

# 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) {
```

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

# 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 application interface. On the left, a sidebar shows a collection named 'SistemaRestaurante' containing a sub-collection 'Usuarios'. The 'listar' method (GET) is selected, with a red arrow pointing to it. The main panel shows the details of the 'listar' method, including the URL 'http://localhost:3333/usuarios' and a 'Send' button, with a red arrow pointing to it. The right panel shows the response status '200 OK' and the response body, which is a JSON array of user objects. A red arrow points to the 'listar' method in the sidebar.

THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

JS usuarios.js TC listar

New Request

Activity Collections Env

filter collections

SistemaRestaurante

Usuarios

GET listar 33 mins ago

POST cadastrar 33 mins ago

PATCH editar 33 mins ago

DEL apagar 33 mins ago

Produtalagradientes

GET http://localhost:3333/usuarios

Send

Query Headers 2 Auth Body

Query Parameters

parameter

Status: 200 OK Size: 49.26 KB Time: 5 ms

Response Headers 7 Cookies Results Docs

```
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    }
27  ]
28 }
```


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]  
});
```

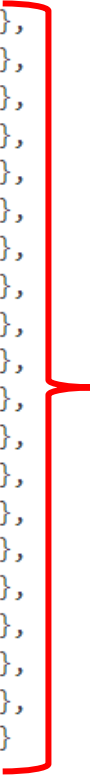


No exemplo abaixo o resultado apresenta alguns objetos dentro de uma array que está contido no atributo dados

```
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 }
```

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

```
async cadastrarUsuarios(request, response) {  
  try {  
    // parâmetros recebidos no corpo da requisição  
    const { usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo } = request.body;  
    // instrução SQL  
    const sql = `INSERT INTO usuarios  
      (usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo)  
      VALUES (?, ?, ?, ?, ?, ?)`;  
    // definição dos dados a serem inseridos em um array  
    const values = [usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo];  
    // execução da instrução sql passando os parâmetros  
    const execSql = await db.query(sql, values);  
    // identificação do ID do registro inserido  
    const usu_id = execSql[0].insertId;  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: 'Cadastro de usuário efetuado com sucesso.',  
      dados: usu_id,   
      //mensSql: execSql  
    });  
  } catch (error) {  
    return response.status(500).json({  
      sucesso: false,  
      mensagem: 'Erro na requisição.',  
      dados: error.message  
    });  
  }  
},  
async editarUsuarios(request, response) {
```

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

The screenshot displays the ThunderClient application interface. On the left sidebar, under the 'Collections' tab, a collection named 'SistemaRestaurante' is expanded, showing a list of requests. The 'POST' request named 'cadastrar' is highlighted, with a red arrow pointing to it. The main panel shows the details of this request. The method is 'POST' and the URL is 'http://localhost:3333/usuarios'. The 'Body' tab is selected, and the content is in 'JSON' format. The JSON content is a valid object with the following fields: 'usu\_nome', 'usu\_email', 'usu\_dt\_nasc', 'usu\_senha', 'usu\_tipo', and 'usu\_ativo'. A red bracket highlights the entire JSON object. The 'Send' button is also highlighted with a red arrow. On the right, the 'Response' tab shows the server's reply with a status of '200 OK', a size of '83 Bytes', and a time of '73 ms'. The response body is a JSON object with 'sucesso' set to true, a message in Portuguese, and a 'dados' field with the value 25. A red arrow points to the 'dados' field in the response.

THUNDER CLIENT

File Edit Selection View Go Run ...

api\_nodejs\_2des\_a\_2sem\_24

JS usuarios.js TC cadastrar JS produtos.js JS routes.js

New Request

Activity Collections Env

filter collections

SistemaRestaurante

Usuarios

GET listar 15 days ago

POST cadastrar just now

PATCH editar 6 days ago

DEL apagar 23 hours ago

POST http://localhost:3333/usuarios Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

```
1 {
2   "usu_nome": "Jorisvaldo Jonas",
3   "usu_email": "jj@email.com",
4   "usu_dt_nasc": "1978-04-10",
5   "usu_senha": "123@321",
6   "usu_tipo": 2,
7   "usu_ativo": true
8 }
```

Status: 200 OK Size: 83 Bytes Time: 73 ms

Response Headers 7 Cookies Results Docs

```
1 {
2   "sucesso": true,
3   "mensagem": "Cadastro de usuário efetuado com sucesso.",
4   "dados": 25
5 }
```

# Update

```
async editarUsuarios(request, response) {
  try {
    // parâmetros recebidos pelo corpo da requisição
    const { usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo } = request.body;
    // parâmetro recebido pela URL via params ex: /usuario/1
    const { usu_id } = request.params;
    // instruções SQL
    const sql = `UPDATE usuarios SET usu_nome = ?, usu_email = ?,
      usu_dt_nasc = ?, usu_senha = ?, usu_tipo = ?,
      usu_ativo = ? WHERE usu_id = ?`;
    // preparo do array com dados que serão atualizados
    const values = [usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo, usu_id];
    // execução e obtenção de confirmação da atualização realizada
    const atualizaDados = await db.query(sql, values);


    return response.status(200).json({
      sucesso: true,
      mensagem: `Usuário ${usu_id} atualizado com sucesso!`,
      dados: atualizaDados[0].affectedRows
      // mensSql: atualizaDados
    });
  } catch (error) {
    return response.status(500).json({
      sucesso: false,
      mensagem: 'Erro na requisição.',
      dados: error.message
    });
  }
},
async apagarUsuarios(request, response) {
```



## Editar rota

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

```
53
54  router.get('/usuarios', UsuariosController.listarUsuarios);
55  router.post('/usuarios', UsuariosController.cadastrarUsuarios);
56  router.patch('/usuarios/:usu_id', UsuariosController.editarUsuarios);
57
```



# Visualizar dado a ser atualizado, antes da atualização

GET ⌵ http://localhost:3333/usuarios Send

Query

Headers <sup>2</sup>

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK   Size: 3.58 KB   Time: 51 ms

Response

Headers <sup>7</sup>

Cookies

Results

Docs

{ }

≡

```
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": 1
13     },
14   ]
15 }
```

# Testar atualização

The screenshot displays a REST client interface with a PATCH request to `http://localhost:3333/usuarios/1`. The request body is a JSON object with user details. The response is a 200 OK status with a JSON body indicating success.

**Request:**

- Method: PATCH
- URL: `http://localhost:3333/usuarios/1`
- Body (JSON):

```
1 {
2   "usu_id": 1,
3   "usu_nome": "Thomas Francisco Corte Real",
4   "usu_email": "thomasfranciscocortereal@kaynak.com.br",
5   "usu_dt_nasc": "1990-10-08",
6   "usu_senha": "543@21",
7   "usu_tipo": 0,
8   "usu_ativo": 1
9 }
```

**Response:**

- Status: 200 OK
- Size: 74 Bytes
- Time: 56 ms
- Body (JSON):

```
1 {
2   "sucesso": true,
3   "mensagem": "Usuário 1 atualizado com sucesso!",
4   "dados": 1
5 }
```

# Verificar se houve atualização na requisição listar

GET ⌵ http://localhost:3333/usuarios Send

Query Headers <sup>2</sup> Auth Body Tests Pre Run

Query Parameters

☐ parameter

value

Status: 200 OK Size: 3.58 KB Time: 50 ms

Response Headers <sup>7</sup> Cookies Results Docs

{}

≡


```
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": "543@21",
11       "usu_tipo": 0,
12       "usu_ativo": 1
13     },
14   ]
}
```

# Apagar

```
async apagarUsuarios(request, response) {  
  try {  
    // parâmetro passado via url na chamada da api pelo front-end  
    const { usu_id } = request.params;  
    // comando de exclusão  
    const sql = `DELETE FROM usuarios WHERE usu_id = ?`;   
    // array com parâmetros da exclusão  
    const values = [usu_id];  
    // executa instrução no banco de dados  
    const excluir = await db.query(sql, values);  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: `Usuário ${usu_id} excluído com sucesso`,  
      dados: excluir[0].affectedRows  
    });  
  } catch (error) {  
    return response.status(500).json({  
      sucesso: false,  
      mensagem: 'Erro na requisição.',  
      dados: error.message  
    });  
  }  
},
```

# Rota

```
53  router.get('/usuarios', UsuariosController.listarUsuarios);
54  router.post('/usuarios', UsuariosController.cadastrarUsuarios);
55  router.patch('/usuarios/:usu_id', UsuariosController.editarUsuarios); // params(link) e body
56  router.delete('/usuarios/:usu_id', UsuariosController.apagarUsuarios); // params(link) e body
```



# Verificar usuário a serem excluídos

GET ⌵ http://localhost:3333/usuarios Send

Query

Headers <sup>2</sup>

Auth

Body

Tests

Pre Run

Query Parameters

☐ parameter

value

Status: 200 OK Size: 3.58 KB Time: 38 ms

Response

Headers <sup>7</sup>

Cookies

Results

Docs

```
124  "usu_nome": "David Vanjao",
125  "usu_email": "david@vivo.com.br",
126  "usu_dt_nasc": "1994-07-01T03:00:00.000Z",
127  "usu_senha": "123456",
128  "usu_tipo": 2,
129  "usu_ativo": 1
130  },
131  {
132    "usu_id": 15,
133    "usu_nome": "Alberto Roberto",
134    "usu_email": "albrob@email.com",
135    "usu_dt_nasc": "2001-04-10T03:00:00.000Z",
136    "usu_senha": "123",
137    "usu_tipo": 2,
138    "usu_ativo": 1
139  },
140  {
141    "usu_id": 16,
142    "usu_nome": "Felipe Henrique",
143    "usu_email": "feemlk-top@hotmail.com",
144    "usu_dt_nasc": "2000-09-09T03:00:00.000Z",
145    "usu_senha": "1234",
146    "usu_tipo": 2,
147    "usu_ativo": 1
148  },
149  {
150    "usu_id": 17
```

# Teste exclusão

The screenshot displays the Thunder Client interface with a DELETE request configured and executed. The request is to `http://localhost:3333/usuarios/15`. The response is a 200 OK status with a JSON body indicating successful deletion of user 15. Red arrows highlight the 'apagar' endpoint in the sidebar, the request URL, the 'Send' button, and the success message in the response body.

**THUNDER CLIENT**

**JS** usuarios.js **TC** apagar **TC** listar

**New Request**

**Activity** **Collections** **Env**

filter collections

▼ SistemaRestaurante

- 📁 Usuarios
  - GET** listar just now
  - POST** cadastrar just now
  - PATCH** editar just now
  - DEL** apagar just now

**DELETE** `http://localhost:3333/usuarios/15` **Send**

**Query** **Headers** <sup>2</sup> **Auth** **Body** **Tests** **Pre Run**

Query Parameters

<input type="checkbox"/>	parameter	value
--------------------------	-----------	-------

**Status:** 200 OK **Size:** 73 Bytes **Time:** 74 ms

**Response** **Headers** <sup>7</sup> **Cookies** **Results** **Docs**

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Usuário 15 excluído com sucesso",  
4   "dados": 1  
5 }
```



# Verificar se registro foi excluído

THUNDER CLIENT

New Request

Activity Collections Env

filter collections

SystemaRestaurante

- Usuarios
  - GET listar 5 mins ago
  - POST cadastrar just now
  - PATCH editar just now
  - DEL apagar just now
- ProdutoIngredientes
- ProdutoTipos
- Pedidos
- PedidoProdutos
- Mesas

JS usuarios.js TC apagar TC listar

GET http://localhost:3333/usuarios Send

Query Headers 2 Auth Body Tests Pre Run

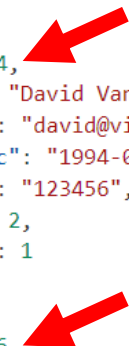
Query Parameters

parameter	value
-----------	-------

Status: 200 OK Size: 3.42 KB Time: 36 ms

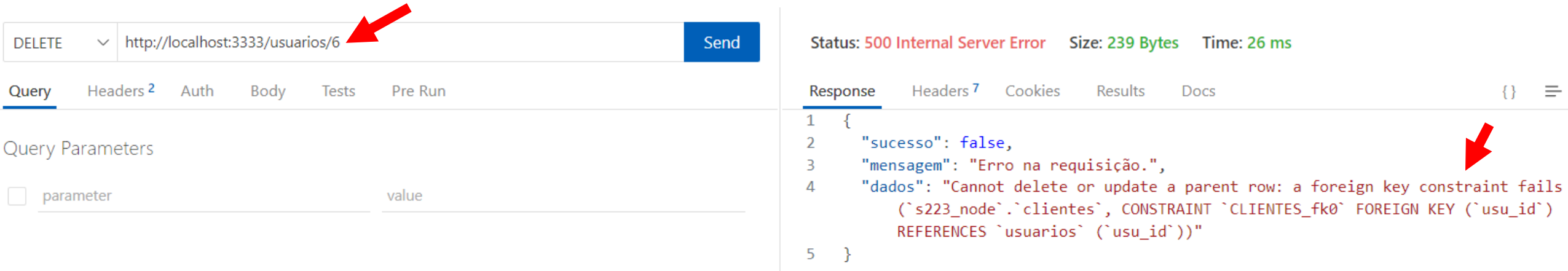
Response Headers 7 Cookies Results Docs

```
117 "usu_dt_nasc": "2001-05-10T03:00:00.000Z",
118 "usu_senha": "123",
119 "usu_tipo": 2,
120 "usu_ativo": 1
121 },
122 {
123   "usu_id": 14,
124   "usu_nome": "David Vanjao",
125   "usu_email": "david@vivo.com.br",
126   "usu_dt_nasc": "1994-07-01T03:00:00.000Z",
127   "usu_senha": "123456",
128   "usu_tipo": 2,
129   "usu_ativo": 1
130 },
131 {
132   "usu_id": 16,
133   "usu_nome": "Felipe Henrique",
134   "usu_email": "feemlk-top@hotmail.com",
135   "usu_dt_nasc": "2000-09-09T03:00:00.000Z",
136   "usu_senha": "1234",
137   "usu_tipo": 2,
138   "usu_ativo": 1
139 },
140 {
141   "usu_id": 17,
142   "usu_nome": "Kian Savuri".
```



# 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 displays a REST client interface. On the left, a DELETE request is configured for the URL `http://localhost:3333/usuarios/6`. A red arrow points to the URL field. Below the URL bar, the 'Query Parameters' section is visible, showing a table with columns 'parameter' and 'value'. On the right, the 'Response' tab is active, showing a 500 Internal Server Error. The response body is a JSON object with the following structure:

```
{
  "sucesso": false,
  "mensagem": "Erro na requisição.",
  "dados": "Cannot delete or update a parent row: a foreign key constraint fails\n(`s223_node`.`clientes`, CONSTRAINT `CLIENTES_fk0` FOREIGN KEY (`usu_id`)\nREFERENCES `usuarios` (`usu_id`))"
}
```

A red arrow points to the error message in the 'dados' field of the response.

# 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

```
111 },
112 async ocultarUsuario(request, response) {
113   try {
114     const usu_ativo = false;
115     const { usu_id } = request.params;
116     const sql = `UPDATE usuarios SET usu_ativo = ?
117       WHERE usu_id = ?`;
118     const values = [usu_ativo, usu_id];
119     const atualizacao = await db.query(sql, values);
120
121     return response.status(200).json({
122       sucesso: true,
123       mensagem: `Usuário ${usu_id} excluído com sucesso`,
124       dados: atualizacao[0].affectedRows
125     });
126   } catch (error) {
127     return response.status(500).json({
128       sucesso: false,
129       mensagem: 'Erro na requisição.',
130       dados: error.message
131     });
132   }
133 },
134 }
135
```

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

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



# Teste a nova funcionalidade

The screenshot displays the Thunder Client interface. The top bar shows several tabs: 'JS usuarios.js', 'TC ocultar', 'JS routes.js', 'TC apagar', and 'TC listar'. The 'JS routes.js' tab is active, and a red arrow points to it. Below the tabs, the request details are shown: a DELETE request to 'http://localhost:3333/usuarios/del/6'. A red arrow points to the URL. The 'Send' button is visible. The left sidebar shows a collection named 'SistemaRestaurante' with a sub-collection 'Usuarios'. The 'ocultar' endpoint is highlighted with a red arrow. The right panel shows the response: 'Status: 200 OK', 'Size: 72 Bytes', and 'Time: 41 ms'. The response body is a JSON object: { 'sucesso': true, 'mensagem': 'Usuário 6 excluído com sucesso', 'dados': 1 }. A red arrow points to the 'mensagem' field.

THUNDER CLIENT

New Request

Activity Collections Env

filter collections

SistemaRestaurante

Usuarios

GET listar 13 mins ago

POST cadastrar 13 mins ago

PATCH editar 13 mins ago

DEL apagar 6 mins ago

DEL ocultar just now

JS usuarios.js TC ocultar JS routes.js TC apagar TC listar

DELETE http://localhost:3333/usuarios/del/6 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 72 Bytes Time: 41 ms

Response Headers 7 Cookies Results Docs

```
1 {
2   "sucesso": true,
3   "mensagem": "Usuário 6 excluído com sucesso",
4   "dados": 1
5 }
```

# Visualizando atualização

GET ⌵ http://localhost:3333/usuarios Send

Query

Headers <sup>2</sup>

Auth

Body

Tests

Pre Run

Query Parameters

<input type="checkbox"/>	parameter	value
--------------------------	-----------	-------

Status: 200 OK Size: 3.42 KB Time: 42 ms

Response Headers <sup>7</sup> Cookies Results Docs

```
45  "usu_dt_nasc": "1982-12-30T03:00:00.000Z",
46  "usu_senha": "123456",
47  "usu_tipo": 2,
48  "usu_ativo": 0
49  },
50  {
51    "usu_id": 6,
52    "usu_nome": "Rosângela Marina Nicole Aragão",
53    "usu_email": "rosangela_marina_aragao@vivo.com.br",
54    "usu_dt_nasc": "1970-10-22T03:00:00.000Z",
55    "usu_senha": "123456",
56    "usu_tipo": 2,
57    "usu_ativo": 0
58  },
59  {
60    "usu_id": 7,
61    "usu_nome": "Alberto Roberto"
```





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

```
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  
    WHERE usu_ativo = 1;`;  
    // executa instruções SQL e armazena o resultado na variável usuários
```

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

THUNDER CLIENT

JS usuarios.js TC ocultar JS routes.js TC apagar TC listar

New Request

Activity Collections Env

filter collections

SistemaRestaurante

Usuarios

GET listar 20 mins ago

POST cadastrar 13 mins ago

PATCH editar 13 mins ago

DEL apagar 6 mins ago

DEL ocultar just now

ProdutoIngredientes

GET http://localhost:3333/usuarios Send

Query Headers 2 Auth Body Tests Pre Run

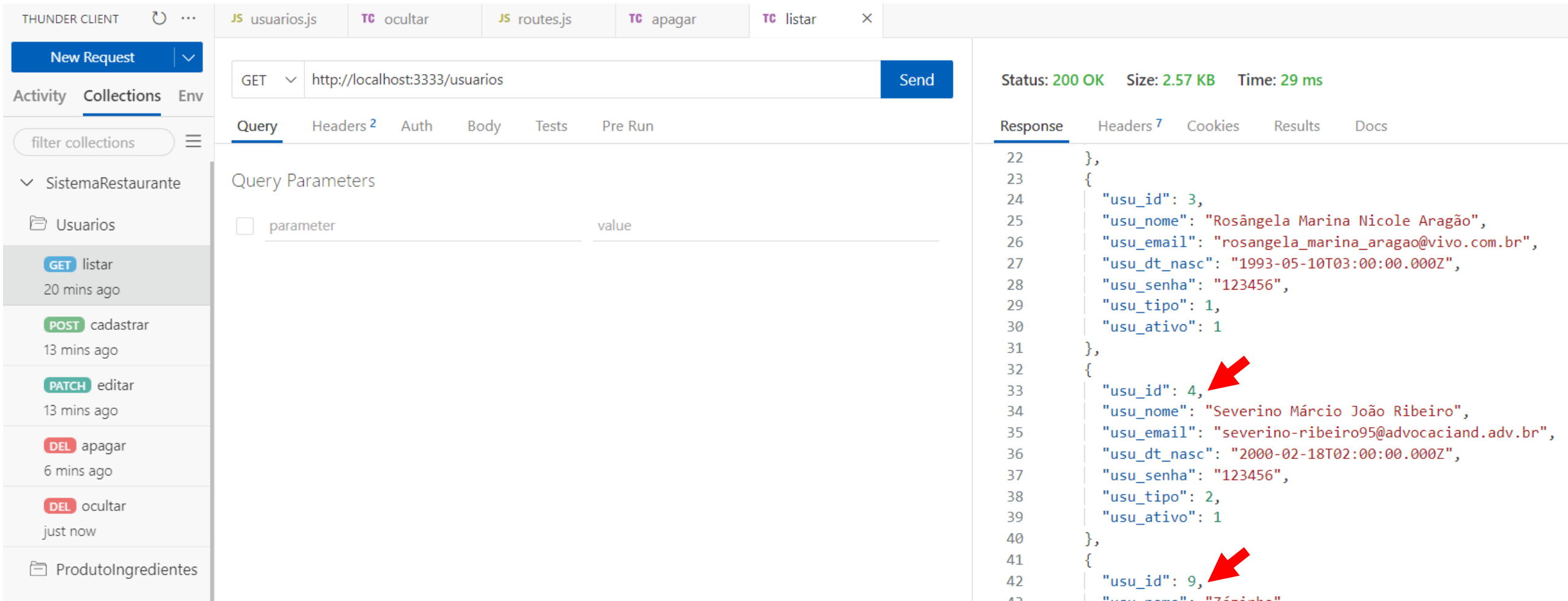
Query Parameters

parameter value

Status: 200 OK Size: 2.57 KB Time: 29 ms

Response Headers 7 Cookies Results Docs

```
22 },
23 {
24   "usu_id": 3,
25   "usu_nome": "Rosângela Marina Nicole Aragão",
26   "usu_email": "rosangela_marina_aragao@vivo.com.br",
27   "usu_dt_nasc": "1993-05-10T03:00:00.000Z",
28   "usu_senha": "123456",
29   "usu_tipo": 1,
30   "usu_ativo": 1
31 },
32 {
33   "usu_id": 4,
34   "usu_nome": "Severino Márcio João Ribeiro",
35   "usu_email": "severino-ribeiro95@advocaciand.adv.br",
36   "usu_dt_nasc": "2000-02-18T02:00:00.000Z",
37   "usu_senha": "123456",
38   "usu_tipo": 2,
39   "usu_ativo": 1
40 },
41 {
42   "usu_id": 9,
43   "usu_nome": "Zézinha"
```



# Sugestão para estudo

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