NodeJS

Acesso a bases de dados MySQL

1 Introdução

Neste material, estudaremos sobre o acesso a bases de dados relacionais gerenciadas pelo MySQL a partir do NodeJS.

2 Instalação do MySQL

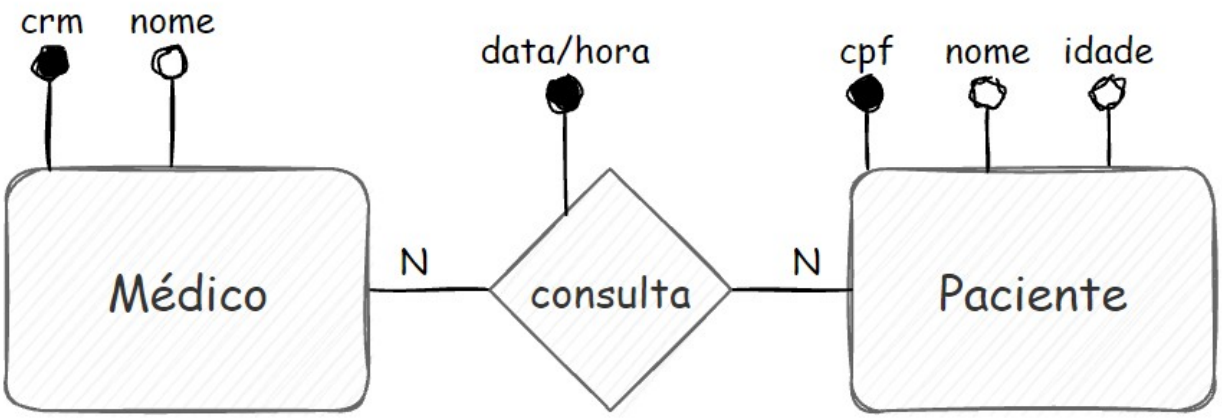
Certifique-se de que você possui o MySQL Community Server instalado. Também pode ser útil fazer a instalação de um cliente como o MySQL Workbench. Visite o Link 2.1 para obter os instaladores.

Link 2.1
https://dev.mysql.com/downloads

3 Modelo de dados

Utilizaremos o modelo de dados retratado pela Diagrama Entidade-Relacionamento da Figura 3.1.

Figura 3.1



4 Comandos SQL

Dado que utilizaremos o MySQL, que é um Sistema Gerenciador de Bancos de Dados **Relacional**, utilizaremos **relações** – ou seja, tabelas – para implementar aquilo o DER da Figura 3.1 descreve **conceitualmente**, ou seja, independente de implementação. Use os comandos do Bloco de Código 4.1 para criar a base de dados e informar ao MySQL Server que desejamos que os comandos a seguir tenham impacto sobre ela.

Bloco de Código 4.1

```
-- cria a base
CREATE DATABASE IF NOT EXISTS hospital;

-- informa ao MySQL Server qual a base sobre a qual desejamos operar
USE hospital;
```

Os comandos do Bloco de Código 4.2 criam as tabelas necessárias.

Bloco de Código 4.2

```
-- cria tabela medico
CREATE TABLE IF NOT EXISTS tb_medico (
        crm INT PRIMARY KEY,
   nome VARCHAR(200) NOT NULL
);

-- cria tabela paciente
CREATE TABLE IF NOT EXISTS tb_paciente (
        cpf BIGINT PRIMARY KEY,
   nome VARCHAR (200) NOT NULL,
   idade SMALLINT NOT NULL
);

-- cria tabela consulta
CREATE TABLE IF NOT EXISTS tb_consulta (
        crm INT NOT NULL,
   cpf BIGINT NOT NULL,
   data_hora DATETIME NOT NULL,
   PRIMARY KEY (crm, cpf, data_hora),
   CONSTRAINT fk_medico FOREIGN KEY (crm) REFERENCES tb_medico(crm),
   CONSTRAINT fk_paciente FOREIGN KEY (cpf) REFERENCES tb_paciente(cpf)
);
```

Faça também a inserção de alguns médicos, pacientes e consultas, como no Bloco de Código 4.3.

Bloco de Código 4.3

```
INSERT INTO tb_medico (crm, nome) VALUES (12345, 'José');
INSERT INTO tb_paciente (cpf, nome, idade) VALUES (998877, 'Maria', 22);
INSERT INTO tb_paciente (cpf, nome, idade) VALUES (11111111, 'Antônio', 30);

INSERT INTO tb_consulta (crm, cpf, data_hora) VALUES (12345, 998877, '2021-10-12 13:53:00');
INSERT INTO tb_consulta (crm, cpf, data_hora) VALUES (12345, 998877, '2021-10-13 18:00:00');
INSERT INTO tb_consulta (crm, cpf, data_hora) VALUES (12345, 11111111, '2021-10-17 22:00:00');
```

Veja alguns exemplos de consultas no Bloco de Código 4.4.

Bloco de Código 4.4

```
-- todos os médicos
SELECT * FROM tb_medico;
-- todos os pacientes
SELECT * FROM tb_paciente;
-- todas as consultas
SELECT * FROM tb_consulta;

-- todas as consultas ordenadas por data
SELECT * FROM tb_consulta ORDER BY data_hora;

-- todas as consultas ordenadas por data, mais recentes primeiro
SELECT * FROM tb_consulta ORDER BY data_hora DESC;

-- todos os médicos associados a suas consultas
SELECT
        m.crm, m.nome, c.data_hora
FROM
        tb_medico m
  INNER JOIN tb_consulta c
        ON m.crm = c.crm;

-- todos os médicos associados a suas consultas e pacientes
SELECT
        m.crm, m.nome 'nome_medico', c.data_hora, p.nome as 'nome_paciente'
FROM
        tb_medico m
  INNER JOIN tb_consulta c
        ON m.crm = c.crm
  INNER JOIN tb_paciente p
  ON c.cpf = p.cpf;

-- totais de consultas por nome
SELECT
        p.nome, COUNT(*) as 'total_consultas'
FROM
        tb_consulta c
  INNER JOIN tb_paciente p
  ON c.cpf = p.cpf
  GROUP BY p.nome;
```

5 Projeto NodeJS

Crie uma pasta para o seu projeto NodeJS. Com um terminal vinculado a ela, use

npm init -y

para criar um projeto. Use

code .

para abrir uma instância do VS Code vinculada ao diretório atual. No VS Code, clique **Terminal >> New Terminal** para obter um terminal interno do VS Code, o que facilita os trabalhos.

5.1 (Dependências: MySQL) Para utilizar o MySQL com o NodeJS, precisamos fazer a instalação de um pacote que desempenhará o papel de "driver". Há diversos pacotes próprios para isso. Dois dos mais comuns se chamam **mysql** e **mysql2**. Vejas os links 5.1.1 e 5.1.2.

Link 5.1.1
https://www.npmjs.com/package/mysql

Link 5.1.2
https://www.npmjs.com/package/mysql2

No momento em que este documento está sendo escrito, ambos possuem em torno de 700 mil downloads semanais. O primeiro teve a sua última versão publicada há dois anos. O segundo teve a sua última versão publicada há 3 dias. Aparentemente, o segundo é mantido com maior frequência. Além disso, pesquisas na Internet revelam que, de fato, ele pode ser mais moderno. Por essa razão, optaremos por utilizá-lo. Vá em frente e faça a sua instalação com

npm install mysql2

5.2 (Dependências: express e nodemon) Utilizaremos o express para o tratamento de requisições HTTP e o nodemon para simplificar a execução do programa. Instale ambos com

npm install express

e

npm install nodemon --save-dev

Abra o arquivo **package.json** e adicione o script destacado no Bloco de Código 5.2.1.

Bloco de Código 5.2.1

```
{
  "name": "medico_paciente_consulta",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "mysql2": "^2.3.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.13"
  }
}
```

Na raiz do projeto - lado a lado com o arquivo **package.json** - crie um arquivo chamado **index.js**. Usando um terminal interno do VS Code, coloque o programa em execução com

npm start

5.3 (Servidor HTTP com express) Nosso programa será um servidor HTTP que recebe requisições e se comunica com um servidor MySQL. Veja sua definição inicial no Bloco de Código 5.3.1.

Bloco de Código 5.3.1

```
const express = require ('express')
const app = express()
app.use(express.json())

const porta = 3000
app.listen(porta, () => console.log(`Executando. Porta ${porta}`))
```

5.4 (Endpoint: Obtenção de todos os médicos) O Bloco de Código 5.4.1 mostra um endpoint capaz de trazer todos os médicos da base. Esta versão apenas exibe os dados – e os dados sobre os dados – no console.

Bloco de Código 5.4.1

```
const express = require ('express')
//mysql é o nome de uma variável, pode ser qualquer coisa
//mysql parece mais intuitivo do que mysql2
const mysql = require('mysql2')
const app = express()
app.use(express.json())
app.get('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })

    connection.query('SELECT * FROM tb_medico', (err, results, fields) => {
        //results tem as linhas
        //fields tem meta dados sobre os resultados, caso estejam disponível
        console.log(results)
        console.log(fields)
        res.send('ok')
    })

})
const porta = 3000
app.listen(porta, () => console.log(`Executando. Porta ${porta}`))
```

Exercícios

1. Ajuste o endpoint de médicos para que ele devolva um objeto JSON ao cliente.

2. Escreva um endpoint que permita a obtenção de todos os pacientes.

3. Escreva um endpoint que permita a inserção de um novo médico. Os dados do médico devem ser enviados por meio da requisição. Pesquise a documentação e escreva o seu comando SQL usando placeholders – ou seja, símbolos "?".

4. Ajuste a aplicação para que ela passe a usar o pacote **dotenv**. Escreva os dados de acesso ao banco em um arquivo à parte e faça a configuração necessária.

5.5 (Endpoint: Obtenção de todos os médicos) O Bloco de Código 5.5.1 mostra o endpoint de obtenção de todos os médicos ajustado. Ele devolve a lista de médicos ao cliente.

Bloco de Código 5.5.1

```
app.get('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })

    connection.query('SELECT * FROM tb_medico', (err,
results, fields) => {
        res.json(results)
    })

})
```

5.6 (Endpoint: Obtenção de todos os pacientes) O Bloco de Código 5.6.1 mostra um endpoint que permite a obtenção de todos os pacientes.

Bloco de Código 5.6.1

```
const express = require ('express')
//mysql é o nome de uma variável, pode ser qualquer coisa
//mysql parece mais intuitivo do que mysql2
const mysql = require('mysql2')
const app = express()
app.use(express.json())

app.get('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })

    connection.query('SELECT * FROM tb_medico', (err, results, fields) => {
        res.json(results)
    })

})

app.get('/pacientes', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })

    connection.query('SELECT * FROM tb_paciente', (err, results, fields) =>
{
        res.json(results)
    })
})

const porta = 3000
```

```
app.listen(porta, () => console.log(`Executando. Porta ${porta}`))
```

5.6 (Endpoint: Inserção de um médico) A inserção de um médico requer que os seus dados – crm e nome – sejam enviados via requisição. Uma vez extraídos da requisição, precisamos encaixar esses valores no comando SQL para fazer a sua execução. Há diversas formas para fazê-lo. A mais simples talvez seja a simples concatenação. Veja o Bloco de Código 5.6.1.

Bloco de Código 5.6.1

```
app.post('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })
    const crm = req.body.crm
    const nome = req.body.nome
    const sql = "INSERT INTO tb_medico (crm, nome) VALUES ("
+ crm + ", '" + nome + "')"
    connection.query(sql, (err, results, fields) => {
        console.log (results)
        console.log(fields)
        res.send('ok')
    })
})
```

Observe como a concatenação pode ser um tanto inconveniente. Podemos fazer a mesma operação de uma maneira mais elegante, usando **placeholders**. Veja o Bloco de Código 5.6.2.

Bloco de Código 5.6.2

```
app.post('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'rodrigo',
        database: 'hospital',
        password: '1234'
    })
    const crm = req.body.crm
    const nome = req.body.nome
    const sql = "INSERT INTO tb_medico (crm, nome) VALUES
(?, ?)"
    connection.query(
        sql,
        [crm, nome],
        (err, results, fields) => {
            console.log (results)
            console.log(fields)
            res.send('ok')
    })
})
```

5.7 (Dotenv) Não é boa ideia manter os dados de acesso à base de dados misturados com o código, já que poderemos fazer o controle de versão utilizando um repositório público e, também, eles poderão ser alterados dependendo do ambiente em que o sistema for colocado em execução (desenvolvimento, teste, homologação, produção etc.). Assim, Vamos utilizar o pacote dotenv para especificar esses valores em um arquivo à parte e torná-los acessíveis por meio de variáveis de ambiente. O primeiro passo é instalar o pacote dotenv com

npm install dotenv

Depois disso, importe o pacote e já faça a configuração como mostra o Bloco de Código 5.7.1.

Bloco de Código 5.7.1

```
require ('dotenv').config()
const express = require ('express')
…
```

Agora, crie um arquivo chamado **.env** na raiz da aplicação. Veja seu conteúdo no Bloco de Código 5.7.2.

Bloco de Código 5.7.2

```
DB_USER=rodrigo
DB_PASSWORD=1234
DB_HOST=localhost
DB_DATABASE=hospital
```

Cada variável fica acessível como uma propriedade do objeto **process.env**. Podemos, portanto, obtê-las como mostra o Bloco de Código 5.7.3.

Bloco de Código 5.7.3

```
require ('dotenv').config()
const express = require ('express')
//mysql é o nome de uma variável, pode ser qualquer coisa
//mysql parece mais intuitivo do que mysql2
const mysql = require('mysql2')
const app = express()
app.use(express.json())

const {DB_USER, DB_PASSWORD, DB_HOST, DB_DATABASE} = process.env
…
```

A seguir, passamos a utilizá-las quando ao criar conexões com o banco. Veja o Bloco de Código 5.7.4.

Bloco de Código 5.7.4

```
...
app.get('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: DB_HOST,
        user: DB_USER,
        database: DB_DATABASE,
        password: DB_PASSWORD
    })

    connection.query('SELECT * FROM tb_medico', (err, results, fields) => {
        res.json(results)
    })

})

app.post('/medicos', (req, res) => {
    const connection = mysql.createConnection({
        host: DB_HOST,
        user: DB_USER,
        database: DB_DATABASE,
        password: DB_PASSWORD
    })
    const crm = req.body.crm
    const nome = req.body.nome
    const sql = "INSERT INTO tb_medico (crm, nome) VALUES (?, ?)"
    connection.query(
        sql,
        [crm, nome],
        (err, results, fields) => {
            console.log (results)
            console.log(fields)
            res.send('ok')
    })
})

app.get('/pacientes', (req, res) => {
    const connection = mysql.createConnection({
        host: DB_HOST,
        user: DB_USER,
        database: DB_DATABASE,
```

```
        password: DB_PASSWORD
    })

    connection.query('SELECT * FROM tb_paciente', (err, results, fields) => {
        res.json(results)
    })
})

const porta = 3000
app.listen(porta, () => console.log(`Executando. Porta ${porta}`))
```

Crie também um arquivo chamado .gitignore na raiz do projeto. Assim instruímos o Git a ignorar este arquivo. Aproveite também e instrua o Git a ignorar a pasta node_modules. O conteúdo do arquivo .gitignore aparece no Bloco de Código 5.7.5.

Bloco de Código 5.7.5

```
node_modules
.env
```

5.8 (Endpoint: Obtenção de consultas) No endpoint exibido no Bloco de Código 5.8.1, fazemos uma busca por consultas, incluindo nome de médico e de paciente também.

Bloco de Código 5.8.1

```
app.get('/consultas', (req, res) => {
    const connection = mysql.createConnection({
        host: DB_HOST,
        user: DB_USER,
        database: DB_DATABASE,
        password: DB_PASSWORD
    })
    const sql = `
        SELECT
            m.nome as nome_medico, c.data_hora, p.nome as
nome_paciente
        FROM
            tb_medico m, tb_consulta c, tb_paciente p
        WHERE
```

```
            m.crm = c.crm AND c.cpf = p.cpf
    `
    connection.query(
        sql,
        (err, results, fields) => {
            res.json(results)
        }
    )
})
```

5.9 (Pool de conexões) Uma das atividades mais custosas quando estamos falando de acesso a bases de dados, é a abertura das conexões. Nossa solução, até então, abre uma conexão a cada requisição recebida. Essa estratégia tem alguns pontos que podem ser melhorados.

I. A aplicação abre conexões sem um limite definido. Sob forte demanda, ela pode abrir um número muito grande de conexões simultaneamente e levar o SGBD ao colapso.

II. Depois de utilizada, uma conexão aberta é simplesmente descartada.

Um pool de conexões é um mecanismo que possui características como as seguintes.

I. Permite a especificação de um número de conexões a ser utilizado pela aplicação.

II. Abre conexões sob demanda.

III. Conexões utilizadas voltam ao pool para serem reutilizadas.

IV. Possui uma fila de espera. Quando recebe uma requisição por uma conexão, ele entrega uma que estiver ociosa. Se todas estiverem ocupadas, o solicitante fica bloqueado esperando para receber uma.

O uso de um pool de conexões é muito simples. Basta construí-lo e, a seguir, usar o seu método **query** para executar os comandos SQL. O Bloco de Código 5.9.1 mostra como um pool de conexões pode ser construído.

Bloco de Código 5.9.1

```
require ('dotenv').config()
const express = require ('express')
//mysql é o nome de uma variável, pode ser qualquer coisa
//mysql parece mais intuitivo do que mysql2
const mysql = require('mysql2')
const app = express()
app.use(express.json())

const {DB_USER, DB_PASSWORD, DB_HOST, DB_DATABASE} = process.env
const pool = mysql.createPool({
    host: DB_HOST,
    user: DB_USER,
    password: DB_PASSWORD,
    database: DB_DATABASE,
    //se todas as conexões estiverem ocupadas, novos solicitantes
esperam numa fila
    //se configurado com false, causa um erro quando recebe requisições
e todas
    //as conexões estão ocupadas
    waitForConnections: true,
    //no máximo 10 conexões. Elas são abertas sob demanda e não no
momento de
    //construção do pool
    connectionLimit: 10,
    //quantos solicitantes podem aguardar na fila? 0 significa que não
há limite
    queueLimit: 0
})
...
```

Agora podemos ajustar nossos endpoints para que usem o pool. Veja o Bloco de Código 5.9.2. Observe que as instruções "createConnection" foram todas removidas.

Bloco de Código 5.9.2

```
app.get('/medicos', (req, res) => {
    pool.query('SELECT * FROM tb_medico', (err, results, fields) => {
        res.json(results)
    })

})

app.post('/medicos', (req, res) => {
    const crm = req.body.crm
    const nome = req.body.nome
    const sql = "INSERT INTO tb_medico (crm, nome) VALUES (?, ?)"
    pool.query(
        sql,
        [crm, nome],
        (err, results, fields) => {
            console.log (results)
            console.log(fields)
            res.send('ok')
    })
})

app.get('/pacientes', (req, res) => {
    pool.query('SELECT * FROM tb_paciente', (err, results, fields) => {
        res.json(results)
    })
})

app.get('/consultas', (req, res) => {
    const sql = `
        SELECT
            m.nome as nome_medico, c.data_hora, p.nome as nome_paciente
        FROM
            tb_medico m, tb_consulta c, tb_paciente p
        WHERE
            m.crm = c.crm AND c.cpf = p.cpf
    `
    pool.query(
        sql,
        (err, results, fields) => {
            res.json(results)
        }
    )})
```

Exercícios

Nossos endpoints fazem uso de funções callback, o que pode dar origem ao conhecido "callback hell". Visite a documentação do pacote mysql2 e pesquise sobre o uso de promises. Reescreva os endpoints para que eles utilizem promises. A documentação pode ser encontrada no Link 5.9.1.

Link 5.9.1
https://www.npmjs.com/package/mysql2