

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 14 de março de 2022

# 1 Criação e acesso a banco de dados relacional usando NodeJS

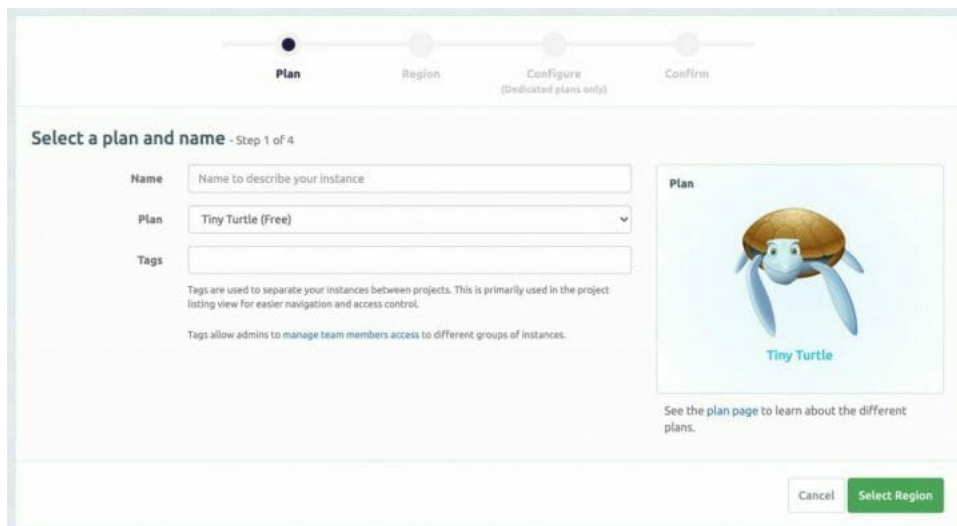
*Hoje você acha cansativo, porém mais tarde receberá a recompensa por todo esse tempo que passou estudando.*

**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.

## 1.1 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. 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 1** – 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.

### 1.1.1 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  );
```

### 1.1.2 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:foreignkey, empregado:foreignkey)

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

## 1.2 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**.

### 1.2.1 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
```

### 1.2.2 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.

### 1.2.3 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)<sup>1</sup>.

#### 1.2.3.1 Create

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

<sup>1</sup> 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   }
}
```

#### 1.2.3.2 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 }
```

#### 1.2.3.3 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 }
```

#### 1.2.3.4 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  }
```

### 1.2.4 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;
```



## Referências

ELMASRI, R. et al. Sistemas de banco de dados. Pearson Addison Wesley São Paulo, 2005. Nenhuma citação no texto.

SANCHES, A. R. *Disciplina: Fundamentos de Armazenamento e Manipulação de Dados*. 2005. Acessado em:. Disponível em: <<https://www.ime.usp.br/~andrers/aulas/bd2005-1/aula7.html>>. Nenhuma citação no texto.

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. Nenhuma citação no texto.