

**GEX613 – Programação II**  
**Padrão *Model-View-Controller***  
***Conexão com o Banco de Dados***



1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. Giancarlo Salton

# Conexão com o PostgreSQL

Para conectarmos com o banco de dados no backend, precisamos utilizar bibliotecas de código que nos permitam armazenar e consultar os dados de forma rápida e fácil.

No caso do PostgreSQL, utilizaremos a biblioteca `pg-promise`

<https://www.npmjs.com/package/pg-promise>

## Com npm

```
npm install pg-promise
```

## Com yarn

```
yarn add pg-promise
```

```
const pgp = require("pg-promise")({});

const usuario = "seu_usuario";
const senha = "sua_senha";
const host = "ip_do_server_postgresql";
const porta = "porta_do_server_postgresql";
const banco_de_dados = "nome_do_banco_do_seu_app";

const db = pgp(`postgres://${usuario}:${senha}@${host}:${porta}/${banco_de_dados}`);
```

```
const pgp = require("pg-promise")({});

const usuario = "seu_usuario";
const senha = "sua_senha";
const host = "ip_do_server_postgresql";
const porta = "porta_do_server_postgresql";
const banco_de_dados = "nome_do_banco_do_seu_app";

const db = pgp(`postgres://${usuario}:${senha}@${host}:${porta}/${banco_de_dados}`);
```

*"string"* de conexão com o PostgreSQL,  
caso seja necessário alterar algum  
padrão desta conexão

como não retornamos informação, o método utilizado é o `none`

```
await db.none(  
    "INSERT INTO clientes (nome, email) VALUES ($1, $2);",  
    [clienteNome, clienteEmail]  
);
```

instrução que será executada em SQL

```
await db.none(  
    "INSERT INTO clientes (nome, email) VALUES ($1, $2);",  
    [clienteNome, clienteEmail]  
);
```



```
db.none(  
    "INSERT INTO clientes (nome, email) VALUES ($1, $2);",  
    [clienteNome, clienteEmail]  
);
```

parâmetros que substituirão os coringas `$1` e `$2`. Note que se houver mais do que 1 parâmetro, estes devem ser passados através de um array.

**não concatene strings para inserir valores!**

db.**any**: 0 ou mais resultados

db.**all**: retorna 1 ou mais resultados. Se houverem 0 registros, retorna erro.

db.**one**: apenas 1 resultado. Se houver mais do que 1, retorna erro.

```
const clientes = await db.any("SELECT * FROM clientes;");
```

```
const clientes = await db.all("SELECT * FROM clientes;");
```

```
const clientes = await db.one(  
    "SELECT id, nome, email FROM clientes WHERE id = $1;",  
    clienteId  
);
```

retorna 0 ou mais clientes

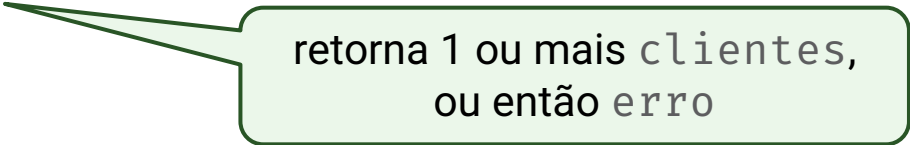
```
const clientes = await db.any("SELECT * FROM clientes;");
```

```
const clientes = await db.all("SELECT * FROM clientes;");
```

```
const clientes = await db.one(  
  "SELECT id, nome, email FROM clientes WHERE id = $1;",  
  clienteId  
);
```

```
const clientes = await db.any("SELECT * FROM clientes;");
```

```
const clientes = await db.all("SELECT * FROM clientes;");
```



retorna 1 ou mais clientes,  
ou então erro

```
const clientes = await db.one(  
  "SELECT id, nome, email FROM clientes WHERE id = $1;",  
  clienteId  
);
```

```
const clientes = await db.any("SELECT * FROM clientes;");
```

```
const clientes = await db.all("SELECT * FROM clientes;");
```

```
const clientes = await db.one(  
    "SELECT id, nome, email FROM clientes WHERE id = $1;",  
    clienteId  
);
```

retorna apenas 1 cliente ou  
então erro

```
const clientes = await db.any("SELECT * FROM clientes;");
```

```
const clientes = await db.all("SELECT * FROM clientes;");
```

```
const clientes = await db.one(  
  "SELECT id, nome, email FROM clientes WHERE id = $1;",  
  clienteId  
);
```

perceba que sendo apenas um  
parâmetro, não precisa estar  
em um array