosso primeiro projeto usando o **express.js**. Para maiores detalhes, acesse o link para relem-

brar sobre o funcionamento do *Express Generator* <<https://expressjs.com/pt-br/starter/generator.html>>. Se, estiver usando o <<https://replit.com>> use o comando abaixo

```
1 npx express --view=ejs .
```

Contudo, para facilitar nosso trabalho, vamos realizar um *fork* do seguinte repositório <<https://github.com/luizpicolo/skeleton-nodejs-express-ejs.git>> e importá-lo no **replit**.

2.2.1 Conexão com o banco de dados

Assim, após todo o processo ter ocorrido sem erros, vamos configurar a conexão com o banco. Crie um arquivo **db.js** na raiz do seu repositório. Não podemos criando conexões infinitas no banco pois isso, além de ser lento, é inviável do ponto de vista de infraestrutura. Usaremos aqui um conceito chamado *connection pool*, onde um objeto irá gerenciar as nossas conexões, para abrir, fechar e reutilizar conforme possível. Vamos guardar este único pool em uma variável global, que testamos logo no início da execução para garantir que se já tivermos um *pool*, que vamos utilizar o mesmo.

```
1 let connect = function() {  
2   if (global.connection){  
3     return global.connection.connect();  
4   }  
5  
6   const { Pool } = require('pg');  
7   const pool = new Pool({  
8     connectionString: 'URL PARA O BANCO DE DADOS'  
9   });  
10  
11   global.connection = pool;  
12   return pool.connect();  
13 }  
14  
15 module.exports = { connect };
```

Para que o código acima funcione corretamente, devemos instalar a dependência **pg**, executando o comando abaixo.

```
1 npm i pg --save
```

2.2.2 Criando nosso primeiro modelo para acesso ao dados

Para que possamos testar a conexão com o banco em nossos modelos, vamos criar um modelo de exemplo chamado **Autor** e invocar o código de conexão da seguinte forma.

```
1 const db = require("../db");
2
3 class Autor {
4
5 }
6
7 module.exports = Autor;
```

Usando apenas a chamada acima, já possuímos um modelo que pode obter os dados necessários por meio de uma conexão com o banco de dados.

2.2.3 As quatro operações básicas em um banco de dados

Nas manipulações de registros realizadas diretamente em banco de dados ou em plataformas desenvolvidas no padrão *RESTful*, o conceito **CRUD** estabelece o modelo correto no manuseio desses dados.

CRUD representa as quatro principais operações realizadas em banco de dados, seja no modelo relacional (SQL) ou não-relacional (NoSQL), facilitando no processamento dos dados e na consistência e integridade das informações.

A sigla CRUD significa as iniciais das operações create (criação), read (leitura), update (atualização) e delete (exclusão). Essas quatro siglas tratam a respeito das operações executadas em bancos de dados relacional (SQL) e não-relacional (NoSQL). Essas operações pertencem ao agrupamento chamado de *Data Manipulation Language (DML)*, utilizado na linguagem Structured Query Language (SQL)¹.

2.2.3.1 Create

A operação de criação de um registro em uma tabela é realizada pelo comando INSERT. Exemplo:

¹ Para mais detalhes acesse: <<https://blog.betrybe.com/tecnologia/crud-operacoes-basicas/>>

```
1 class Autor {
2   static async insert(data){
3     const connect = await db.connect();
4     const sql = 'insert into autores(nome, sobrenome, datanascimento)
5       ↪ values ($1, $2, $3)';
6     const values = [data.nome, data.sobrenome, data.datanascimento];
7     return await connect.query(sql, values);
8   }
}
```

2.2.3.2 Read

A operação de consulta de um ou mais registros em uma tabela é realizada pelo comando SELECT. Exemplo:

```
1 static async select(){
2   const connect = await db.connect();
3   return await connect.query('select * from clientes');
4 }
```

2.2.3.3 Update

Comando utilizado para a atualização de um ou mais registros de uma tabela. Exemplo:

```
1 static async update(id, data){
2   const connect = await db.connect();
3   const sql = 'UPDATE clientes SET nome=$1, idade=$2, uf=$3 WHERE id=$4';
4   const values = [data.nome, data.idade, data.uf, id];
5   return await connect.query(sql, values);
6 }
```

2.2.3.4 Delete

Comando utilizado para a exclusão de registro (s) de uma tabela. Exemplo:

```
1  static async delete(id){
2      const connect = await db.connect();
3      const sql = 'DELETE FROM clientes where id=$1;';
4      return await connect.query(sql, [id]);
5  }
```

2.2.4 Invocando os métodos nas rotas

Para que possamos invocar os métodos de manipulação de dados do modelo, precisamos criar uma rota. Para tanto, crie uma nova rota utilizando o **express** e adicione a seguinte rota para que seja feita a seleção dos dados.

```
1  var express = require('express');
2  var router = express.Router();
3  // Invocando o modelo Autor
4  const Autor = require("../models/autor");
5
6  /* Listando os usuários e apresentando um Json */
7  router.get('/', async function(req, res, next) {
8      const data = await Autor.select();
9      res.json(data.rows);
10 });
11
12 module.exports = router;
```

3 Mapeamento objeto-relacional (ORM)

É melhor você tentar algo, vê-lo não funcionar e aprender com isso, do que não fazer nada.

Mark Zuckerberg

Criar um banco de dados, ou mais formalmente o Sistema Gerenciador de Banco de Dados (SGBD), é uma tarefa complexa presente em todos os projetos. Escolher qual será utilizar em meio a tantos como: **mysql**, **sqlserver**, **oracle**, **sqlite**, **postgre**, **mongodb** e **etc**, ou, qual se adapta melhor ao projeto é uma decisão que deve ser tomada com cautela.

Outro fato é a forma com que os dados são tratados em cada um. Por exemplo, para um campo **id** que deve ser auto-incrementado, no Postgres usando o seguinte código SQL:

```
1 CREATE TABLE Pessoas (  
2     id SERIAL NOT NULL,  
3     nome varchar  
4 );
```

Já no Mysql, o mesmo código seria escrito da seguinte forma:

```
1 CREATE TABLE Pessoas (  
2     id int NOT NULL AUTO_INCREMENT,  
3     nome varchar  
4 );
```

Logo, caso houvesse uma mudança, nosso código teria que ser alterado para satisfazer a nova base de dados com todas as suas diferenças.

Assim, pensando nas possíveis mudanças que o projeto pode ter durante sua vida útil, foram desenvolvidas algumas técnicas para facilitar a vida dos desenvolvedores. Uma que iremos comentar é o Object Relational Mapping (ORM) ou Mapeamento Objeto Relacional. Para diminuir a complexidade, já que o ORM torna o banco de dados mais próximo da arquitetura de classe, removendo os comando SQL de vista, para que possamos focar em “Qual é o fluxo que minha aplicação deve seguir” e deixando de lado “Qual é a

query que eu deveria usar aqui?”. Então, nesse aspecto, iremos abordar um pouco sobre o que são ORM, suas práticas e exemplos usando JavaScript.

3.1 ORM: o que é e como funciona?

Como já citei anteriormente um ORM é um Mapeamento Objeto Relacional, sua base consiste em manter o uso de orientação a objetos e um pouco do conceito de non-query. Pois serão raros os momentos onde teremos que escrever uma linha de código SQL para esse tipo de ferramenta [Muramatsu 2020].

Outro fato muito importante e curioso sobre os ORM é que eles operam como um agente de banco de dados, sendo possível através de pouquíssimas mudanças, utilizar o mesmo código para mais de um banco de dados. Não importa se ele está em Mysql, SqlServer ou até mesmo Oracle. Ele consegue agir da mesma forma em alguns bancos de dados, você só precisa mudar o driver de conexão e está pronto para uso. Neste conceito é importante lembrar que cada uma de nossas tabelas são vistas como uma instância de uma classe, tendo suas características declaradas diretamente na sua classe “esquema”. Quando trabalhamos com esse tipo de esquema sempre teremos um arquivo de configuração, responsável por fornecer os dados para que o componente de ORM possa se comunicar com o banco e aplicação. Uma outra questão que pode causar dúvidas é como gerar o banco de dados através dessas classes que comentamos anteriormente. Bom veremos isso na prática mais a abaixo [Muramatsu 2020].

Referências

ELMASRI, R. et al. *Sistemas de banco de dados*. Pearson Addison Wesley São Paulo, 2005. Citado na página 3.

MURAMATSU, A. *Introdução a ORM no Node.js com Sequelize + exemplo prático*. 2020. Disponível em: <<https://ezdevs.com.br/introducao-a-orm-no-node-js-com-sequelize/>>. Acessado em: 11/04/2022. Citado na página 17.

SANCHES, A. R. *Disciplina: Fundamentos de Armazenamento e Manipulação de Dados*. 2005. Acessado em:. Disponível em: <<https://www.ime.usp.br/~andrrs/aulas/bd2005-1/aula7.html>>. Citado 3 vezes nas páginas 5, 6 e 7.

TAKAI, O. K.; ITALIANO, I. C.; FERREIRA, J. E. *Introdução a banco de dados. Departamento de Ciências da Computação. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo*, 2005. Citado 3 vezes nas páginas 3, 4 e 5.