

BANCO DE DADOS COM NODE JS



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifroma@gmail.com

Slides: <https://git.io/vbU3N>

AGENDA

- Banco de dados SQL
 - MySQL
 - SQLite
- Sequelize JS
- Definição dos modelos
- Usando os modelos
- Consultas
- Associações
- Transações
- Referências

BANCO DE DADOS RELACIONAL

- Assim como em linguagens como Java e C#, no NodeJS é preciso instalar o módulo (driver) para conexão com o banco de dados desejado.
- Para consultar a disponibilidade do banco de dados desejado, consulte o nome do banco no repositório <http://npmjs.com>
- Por exemplo, para MySQL podemos utilizar o módulo `mysql2` e para SQLite o módulo `sqlite3`.

```
$ npm install --save sqlite3
```

```
$ npm install --save mysql2
```

EXEMPLO MYSQL

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost', user: 'root',
  password: '1234', database: 'test'
});

connection.query(
  'SELECT * FROM `table` WHERE `name` = "Page" AND `age` > 45',
  function (err, results, fields) {
    console.log(results); // resultado contendo as linhas da consulta
    console.log(fields); // campos contendo metadados sobre os resultados
  }
);
```

EXEMPLO MYSQL

- Prepared Statements

```
connection.execute(  
  'SELECT * FROM `table` WHERE `name` = ? AND `age` > ?',  
  ['Rick C-137', 53],  
  function (err, results, fields) {  
    console.log(results); // resultado contendo as linhas da consulta  
    console.log(fields); // campos contendo meta dados sobre o resultados  
  }  
);
```

EXEMPLO SQLITE

```
const sqlite3 = require('sqlite3');
const db = new sqlite3.Database('database.sqlite');

db.serialize(function () {
  db.run("CREATE TABLE lorem (info TEXT)");

  const stmt = db.prepare("INSERT INTO lorem VALUES (?)");
  for (let i = 0; i < 10; i++) {
    stmt.run("Ipsum " + i);
  }
  stmt.finalize();

  db.each("SELECT rowid AS id, info FROM lorem", function (err, row) {
    console.log(row.id + ": " + row.info);
  });
});
```

SEQUELIZE JS



Sequelize

- Sequelize é um framework para Mapeamento de Objetos Relacionais (ORM) para Node JS.
- Possui suporte à PostgreSQL, MySQL, SQLite e MSSQL.
- Possui funcionalidades sólidas como relacionamento entre entidades e transações.

INTRODUÇÃO AO SEQUELIZE JS

- Para iniciar com o Sequelize basta instalar o módulo:

```
$ npm install --save sequelize
```

- Em seguida, instalar o driver para o banco de dados desejado:

```
$ npm install --save pg pg-hstore
```

```
$ npm install --save mysql2
```

```
$ npm install --save sqlite3
```

```
$ npm install --save tedious // MSSQL
```


INTRODUÇÃO AO SEQUELIZE JS

- Conectando ao banco de dados SQLite.

```
const Sequelize = require('sequelize');  
  
const sequelize = new Sequelize(null, null, null, {  
  dialect: 'sqlite',  
  storage: './database.sqlite',  
});
```

- Conectando ao banco de dados MySQL.

```
const sequelize = new Sequelize('database', 'user', 'password', {  
  dialect: 'mysql',  
  host: '172.0.0.1',  
  port: '3306'  
});
```

INTRODUÇÃO AO SEQUELIZE JS

- Testando a conexão.

```
sequelize.authenticate()  
  .then(() => {  
    console.log('Banco de dados conectado com sucesso.');
```

```
  }).catch((ex) => {  
    console.error("Não foi possível se conectar ao banco de dados.", ex);  
  });
```

DEFINIÇÃO DOS MODELOS

- Vamos definir a entidade que representará a tabela “usuario” no banco de dados.

```
const Usuario = sequelize.define('usuario', {  
  id: {  
    primaryKey: true,  
    type: Sequelize.BIGINT,  
  },  
  nome: {  
    type: Sequelize.STRING(200),  
    allowNull: false,  
  },  
  nascimento: Sequelize.DATEONLY,  
  email: Sequelize.STRING(150),  
});
```

USANDO OS MODELOS

- Inserindo um elemento no banco de dados.

```
Usuario.create({  
  nome: 'Douglas Junior', email: 'nassifrroma@gmail.com'  
}).then(usuario => {  
  // você pode acessar agora o usuário criado  
  // através da variável "usuario"  
  console.log("Usuário inserido:", JSON.stringify(usuario));  
})
```

USANDO OS MODELOS

- Buscando por um elemento específico

```
Usuario.findById(123).then(usuario => {  
    // Retorna o usuário correspondente ao ID especificado,  
    // ou Null caso não seja encontrado.  
})  
  
Usuario.findOne({  
    where: {  
        nome: 'Douglas Junior'  
    }  
}).then(usuario => {  
    // Retorna o primeiro usuário com a condição especificada,  
    // ou Null caso não seja encontrado.  
})
```

USANDO OS MODELOS

- Atualizando um elemento no banco de dados.

```
// Maneira 1
usuario.nome = 'Douglas Nassif'
usuario.save().then(() => { });
```

```
// Maneira 2
usuario.update({
  nome: 'Douglas Nassif'
}).then(() => { });
```

```
// Maneira 3
Usuario.update({
  nome: 'Douglas Nassif'
},{
  where: {
    id: 123
  }
}).then(() => { });
```

USANDO OS MODELOS

- Excluindo um elemento no banco de dados.

```
// Maneira 1
usuario.destroy()
  .then(() => { });
```

```
// Maneira 2
Usuario.destroy({
  where: {
    id: 1
  }
}).then(() => { });
```

CONSULTAS

- Definindo quais atributos devem ser retornados na consulta.

```
Usuario.findAll({  
  attributes: ['nome', 'email']  
}).then(usuarios => {  
  console.log('Usuários selecionados:', JSON.stringify(usuarios));  
})
```


CONSULTAS

- Executando funções do banco de dados, como COUNT, MAX, MIN, etc.

```
Usuario.findAll({
  attributes: [[sequelize.fn('COUNT', sequelize.col('id')), 'qtd_usuarios']]
}).then(resultado => {
  console.log('Quantidade de usuários:', JSON.stringify(resultado));
})
```

CONSULTAS

- Filtrando consultas com “where”.

```
Usuario.findAll({  
  where: {  
    nome: {  
      $like: '%douglas%'  
    }  
  }  
}).then(usuarioes => {  
  console.log('Usuários selecionados:', JSON.stringify(usuarioes));  
})
```

CONSULTAS

- Filtrando e contando o total de registros, útil para uso em paginação.

```
Usuario.findAndCountAll({  
  where: { },  
  limit: 10,  
  offset: 0,  
}).then(usuarios => {  
  console.log('Quantidade de usuários:', JSON.stringify(usuarios.count));  
  console.log('Usuários selecionados:', JSON.stringify(usuarios.rows));  
})
```

ASSOCIAÇÕES

- Associar a entidade “usuario” à “tarefa”, onde a tarefa recebe a chave estrangeira do usuário.

```
// O usuário tem muitas tarefas
Usuario.hasMany(Tarefa, {
  onDelete: 'NO ACTION',
  onUpdate: 'NO ACTION'
})

// A tarefa tem a chave estrangeira do usuário
Tarefa.belongsTo(Usuario, {
  onDelete: 'NO ACTION',
  onUpdate: 'NO ACTION'
});
```

ASSOCIAÇÕES

- Consultando com JOINS.

```
Usuario.findAll({  
  where: { },  
  include: [{  
    model: Tarefa,  
    required: true, // true para inner join, false para left join  
  }],  
}).then(usuarios => {  
  console.log('Usuários com tarefas:', JSON.stringify(usuarios));  
})
```

TRANSAÇÕES

- Transações são utilizadas para garantir a integração entre diversas ações no banco de dados.

```
sequelize.transaction((transaction) => {  
  return Usuario.create({  
    nome: 'Douglas Junior', email: 'nassifrroma@gmail.com',  
  }, { transaction }).then(usuario => {  
    console.log('Usuário criado:', JSON.stringify(usuario));  
    return Tarefa.create({  
      titulo: 'Minha tarefa',  
      usuarioId: usuario.id  
    }, { transaction })  
  }).then(tarefa => {  
    console.log('Tarefa criada:', JSON.stringify(tarefa));  
  })  
}).then(() => {  
  console.log('transação comitada');  
}).catch(ex => {  
  console.error('transação revertida:', ex);  
})
```

REFERÊNCIAS

- Node MySQL2 - <https://github.com/sidorares/node-mysql2>
- Node SQLite3 - <https://github.com/mapbox/node-sqlite3>
- Sequelize JS - <http://docs.sequelizejs.com>
- Promises - https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise

DÚVIDAS?



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com

Slides: <https://git.io/vbU3N>