Manipulação de dados com sequelize

Adicionando objetos - CREATE.

Para adicionar objetos ao banco de dados, devemos utilizar o comando

create. O comando **create** irá incluir os dados, ou seja, nossas informações na tabela especificada no banco de dados. O comando total é:

```
const <nome_variavel> = await <nome_variavel_sequelize>.create({ <nome_campo1_tabela>:<valor1_dados>,<nome_campo2_tabela>:<valor2_dados>});
```

Exemplificando, a inserção de um valor na tabela “setor”, que criamos anteriormente, ficaria da seguinte forma:

```
const setor_c = await Setor.create({ idsetor: 2, nome: "Financeiro", ramal: "21345", email: "financeiro@empresa.com" });
```

Para testarmos, vamos utilizar o banco que criamos nas aulas anteriores chamado **empresa**.

Após o comando:

await sequelize.sync({ force: true });

Coloque o código abaixo:

```
const setor_create = await Setor.create({ nome: "Financeiro", ramal: "2134", email: "financeiro@empresa.com" });
```

O comando acima fará a inclusão dos dados na tabela "setor".

Agora, vamos incluir mais 2 setores utilizando a mesma lógica do anterior.

const setor_create_S = await Setor.create({ nome: "Secretaria", ramal: "2135", email: "secretaria@empresa.com" });

const setor_create_P = await Setor.create({ nome: "Portaria", ramal: "2136", email: "portaria@empresa.com" });

Listando objetos - READ.

Falaremos mais sobre o processo de listar objetos nas próximas aulas, por enquanto, vamos entender o princípio básico. Todos os objetos que são incluídos no banco de dados podem ser mostrados para nossos usuários em algum momento.

O retorno dos dados pode ocorrer por meio de relatórios ou retorno por variáveis.

O sequelize possui uma maneira bem tranquila de retornar as informações. Vamos ver como fazemos?

const setores_listar = await Setor.findAll();

O findall() disponibiliza um array do nosso objeto **Setor**. Assim, temos todas as informações que foram cadastradas na tabela “setor” alocadas na const **setores_listar**.

Para mostrar na tela, usamos o comando **console.log**

```
console.log("Lista de setores: \n", JSON.stringify(setores_listar, null, 2), "\n\n");
```

O JSON.stringify() é um dos vários métodos nativos da linguagem de programação JavaScript. Sua principal função é converter valores e objetos denotados na linguagem em uma String JSON, formando, assim, um conjunto de dados humanamente legível e representados apenas por conjuntos de caracteres.

O comando possui a seguinte estrutura:

JSON.stringify(**valor**, **replacer**, **space**)

Parâmetro value

O parâmetro valor tem a função de **identificar o que será convertido pelo método JSON.stringify().** Tudo que é declarado nessa primeira posição terá uma saída estruturada em JSON String. Colocamos nosso objeto 'setores_listar' como parâmetro 'valor' para ser convertido em uma string JSON pelo método JSON.stringify().

Parâmetro Replacer

O parâmetro **replacer** é opcional, ou seja, **não tem obrigatoriedade de ser declarado ao utilizar o método JSON.stringfy()**. O replacer pode ser usado para filtrar chaves da tabela.

```
console.log("Lista de setores: \n", JSON.stringify(setores_listar, ['idsetor'], 2), "\n\n");
```

O comando acima retornará somente os valores do campo **idsetor**.

O parâmetro Space

Esse parâmetro **é utilizado para controlar o espaçamento entre os dados de saída**.

No nosso exemplo, colocamos o número 2, ou seja, ao serem mostrados na tela, nossos dados terão espaçamento de 2 caracteres entre eles.

Arquivo Indez.js

Agora vamos executar nosso arquivo index.js. Nosso arquivo completo está abaixo.

```
// Importando as bibliotecas que iremos utilizar
const { Sequelize,Model, DataTypes } = require("sequelize");


//abrindo uma conexão
const sequelize = new Sequelize({
```

```
  dialect: "sqlite",
  storage: "empresa.sqlite"
});
// Definindo a classe setor
class Setor extends Model {
    static init(sequelize) {
        super.init({
            idsetor:{
              type: DataTypes.INTEGER,
              autoIncrement: true,
              allowNull: false,
              primaryKey: true
            },

            nome: {
              type: DataTypes.STRING(60),
              allowNull: false
            },

            ramal:{
              type: DataTypes.STRING(6)
            },
            email:{
              type: DataTypes.STRING(40)
            }
        }, { sequelize, modelName: 'setor', tableName: 'setores' })
    }
}
// inicializando o modelo create table
Setor.init(sequelize);

class Funcionario extends Model {
```

```
    static init(sequelize) {
        super.init({
            matricula:{
              type: DataTypes.INTEGER,
              autoIncrement: true,
              allowNull: false,
              primaryKey: true
            },

            Idsetor: {
              type: DataTypes.INTEGER,
              references: {
                model: Setor,
                key: 'idsetor'
              },
            },

            nome:{
              type: DataTypes.STRING(60),
              allowNull: false
            },
            nascimento:{
              type: DataTypes.DATE
            },
            telefone:{
              type: DataTypes.STRING(15)
            }
        }, { sequelize, modelName: 'funcionario', tableName: 'funcionarios' })
    }
}
// inicializando o modelo create table
Funcionario.init(sequelize);
```

```
(async () => {
  // Sincronizando automaticamente
  await sequelize.sync({ force: true });

  const setor_c = await Setor.create({ nome: "Financeiro", ramal: "2134", email: "financeiro@empresa.com" });
  const setor_S = await Setor.create({ nome: "Secretaria", ramal: "2135", email: "secretaria@empresa.com" });
  const setor_P = await Setor.create({ nome: "Portaria", ramal: "2136", email: "portaria@empresa.com" });

// Listando objetos da tabela Setor

  const setores_listar = await Setor.findAll();
  console.log("Lista de setores: \n", JSON.stringify(setores_listar, null, 2), "\n\n");
})();
```

Repare que o retorno ao executar o arquivo será:

Alterando objetos – UPDATE

Para atualizar os dados da nossa tabela, primeiro precisamos recuperá-los do banco de dados usando alguma função de **find** do Sequelize.

```
const setor_chave = await Setor.findByPk(3);
setor_chave.nome = "Estoque";
const resultado = await setor_chave.save();
console.log(resultado);
```

Criamos uma **const** chamada “setor_chave” para receber o comando Setor.findByPk(3).

Repare: o comando **findByPk()** é utilizado para localizar um único registro usando como filtro sua chave primária. No caso da tabela “setor”, utilizamos como chave primária o campo **idsetor**.

Este comando solicita que sejam recuperados apenas os dados do registro com código 3, que corresponde ao setor de portaria.

Assim, solicitamos que o campo “nome” receba o valor de Estoque.

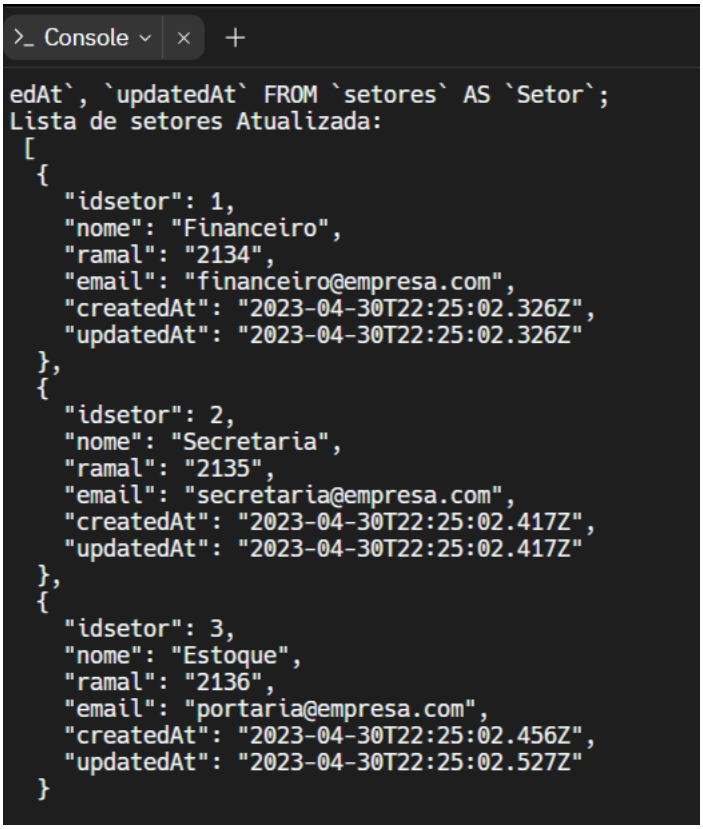
Criamos uma **const** chamada “resultado” e usamos o comando **save** da nossa variável instanciada.

Se adicionarmos o comando acima ao nosso código, o setor com **idsetor = 3** será renomeado para "Estoque".

Vamos listar os objetos novamente utilizando a técnica do **findall**.

```
setores_update = await Setor.findAll();
console.log("\nLista de setores atualizada: \n", JSON.stringify(setores_update, null, 2), "\n\n");
```

Agora veremos a lista atualizada.



Deletar objetos – DELETE

Assim como no comando **save**, em que é necessário especificar o objeto a ser salvo, no comando **delete** também é necessário especificar o objeto que será excluído.

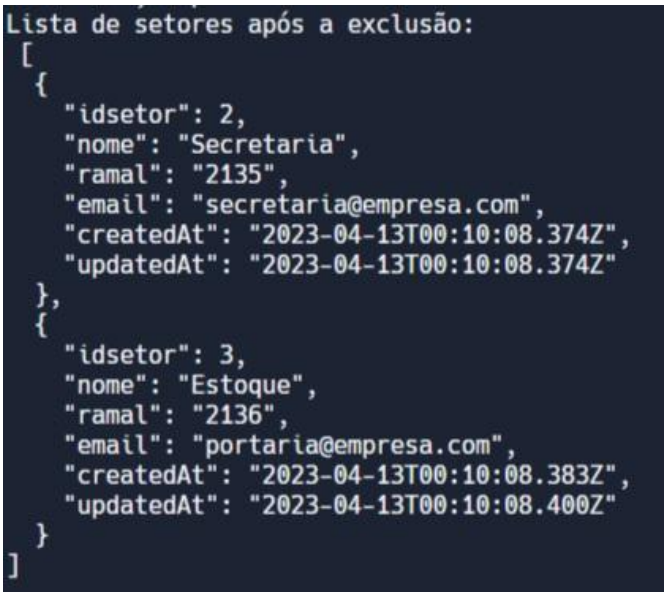
```
// Deletando objetos
const setor_delete = await Setor.findByPk(1);
setor_delete.destroy();
```

Primeiro, criamos uma variável chamada **setor_delete** para listar somente o registro que queremos deleter usando o comando **findByPk()**.
Escolhi o **idsetor** igual a 1 que é o setor financeiro.

Após a localização e inclusão dos dados, chamamos a variável **setor_delete.destroy()** e pronto, nosso registro foi escolhido.

Para visualizar a exclusão, basta criar uma lista nova de itens da tabela e mostrar na tela.

```
//listando objetos após a exclusão do setor 1 - Financeiro
    const setores_exclusao = await Setor.findAll();
    console.log("Lista de setores após a exclusão: \n",
JSON.stringify(setores_exclusao, null, 2), "\n\n");
```



Assim, finalizamos nossa aula de criar **(CREATE)**, listar **(READ)**, atualizar **(UPDATE)** e excluir **(DELETE)** objetos do banco de dados.

CREATE – READ – UPDATE – DELETE

Parabéns! Você criou seu primeiro CRUD. Em outro momento, vamos explorar esse assunto.

Até a próxima aula...

Fontes Bibliográficas

ZHAO, Alice. **SQL Guia Prático: Um guia para o uso de SQL.** Editora Novatec, 2023.

NEWMAN, Chris. **SQLite**. 1ª. Editora Sams, 2004.