

CRUD (Create, Read, Update e Delete) com Node JS

Prof. Me. Ewerton J. Silva

Introdução

Um CRUD se refere a um conjunto de operações básicas que podem ser realizadas em um banco de dados. CRUD é um acrônimo que representa as quatro operações fundamentais em relação aos dados: Create (Criação), Read (Leitura), Update (Atualização) e Delete (Exclusão). Essas operações permitem que você realize ações comuns em bancos de dados, como criar registros, ler dados existentes, atualizar informações e excluir entradas.

Após definir os métodos básicos em todos os controllers, daremos início a configuração dos métodos de persistência de dados.

Os códigos que iremos inserir para cada método devem ficar dentro do bloco “try”, conforme apontado ao lado.

```
3 module.exports = {
4   async listarUsuarios(request, response) {
5     try {
6       // instruções SQL
7       const sql = `SELECT
8         usu_id, usu_nome, usu_email, usu_dt_nasc, usu_senha,
9         usu_tipo, usu_ativo = 1 AS usu_ativo
10        FROM usuarios`;
11      // executa instruções SQL e armazena o resultado na variável usuarios
12      const usuarios = await db.query(sql);
13      const nItens = usuarios[0].length;
14
15      return response.status(200).json({
16        sucesso: true,
17        mensagem: 'Lista de usuários.',
18        dados: usuarios[0],
19        nItens
20      });
21    } catch (error) {
```

Read – Neste exemplo listaremos todos os usuários cadastrados.

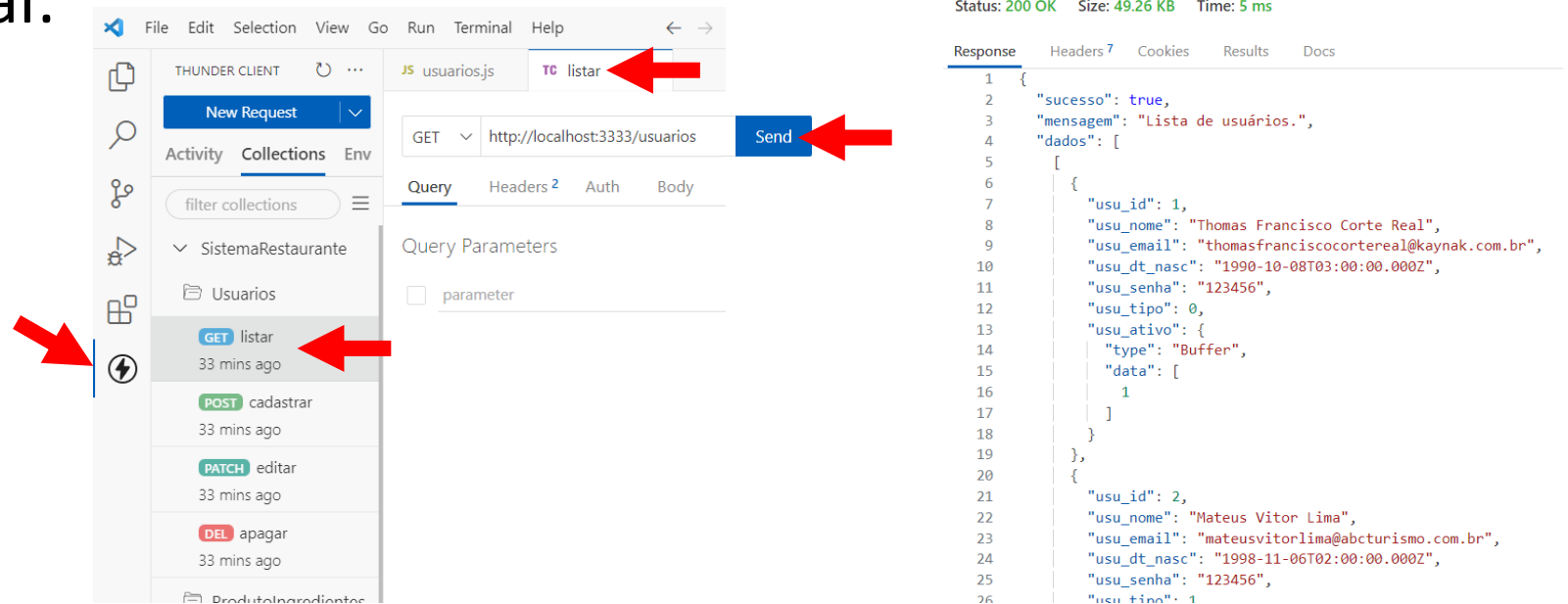
```
controllers > JS usuarios.js > <unknown> > cadastrarUsuarios > usu_tipo
1  const db = require('../database/connection');
2
3  module.exports = {
4    async listarUsuarios(request, response) {
5      try {
6        // instruções SQL
7        const sql = `SELECT
8          usu_id, usu_nome, usu_email, usu_dt_nasc, usu_senha,
9          usu_tipo, usu_ativo
10         FROM usuarios`;
11        // executa instruções SQL e armazena o resultado na variável usuarios
12        const usuarios = await db.query(sql);
13
14        return response.status(200).json({
15          sucesso: true,
16          mensagem: 'Lista de usuários.',
17          dados: usuarios ←
18        });
19      } catch (error) {
20        return response.status(500).json({
21          sucesso: false,
22          mensagem: 'Erro na requisição.',
23          dados: error.message
24        });
25      }
26    },
27    async cadastrarUsuarios(request, response) {
```

Testando no ThunderClient

Para testar, não se esqueça de inicializar o servidor com o comando “npm run dev”.

```
D:\TEMP\Ewerton\intro_node_2sem_23_3des>npm run dev
```

Acesse a collection onde estão as rotas do controller que será testado e execute o método listar.



The screenshot displays the ThunderClient interface. On the left, the 'Collections' panel shows a collection named 'SistemaRestaurante' containing several endpoints. The 'GET listar' endpoint, located under the 'Usuarios' folder, is selected and highlighted with a red arrow. The main panel shows the details of the selected endpoint, including the method 'GET', the URL 'http://localhost:3333/usuarios', and the 'Send' button, which is also highlighted with a red arrow. The 'Response' tab is active, showing the status '200 OK', size '49.26 KB', and time '5 ms'. The response body is a JSON array of user objects, with the first object being:

```
{
  "sucesso": true,
  "mensagem": "Lista de usuários.",
  "dados": [
    {
      "usu_id": 1,
      "usu_nome": "Thomas Francisco Corte Real",
      "usu_email": "thomasfranciscocortereal@kaynak.com.br",
      "usu_dt_nasc": "1990-10-08T03:00:00.000Z",
      "usu_senha": "123456",
      "usu_tipo": 0,
      "usu_ativo": {
        "type": "Buffer",
        "data": [
          1
        ]
      }
    },
    {
      "usu_id": 2,
      "usu_nome": "Mateus Vitor Lima",
      "usu_email": "mateusvitorlima@abcturismo.com.br",
      "usu_dt_nasc": "1998-11-06T02:00:00.000Z",
      "usu_senha": "123456",
      "usu_tipo": 1
    }
  ]
}
```


Nele temos um objeto com atributos chamados “sucesso”, “mensagem” e “dados” este é composto por um array, que por sua vez é composto por dois arrays.

```
1  {
2    "sucesso": true,
3    "mensagem": "Lista de usuários.",
4    "dados": [
5      > [...],
287    [
288      {
289        "_buf": {
290          "type": "Buffer",
291          "data": [
292            1,
293            0,
294            0,
295            1,
296            7,
297            59,
298            0,
299            0,
300            2,
301            3,
302            100,
303            101,
304            102,
305            9,
306            115,
1026        "_clientEncoding": "utf8",
1027        "_catalogLength": 3,
1028        "_catalogStart": 10,
1029        "_schemaLength": 12,
1030        "_schemaStart": 14,
1031        "_tableLength": 8,
1032        "_tableStart": 27,
1033        "_orgTableLength": 8,
1034        "_orgTableStart": 36,
1035        "_orgNameLength": 6,
1036        "_orgNameStart": 52,
1037        "characterSet": 63,
1038        "encoding": "binary",
1039        "name": "usu_id",
1040        "columnLength": 11,
1041        "columnType": 3,
1042        "type": 3,
1043        "flags": 16899,
1044        "decimals": 0
1045      },
1046      {
1047        "_buf": {
1048          "type": "Buffer",
1988        "_clientEncoding": "utf8",
1989        "_catalogLength": 3,
1990        "_catalogStart": 76,
1991        "_schemaLength": 12,
1992        "_schemaStart": 80,
1993        "_tableLength": 8,
1994        "_tableStart": 93,
1995        "_orgTableLength": 8,
1996        "_orgTableStart": 102,
1997        "_orgNameLength": 8,
1998        "_orgNameStart": 120,
1999        "characterSet": 224,
2000        "encoding": "utf8",
2001        "name": "usu_nome",
2002        "columnLength": 240,
2003        "columnType": 253,
2004        "type": 253,
2005        "flags": 4097,
2006        "decimals": 0
13834    ]
13835  }
```

O primeiro traz o resultado gerado pela consulta ao banco de dados e o segundo configurações relacionadas aos dados apresentados.

Nosso objetivo é de mostrar apenas os registros resultantes da consulta, para isso só é necessário apresentar o conteúdo da primeira posição do array.

```
return response.status(200).json({  
    sucesso: true,  
    mensagem: 'Lista de usuários.',  
    dados: usuarios[0]  
});
```

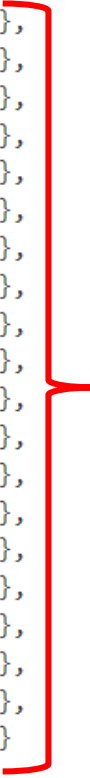


No exemplo abaixo o resultado apresenta alguns objetos dentro de uma array que está contido no atributo dados

```
1  {
2    "sucesso": true,
3    "mensagem": "Lista de usuários.",
4  >  "dados": [...]
286 }
```



```
1  {
2    "sucesso": true,
3    "mensagem": "Lista de usuários.",
4    "dados": [
5  >    { ... },
19 >    { ... },
33 >    { ... },
47 >    { ... },
61 >    { ... },
75 >    { ... },
89 >    { ... },
103 >   { ... },
117 >   { ... },
131 >   { ... },
145 >   { ... },
159 >   { ... },
173 >   { ... },
187 >   { ... },
201 >   { ... },
215 >   { ... },
229 >   { ... },
243 >   { ... },
257 >   { ... },
271 >   { ... }
285 ]
286 }
```



Os objetos representam os registros retornados pela instrução SQL executada no backend.

```
4   "dados": [  
5     {  
6       "usu_id": 1,  
7       "usu_nome": "Thomas Francisco Corte Real",  
8       "usu_email": "thomasfranciscocortereal@kaynak.com.br",  
9       "usu_dt_nasc": "1990-10-08T03:00:00.000Z",  
10      "usu_senha": "123456",  
11      "usu_tipo": 0,  
12      "usu_ativo": {  
13        "type": "Buffer",  
14        "data": [  
15          1  
16        ]  
17      }  
18    },  
19    {  
20      "usu_id": 2,  
21      "usu_nome": "Mateus Vitor Lima"
```



Os objetos são separados por uma “,” após o fechamento da área correspondente ao objeto “{”


Campos do tipo
“bit” são
representados como
“Buffer” de dados e
o valor real do
campo fica dentro
de um objeto em um
array para o atributo
com o nome “data”

```
1  {
2    "sucesso": true,
3    "mensagem": "Lista de usuários.",
4    "dados": [
5      {
6        "usu_id": 1,
7        "usu_nome": "Thomas Francisco Corte Real",
8        "usu_email": "thomasfranciscocortereal@kaynak.com.br",
9        "usu_dt_nasc": "1990-10-08T03:00:00.000Z",
10       "usu_senha": "123456",
11       "usu_tipo": 0,
12       "usu_ativo": {
13         "type": "Buffer",
14         "data": [
15           1
16         ]
17       }
18     },
19     {
20       "usu_id": 2,
21       "usu_nome": "Mateus Vitor Lima",
22       "usu_email": "mateusvitorlima@abcturismo.com.br",
23       "usu_dt_nasc": "1998-11-06T02:00:00.000Z",
24       "usu_senha": "123456",
25       "usu_tipo": 1,
26       "usu_ativo": {
27         "type": "Buffer",
28         "data": [
29           1
30         ]
31       }
32     },
33     { ... }
```


Para melhorar o acesso e a visualização desse tipo de dados insira na instrução “SQL” o comando apontado na imagem abaixo, no lugar do nome do campo que é do tipo “bit”

```
const sql = `SELECT
usu_id, usu_nome, usu_email, usu_dt_nasc, usu_senha,
usu_tipo, usu_ativo = 1 AS usu_ativo
FROM usuarios;`;
```

```
{
  "usu_id": 1,
  "usu_nome": "Thomas Francisco Corte Real",
  "usu_email": "thomasfranciscocortereal@kaynak.com.br",
  "usu_dt_nasc": "1990-10-08T03:00:00.000Z",
  "usu_senha": "123456",
  "usu_tipo": 0,
  "usu_ativo": 1
},
```



```
{
  "usu_id": 5,
  "usu_nome": "Mariah Sebastiana Assunção",
  "usu_email": "mariah_assuncao@queirozgalvao.com",
  "usu_dt_nasc": "1982-12-30T03:00:00.000Z",
  "usu_senha": "123456",
  "usu_tipo": 2,
  "usu_ativo": 0
},
```



Assim as saídas passam a ter o valor 1 para verdadeiro e 0 para falso!

Também é possível identificar a quantidade de registros retornados com a instrução SQL, por meio da propriedade “length”

```
async listarUsuarios(request, response) {  
  try {  
    // instruções SQL  
    const sql = `SELECT  
      usu_id, usu_nome, usu_email, usu_dt_nasc, usu_senha,  
      usu_tipo, usu_ativo = 1 AS usu_ativo  
    FROM usuarios`;  
    // executa instruções SQL e armazena o resultado na variável usuarios  
    const usuarios = await db.query(sql);  
    // armazena em uma variável o número de registros retornados  
    const nItens = usuarios[0].length;  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: 'Lista de usuários.',  
      dados: usuarios[0],  
      nItens  
    });  
  } catch (error) {
```

Saiba mais sobre status http em: <https://www.devmedia.com.br/http-status-code/41222>

Assim temos na saída 4 atributos.

O 1º apresenta uma mensagem de confirmação, o 2º uma mensagem sobre o evento ocorrido, a 3ª traz o resultado da consulta a base de dados e a 4º o nº de itens.

```
1  {  
2    "sucesso": true,  
3    "mensagem": "Lista de usuários.",  
4  > "dados": [...],  
186  "nItens": 20  
187 }
```

Atualizar código:

```
async listarUsuarios(request, response) {  
  try {  
  
    const sql = `  
      SELECT  
        usu_id, usu_nome, usu_email, usu_cpf, usu_dt_nasc, usu_senha,  
        usu_tipo, usu_ativo = 1 AS usu_ativo  
      FROM usuarios;  
    `;  
  
    const [rows] = await db.query(sql);  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: 'Lista de usuários',  
      itens: rows.length,  
      dados: rows  
    });  
  } catch (error) {  
    return response.status(500).json({  
      sucesso: false,  
      mensagem: 'Erro na requisição.',  
      dados: error.message  
    });  
  }  
},
```

Create – Este método será utilizado para realizar o cadastro de usuários do sistema.

```
30   async cadastrarUsuarios(request, response) {
31     try {
32
33       const { nome, email, dt_nasc, senha, tipo, cpf } = request.body;
34       const usu_ativo = 1;
35
36       // instrução SQL
37       const sql = `
38         INSERT INTO usuarios
39         |   (usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo, usu_cpf)
40         VALUES
41         |   (?, ?, ?, ?, ?, ?, ?);
42       `;
43
44       // definição dos dados a serem inseridos em um array
45       const values = [nome, email, dt_nasc, senha, tipo, usu_ativo, cpf];
46
47       // execução da instrução sql passando os parâmetros
48       const [result] = await db.query(sql, values);
49
50       // identificação do ID do registro inserido
51       const dados = {
52         id: result.insertId,
53         nome,
54         email,
55         tipo
56       };
57
```

...

```
58         return response.status(200).json({
59             sucesso: true,
60             mensagem: 'Cadastro de usuários',
61             dados: dados
62         });
63     } catch (error) {
64         return response.status(500).json({
65             sucesso: false,
66             mensagem: 'Erro na requisição.',
67             dados: error.message
68         });
69     }
70 },
```


No INSOMNIA selecione o método “post” entre os métodos de teste dos controllers e altere o corpo da página para JSON, passe os dados do registro a ser inserido como um objeto e teste a inserção.

POST http://localhost:3333/usuarios

Send **200 OK** 49 ms 137 B

Params **Body** Auth Headers **4** Scripts Docs

JSON

```
1 {  
2   "nome": "Janucio Pereira de Souza",  
3   "email": "janu@email.com",  
4   "dt_nasc": "2000-05-18",  
5   "senha": "123",  
6   "tipo": 1,  
7   "cpf": 13212312321  
8 }
```

Preview Headers **8** Cookies Tests **0 / 0**

Preview

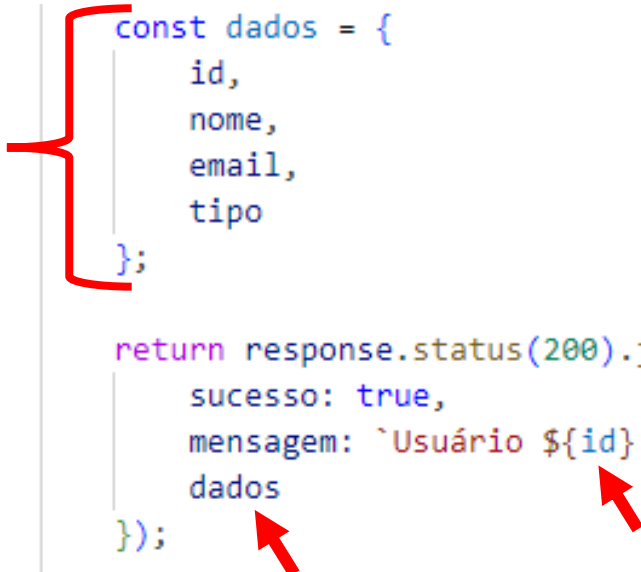
```
1 {  
2   "sucesso": true,  
3   "mensagem": "Cadastro de usuários",  
4   "dados": {  
5     "id": 10,  
6     "nome": "Janucio Pereira de Souza",  
7     "email": "janu@email.com",  
8     "tipo": 1  
9   }  
10 }
```

Update

```
72 async editarUsuarios(request, response) {
73   try {
74     // parâmetros recebidos pelo corpo da requisição
75     const { nome, email, dt_nasc, senha, tipo, ativo } = request.body;
76     // parâmetro recebido pela URL via params ex: /usuario/1
77     const { id } = request.params;
78     // instruções SQL
79     const sql = `
80       UPDATE usuarios SET
81         usu_nome = ?, usu_email = ?, usu_dt_nasc = ?, usu_senha = ?, usu_tipo = ?, usu_ativo = ?
82       WHERE
83         usu_id = ?;
84     `;
85     // preparo do array com dados que serão atualizados
86     const values = [nome, email, dt_nasc, senha, tipo, ativo, id];
87     // execução e obtenção de confirmação da atualização realizada
88     const [result] = await db.query(sql, values);
89
90     if (result.affectedRows === 0) {
91       return response.status(404).json({
92         sucesso: false,
93         mensagem: `Usuário ${id} não encontrado!`,
94         dados: null
95       });
96     }
97   }
```

...

```
98     const dados = {
99         id,
100         nome,
101         email,
102         tipo
103     };
104
105     return response.status(200).json({
106         sucesso: true,
107         mensagem: `Usuário ${id} atualizado com sucesso!`,
108         dados
109     });
110
111 } catch (error) {
112     return response.status(500).json({
113         sucesso: false,
114         mensagem: 'Erro na requisição.',
115         dados: error.message
116     });
117 }
118 },
```



Editar rota

Para editar será necessário passar o id do registro a ser atualizado.

```
7  router.get('/usuarios', UsuariosController.listarUsuarios);  
8  router.post('/usuarios', UsuariosController.cadastrarUsuarios);  
9  router.patch('/usuarios/:id', UsuariosController.editarUsuarios);
```



Visualizar dado a ser atualizado, antes da atualização

GET <http://localhost:3333/usuarios> Send 200 OK 47 ms 2.2 KB


Params Body Auth Headers 3 Scripts Docs

Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

No Body

Preview

```
1 {
2   "sucesso": true,
3   "mensagem": "Lista de usuários",
4   "itens": 11,
5   "dados": [
6     {
7       "usu_id": 1,
8       "usu_nome": "Thomas Francisco Corte Real",
9       "usu_email": "thomasfranciscocortereal@kaynak.com.br",
10      "usu_cpf": 81088713874,
11      "usu_dt_nasc": "1990-10-08T03:00:00.000Z",
12      "usu_senha": "123456",
13      "usu_tipo": 0,
14      "usu_ativo": 1
15    },
16    {
17      "usu_id": 2,
18      "usu_nome": "Mateus Vitor Lima",
19      "usu_email": "mateusvitorlima@abcturismo.com.br",
```



Testar atualização

The screenshot displays a REST client interface with the following components:


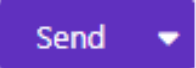
- Request Bar:** Method `PATCH`, URL `http://localhost:3333/usuarios/1`, and a `Send` button.
- Response Bar:** Status `200 OK`, Time `47 ms`, and Size `159 B`.
- Tabs:** `Params`, `Body` (selected), `Auth`, `Headers` (4), `Scripts`, and `Docs`.
- Format:** `JSON` (selected).
- Request Body:**

```
1 {
2   "nome": "Thomas Francisco Corte Real",
3   "email": "thomas@email.com.br",
4   "dt_nasc": "1990-10-08",
5   "senha": "123456",
6   "tipo": 0,
7   "ativo": 1
8 }
```
- Response Preview:**


```
1 {
2   "sucesso": true,
3   "mensagem": "Usuário 1 atualizado com sucesso!",
4   "dados": {
5     "id": "1",
6     "nome": "Thomas Francisco Corte Real",
7     "email": "thomas@email.com.br",
8     "tipo": 0
9   }
10 }
```


Red arrows highlight the URL, the `Send` button, the `Body` tab, the `JSON` format, the `email` field in the request body, and the `email` field in the response body.

Verificar se houve atualização na requisição listar


GET  http://localhost:3333/usuarios  **200 OK** 19 ms 2.2 KB

Params Body Auth Headers **3** Scripts Docs Preview Headers **8** Cookies Tests **0 / 0** → Mock

No Body 

Preview 

```
1 {
2   "sucesso": true,
3   "mensagem": "Lista de usuários",
4   "itens": 11,
5   "dados": [
6     {
7       "usu_id": 1,
8       "usu_nome": "Thomas Francisco Corte Real",
9       "usu_email": "thomas@email.com.br",
10      "usu_cpf": 81088713874,
11      "usu_dt_nasc": "1990-10-08T03:00:00.000Z",
12      "usu_senha": "123456",
13      "usu_tipo": 0,
14      "usu_ativo": 1
15    },
16    {
17      "usu_id": 2,
18      "usu_nome": "Miguel Victor Lima"
```



DELETE


```
async apagarUsuarios(request, response) {
  try {
    // parâmetro passado via url na chamada da api pelo front-end
    const { id } = request.params;
    // comando de exclusão
    const sql = `DELETE FROM usuarios WHERE usu_id = ?`;
    // array com parâmetros da exclusão
    const values = [id];
    // executa instrução no banco de dados
    const [result] = await db.query(sql, values);

    if (result.affectedRows === 0) {
      return response.status(404).json({
        sucesso: false,
        mensagem: `Usuário ${id} não encontrado!`,
        dados: null
      });
    }

    return response.status(200).json({
      sucesso: true,
      mensagem: `Usuário ${id} excluído com sucesso`,
      dados: null
    });
  } catch (error) {
    return response.status(500).json({
      sucesso: false,
      mensagem: 'Erro na requisição.',
      dados: error.message
    });
  }
},
```


Rota

```
router.get('/usuarios', UsuariosController.listarUsuarios);  
router.post('/usuarios', UsuariosController.cadastrarUsuarios);  
router.patch('/usuarios/:id', UsuariosController.editarUsuarios); // params  
router.delete('/usuarios/:id', UsuariosController.apagarUsuarios); // params
```



Verificar usuário a serem excluídos


GET ▼ http://localhost:3333/usuarios Send 200 OK 19 ms 2.2 KB

Params Body Auth Headers 3 Scripts Docs Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

No Body ▼

Preview ▼

```
66 {  
67   "usu_id": 7,  
68   "usu_nome": "Janucio Pereira de Souza",  
69   "usu_email": "janu@email.com",  
70   "usu_cpf": 13212312320,  
71   "usu_dt_nasc": "2000-05-18T03:00:00.000Z",  
72   "usu_senha": "123",  
73   "usu_tipo": 1,  
74   "usu_ativo": 1  
75 },  
76 {  
77   "usu_id": 10,  
78   "usu_nome": "Janucindo Pereira de Souza",  
79   "usu_email": "jacindo@email.com",  
80   "usu_cpf": 13212312321,  
81   "usu_dt_nasc": "2000-05-21T03:00:00.000Z",  
82   "usu_senha": "123",  
83   "usu_tipo": 1,  
84   "usu_ativo": 1  
85 },
```



Teste exclusão

The screenshot displays a REST client interface with a DELETE request to `http://localhost:3333/usuarios/10`. A red arrow points to the URL. The response is a `200 OK` status with a response time of `89 ms` and a body size of `76 B`. The response body is a JSON object: `{ "sucesso": true, "mensagem": "Usuário 10 excluído com sucesso", "dados": null }`. A red arrow points to the `mensagem` field in the JSON response.

DELETE ▼ <code>http://localhost:3333/usuarios/10</code> Send ▼	200 OK 89 ms 76 B
Params Body Auth Headers 3 Scripts Docs	Preview Headers 8 Cookies Tests 0 / 0 → Mock
No Body ▼	Preview ▼
	<pre>1 { 2 "sucesso": true, 3 "mensagem": "Usuário 10 excluído com sucesso", 4 "dados": null 5 }</pre>

Verificar se registro foi excluído


GET ▼ http://localhost:3333/usuarios Send 200 OK 25 ms 2 KB

Params Body Auth Headers 3 Scripts Docs Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

No Body ▼

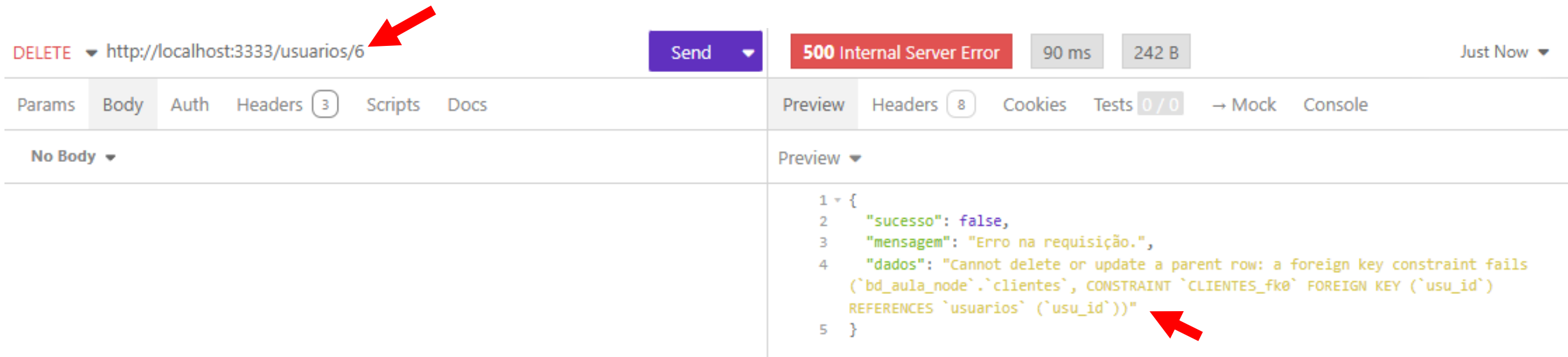
Preview ▼

```
65 },
66 {
67   "usu_id": 7,
68   "usu_nome": "Janucio Pereira de Souza",
69   "usu_email": "janu@email.com",
70   "usu_cpf": 13212312320,
71   "usu_dt_nasc": "2000-05-18T03:00:00.000Z",
72   "usu_senha": "123",
73   "usu_tipo": 1,
74   "usu_ativo": 1
75 },
76 {
77   "usu_id": 11,
78   "usu_nome": "José Manoel",
79   "usu_email": "josema@email.com",
80   "usu_cpf": 45236245845,
81   "usu_dt_nasc": "1990-05-12T03:00:00.000Z",
82   "usu_senha": "123",
83   "usu_tipo": 1,
84   "usu_ativo": 1
85 },
```



Alternativa para exclusão

Ao tentar excluir um registro relacionado a outro o MySQL apresenta um erro. Pois registros relacionados não podem ser excluídos.



The screenshot shows a REST client interface. The top bar displays the method **DELETE** and the URL `http://localhost:3333/usuarios/6`, with a red arrow pointing to the URL. To the right of the URL is a **Send** button. Further right, a red status bar indicates **500 Internal Server Error**, with response time **90 ms** and size **242 B**. The right side of the interface has tabs for **Preview**, **Headers** (8), **Cookies**, **Tests** (0/0), **Mock**, and **Console**. The **Preview** tab is active, showing a JSON response:

```
1 {
2   "sucesso": false,
3   "mensagem": "Erro na requisição.",
4   "dados": "Cannot delete or update a parent row: a foreign key constraint fails
5             (`bd_aula_node`.`clientes`, CONSTRAINT `CLIENTES_fk0` FOREIGN KEY (`usu_id`)
              REFERENCES `usuarios` (`usu_id`))"
6 }
```

A red arrow points to the error message in the JSON response, specifically to the part describing the foreign key constraint failure.

Soluções

Existem duas possibilidades para evitar o erro de exclusão de itens relacionados.

1- Alterar a restrição de chave estrangeira com a instrução “DELETE CASCADE” – Que é uma operação de exclusão em uma tabela referenciada se propaga (cascade = em cascata) para as chaves estrangeiras correspondentes. Ou seja, ao excluir um registro em uma tabela, um registro relacionado em outra tabela é automaticamente excluído. Por exemplo, se uma editora de uma tabela de editoras for excluída, os livros da tabela de livros relacionados com esta editora também serão excluídos automaticamente.

Fonte: <http://www.bosontreinamentos.com.br/bancos-de-dados/restricoes-de-chave-estrangeira-on-delete-cascade-e-outras/#:~:text=ON%20DELETE%20CASCADE%20%E2%80%93%20Uma%20opera%C3%A7%C3%A3o,outra%20tabela%20%C3%A9%20automaticamente%20exclu%C3%ADdo>

Soluções

2- Inserir uma coluna que permita controlar a visualização dos registros, esta coluna seria do tipo BIT.

No exemplo apresentado a seguir, temos na tabela um campo com o nome `usu_ativo` que recebe `true` quando o registro é inserido e o valor `false` quando é excluído.

A partir disso em todas as consultas de usuário deverá ser adicionado um filtro que apresente apenas `usu_ativo = true`, assim o usuário vai ter a impressão de que os registros realmente foram excluídos.

Essa técnica também pode ser útil para recuperar informações excluídas de forma errada.

Crie uma nova função para realizar a exclusão

```
async ocultarUsuario(request, response) {
  try {
    const ativo = false;
    const { id } = request.params;
    const sql = `
      UPDATE usuarios SET
        usu_ativo = ?
      WHERE
        usu_id = ?;
    `;


    const values = [ativo, id];
    const [result] = await db.query(sql, values);

    if (result.affectedRows === 0) {
      return response.status(404).json({
        sucesso: false,
        mensagem: `Usuário ${id} não encontrado!`,
        dados: null
      });
    }

    return response.status(200).json({
      sucesso: true,
      mensagem: `Usuário ${id} excluído com sucesso`,
      dados: null
    });
  } catch (error) {
    return response.status(500).json({
      sucesso: false,
      mensagem: 'Erro na requisição.',
      dados: error.message
    });
  }
},
```


Adicione no arquivo routes.js a rota para chamar a instrução de exclusão.

```
router.get('/usuarios', UsuariosController.listarUsuarios);  
router.post('/usuarios', UsuariosController.cadastrarUsuarios);  
router.patch('/usuarios/:id', UsuariosController.editarUsuarios); // params  
router.delete('/usuarios/:id', UsuariosController.apagarUsuarios); // params  
router.delete('/usuarios/del/:id', UsuariosController.ocultarUsuario); // params
```

A red arrow points from below the text to the newly added route: `router.delete('/usuarios/del/:id', UsuariosController.ocultarUsuario); // params`.

Teste a nova funcionalidade

The screenshot displays a REST client interface with a tab labeled "DEL" and "ocultar" (hidden) with a close button. The main area shows a **DELETE** request to `http://localhost:3333/usuarios/del/7`. The response is a **200 OK** status with a response time of 102 ms and a body size of 75 B. The response body is a JSON object: `{ "sucesso": true, "mensagem": "Usuário 7 excluído com sucesso", "dados": null }`. The interface includes tabs for Params, Body, Auth, Headers (3), Scripts, and Docs. The response preview shows the JSON data with line numbers 1 through 5.

DEL ocultar ✕ +

DELETE ▼ `http://localhost:3333/usuarios/del/7` Send ▼ 200 OK 102 ms 75 B

Params Body Auth Headers (3) Scripts Docs

Preview Headers (8) Cookies Tests 0 / 0 → Mock


No Body ▼

Preview ▼

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Usuário 7 excluído com sucesso",  
4   "dados": null  
5 }
```

Visualizando atualização

```
{  
  "usu_id": 7,  
  "usu_nome": "Janucio Pereira de Souza",  
  "usu_email": "janu@email.com",  
  "usu_cpf": 13212312320,  
  "usu_dt_nasc": "2000-05-18T03:00:00.000Z",  
  "usu_senha": "123",  
  "usu_tipo": 1,  
  "usu_ativo": 0  
},
```



Altere o código SQL da função de listagem para que os itens excluídos não sejam listados.

```
module.exports = {  
  async listarUsuarios(request, response) {  
    try {  
      const sql = `  
        SELECT  
          usu_id, usu_nome, usu_email, usu_cpf, usu_dt_nasc, usu_senha,  
          usu_tipo, usu_ativo = 1 AS usu_ativo  
        FROM usuarios  
        WHERE usu_ativo = 1;`  
      ;  
    }  
  }  
}
```

Teste uma exclusão e veja se o item realmente foi excluído.



Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

[Introduction to Insomnia](#)

GET ▼ http://localhost:3333/usuarios

Send ▼

200 OK

27 ms

1679 B

Params

Body

Auth

Headers 3

Scripts

Docs

Preview

Headers 8

Cookies

Tests 0 / 0

→ Mock

Console

No Body ▼Preview ▼

Sugestão para estudo

- <https://victorhuguw-64.medium.com/construindo-uma-rest-api-utilizando-nodejs-express-e-mysql-parte-1-ef25643ab41b>