

= CRUD MySQL =

Profa. Me. Patrícia Raia Nogueira Cavoto
patricia.nogueira@puc-campinas.edu.br



CRUD MySQL

- Vamos criar uma aplicação que utiliza um serviço contendo as operações básicas de manipulação de dados: CRUD (*create, read, update, delete*).
- Os dados serão armazenados em um banco de dados MySQL.



CRUD MySQL

Passos para implementação:

Banco de Dados:

- Criar banco de dados.

Programação:

- Back-End:* serviços CRUD;
- Front-End:* listagem e manipulação dados.



CRUD MySQL

Passos para implementação:

Banco de Dados:

- Criar banco de dados.

Programação:

- Back-End*: serviços CRUD;
- Front-End*: listagem e manipulação dados.



CRUD MySQL: Criar Banco de Dados

- Utilizar o MySQL Workbench para conectar no servidor e acessar o seu banco de dados:

<https://dev.mysql.com/downloads/workbench/>

CRUD MySQL: Criar Banco de Dados

- Na tela inicial, clique no ícone +



Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

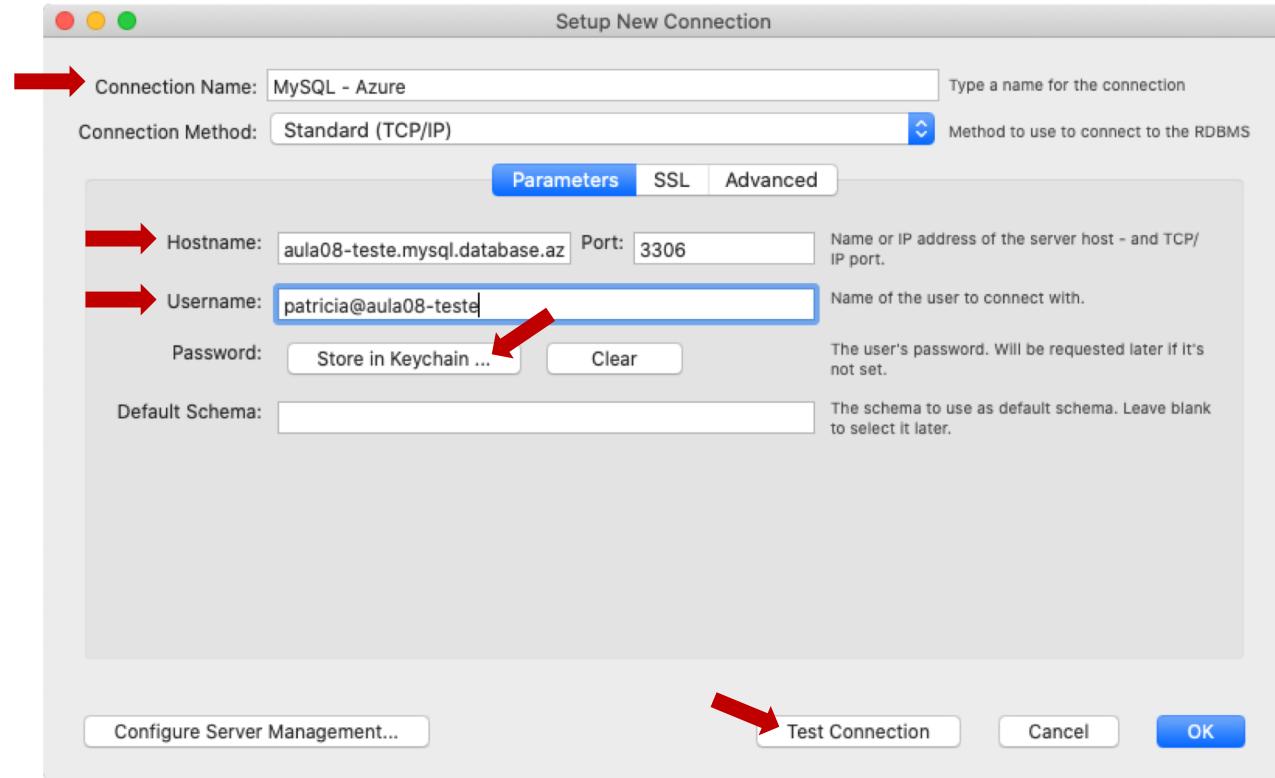
[Browse Documentation >](#)

[Read the Blog >](#)

[Discuss on the Forums >](#)

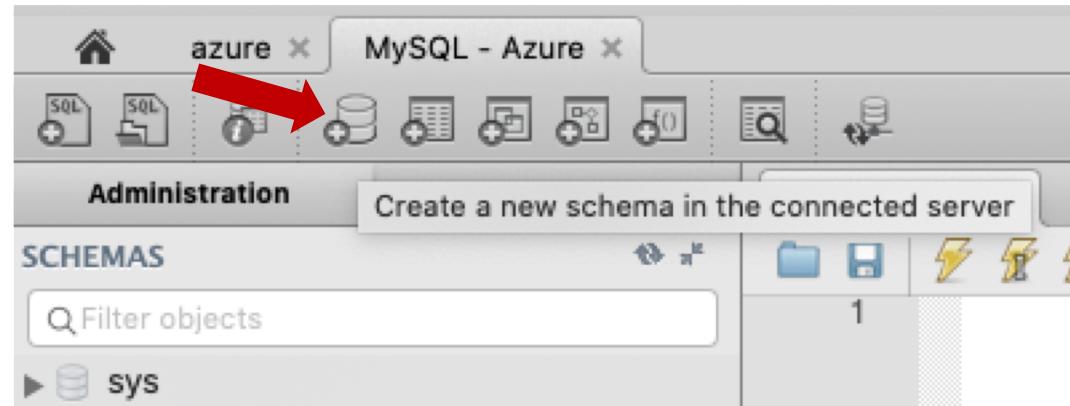
CRUD MySQL: Criar Banco de Dados

- Preencha:
 - *Connection Name*: nome para identificação;
 - *Hostname* e *Username*: conforme obtido das credenciais.
- Clique em ***Store in Keychain*** para digitar a senha e armazená-la.
- Clique em ***Test Connection*** para testar os dados de acesso e depois em ***OK***.



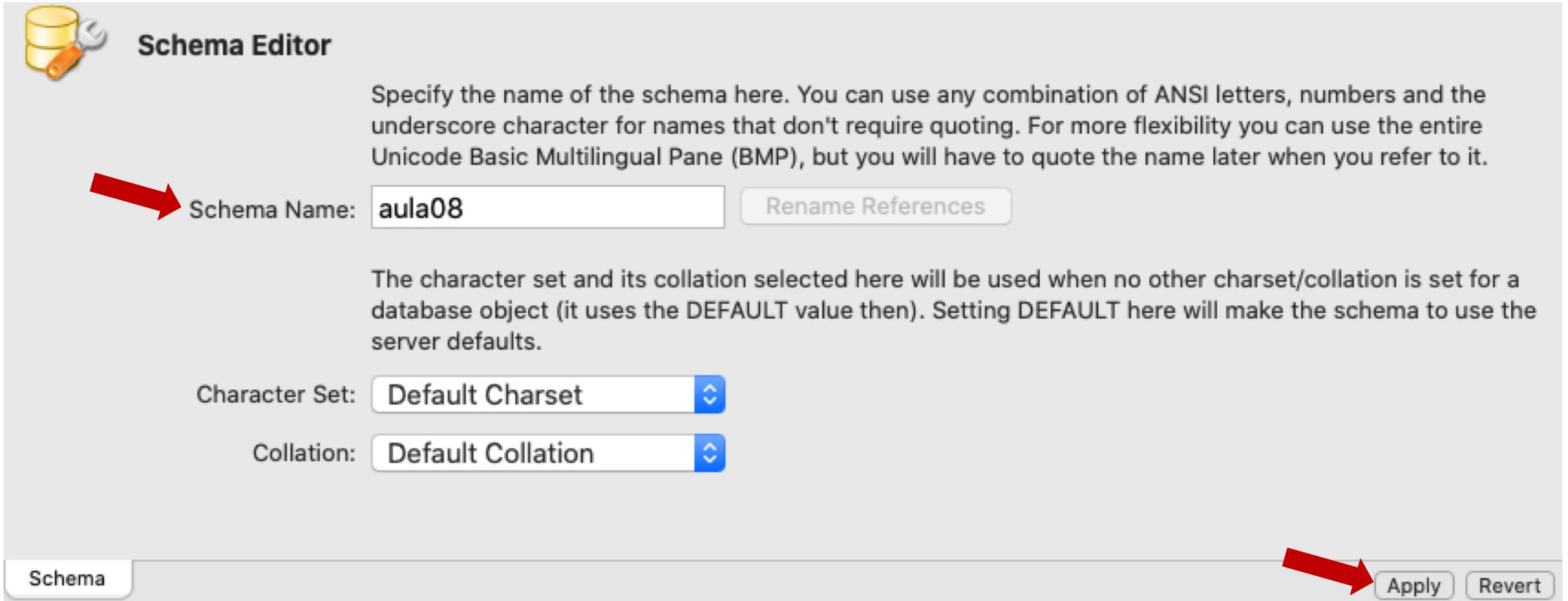
CRUD MySQL: Criar Banco de Dados

- Clique duas vezes na conexão criada para acesso o Banco de Dados.
- Clique no botão ***Create New Schema***.



CRUD MySQL: Criar Banco de Dados

- Informe o *Schema Name* e clique em *Apply*.



The screenshot shows the MySQL Schema Editor interface. At the top left is a database icon with a wrench. To its right is the title "Schema Editor". Below the title is a descriptive text: "Specify the name of the schema here. You can use any combination of ANSI letters, numbers and the underscore character for names that don't require quoting. For more flexibility you can use the entire Unicode Basic Multilingual Plane (BMP), but you will have to quote the name later when you refer to it." A red arrow points to the "Schema Name:" input field, which contains the value "aula08". To the right of the input field is a "Rename References" button. Below this section is another descriptive text: "The character set and its collation selected here will be used when no other charset/collation is set for a database object (it uses the DEFAULT value then). Setting DEFAULT here will make the schema to use the server defaults." Underneath this text are two dropdown menus: "Character Set: Default Charset" and "Collation: Default Collation". At the bottom of the editor, there is a navigation bar with tabs: "Schema" (which is selected and highlighted in white), "Table", "View", and "Structure". To the right of the tabs are two buttons: "Apply" and "Revert", with a red arrow pointing to the "Apply" button.

CRUD MySQL: Criar Banco de Dados

- Você deverá utilizar o script: 08_database.sql para criar a tabela cliente contendo os atributos:
 - id
 - nome
 - endereço
 - email
 - telefone.
- Este script também insere alguns registros nesta tabela.



CRUD MySQL: Criar Banco de Dados

- Pronto, agora que já temos nosso Banco de Dados configurado e com dados, podemos passar para a programação e integração com o node.

CRUD MySQL

Passos para implementação:

✓ **Banco de Dados:**

- ✓ Criar banco de dados.

□ **Programação:**

- *Back-End:* serviços CRUD;
- *Front-End:* listagem e manipulação dados.



CRUD MySQL: *Back-End*

- Utilizando `express` crie um projeto para fazer essa integração.
- Serão necessários dois pacotes adicionais no node para facilitar a conexão com o banco de dados MySQL:
 - `mysql`
 - `express-myconnection`



CRUD MySQL: *Back-End - Package mysql*

- O pacote `mysql` disponibiliza os drivers de conexão com o banco de dados MySQL.
- Na linha de comando, acesse a pasta do seu projeto e digite:

```
npm install mysql
```

CRUD MySQL: *Back-End - Package express-myconnection*

- O pacote `express-myconnection` que facilita o gerenciamento das conexões com o banco de dados, além de trabalhar de forma integrada com a package `mysql`.
- Ainda na pasta do seu projeto e digite:

```
npm install express-myconnection
```

CRUD MySQL: Back-End - Conexão MySQL

- Acessar o arquivo app.js e criar a conexão mysql (incluir o código abaixo após a linha 23 no arquivo):

```
var connection = require('express-myconnection');
var mysql = require('mysql');
app.use(
  connection(mysql,{
    host: '', //servidor do banco mysql (server name do serviço)
    user: '', //usuario (server adm login name do serviço)
    password: '', //senha do servidor configurada no serviço
    port : 3306, //porta do mysql, normalmente 3306
    database:'' //nome da base de dados (esquema criado)
  },'pool')
);
```

CRUD MySQL: *Back-End* - Rota Cliente

- Vamos agora criar a rota cliente, que irá conter todas as operações necessárias à nossa tabela cliente (serviços).
- Ainda no arquivo app.js logo **após** as linhas de código de criação da conexão mysql, inserir:

```
/*-----  
acesso ao serviço de clientes  
-----*/  
var cliente = require('./routes/cliente');  
app.use('/cliente', cliente);  
//-----
```

CRUD MySQL: *Back-End* - Rota Cliente

- Dentro da pasta `routes`, vamos criar o arquivo: `cliente.js`.
- Vamos adicionar o pacote `express` e o `router` padrão do `express` que contém todas as rotas exportadas.

```
var express = require('express');
var router = express.Router();
```

- Implementaremos os seguintes serviços nessa rota: `lista`, `listaCliente`, `deleta`, `insere` e `altera`.

CRUD MySQL: Back-End - Rota Cliente: lista

```
4 // lista todos os clientes
5 router.get('/lista', function (req, res, next) {
6   // realiza conexão com o Banco de Dados
7   req.getConnection(function (err, connection) {
8     if (err)
9       res.json({ status: 'ERRO', data: err.toString() });
10
11   // realiza consulta no bd
12   connection.query("SELECT * FROM cliente", function (err, rows) {
13     if (err)
14       res.json({ status: 'ERRO', data: err.toString() });
15     else
16       res.json({ status: 'OK', data: rows });
17   });
18 });
19});
```

- Vamos criar o serviço `lista` para recuperar os dados de todos os clientes. Este serviço irá retornar um json com todos os clientes cadastrados:

CRUD MySQL: Back-End - Rota Cliente: lista

```
4 // lista todos os clientes
5 router.get('/lista', function (req, res, next) {
6   // realiza conexão com o Banco de Dados
7   req.getConnection(function (err, connection) {
8     if (err)
9       res.json({ status: 'ERRO', data: err.toString() });
10
11   // realiza consulta no bd
12   connection.query("SELECT * FROM cliente", function (err, rows) {
13     if (err)
14       res.json({ status: 'ERRO', data: err.toString() });
15     else
16       res.json({ status: 'OK', data: rows });
17   });
18 });
19});
```

- O serviço é acessado por um **get**, já que provê informações ao *front-end*.

CRUD MySQL: Back-End - Rota Cliente: lista

```
4 // lista todos os clientes
5 router.get('/lista', function (req, res, next) {
6   // realiza conexão com o Banco de Dados
7   req.getConnection(function (err, connection) {
8     if (err)
9       res.json({ status: 'ERRO', data: err.toString() });
10
11    // realiza consulta no bd
12    connection.query("SELECT * FROM cliente", function (err, rows) {
13      if (err)
14        res.json({ status: 'ERRO', data: err.toString() });
15      else
16        res.json({ status: 'OK', data: rows });
17    });
18  });
19});
```

- `getConnection` faz a conexão com o banco de dados utilizando as configuração que definimos no `app.js`. Caso ocorra algum erro na conexão com o banco, esse erro é retornado.

CRUD MySQL: Back-End - Rota Cliente: lista

```
4 // lista todos os clientes
5 router.get('/lista', function (req, res, next) {
6   // realiza conexão com o Banco de Dados
7   req.getConnection(function (err, connection) {
8     if (err)
9       res.json({ status: 'ERRO', data: err.toString() });
10
11    // realiza consulta no bd
12    connection.query("SELECT * FROM cliente", function (err, rows) {
13      if (err)
14        res.json({ status: 'ERRO', data: err.toString() });
15      else
16        res.json({ status: 'OK', data: rows });
17    });
18  });
19});
```

- Caso a conexão seja bem sucedida utilizamos a função `query` para fazer a consulta no banco de dados e retornar as informações necessárias. O primeiro parâmetro é a query SQL e o segundo a função de *call-back* para tratamento do erro e do retorno, que é armazenado em `ROWS`.

CRUD MySQL: Back-End - Rota Cliente: listaCliente

```
22 // lista os dados de um cliente
23 router.get('/listaCliente', function (req, res, next) {
24     // recupera id do cliente que deverá ser consultado
25     var id = req.query.id;
26     req.getConnection(function (err, connection) {
27         if (err)
28             res.json({ status: 'ERRO', data: err.toString() });
29
30         connection.query("SELECT * FROM cliente WHERE id = " + id, function (err, rows) {
31             if (err)
32                 res.json({ status: 'ERRO', data: err.toString() });
33             else
34                 res.json({ status: 'OK', data: rows });
35         });
36     });
37 });


```

- O próximo serviço é o listaCliente que permite recuperar os dados de um único cliente. Este serviço irá retornar um json com os dado do cliente solicitado.

CRUD MySQL: Back-End - Rota Cliente: listaCliente

```
22 // lista os dados de um cliente
23 router.get('/listaCliente', function (req, res, next) {
24     // recupera id do cliente que deverá ser consultado
25     var id = req.query.id;
26     req.getConnection(function (err, connection) {
27         if (err)
28             res.json({ status: 'ERRO', data: err.toString() });
29
30         connection.query("SELECT * FROM cliente WHERE id = " + id, function (err, rows) {
31             if (err)
32                 res.json({ status: 'ERRO', data: err.toString() });
33             else
34                 res.json({ status: 'OK', data: rows });
35         });
36     });
37 });


```

- Já que iremos retornar os dados de um único cliente, precisamos identificar qual é o cliente. O primeiro passo então é recuperar o id do cliente desejado (como já vimos nas aulas passadas).

CRUD MySQL: Back-End - Rota Cliente: listaCliente

```
22 // lista os dados de um cliente
23 router.get('/listaCliente', function (req, res, next) {
24     // recupera id do cliente que deverá ser consultado
25     var id = req.query.id;
26     req.getConnection(function (err, connection) {
27         if (err)
28             res.json({ status: 'ERRO', data: err.toString() });
29
30         connection.query("SELECT * FROM cliente WHERE id = " + id, function (err, rows) {
31             if (err)
32                 res.json({ status: 'ERRO', data: err.toString() });
33             else
34                 res.json({ status: 'OK', data: rows });
35         });
36     });
37 });


```

- Esse id é informado como um parâmetro na própria chamada do serviço (veremos como isso é feito no *front-end*).

CRUD MySQL: Back-End - Rota Cliente: listaCliente

```
22 // lista os dados de um cliente
23 router.get('/listaCliente', function (req, res, next) {
24     // recupera id do cliente que deverá ser consultado
25     var id = req.query.id;
26     req.getConnection(function (err, connection) {
27         if (err)
28             res.json({ status: 'ERRO', data: err.toString() });
29
30         connection.query("SELECT * FROM cliente WHERE id = " + id, function (err, rows) {
31             if (err)
32                 res.json({ status: 'ERRO', data: err.toString() });
33             else
34                 res.json({ status: 'OK', data: rows });
35         });
36     });
37 });


```

- Caso a conexão seja bem sucedida, fazemos a consulta no banco informando o id do cliente. Em caso de sucesso, os dados são retornados em formato json.

CRUD MySQL: Back-End - Rota Cliente: deleta

```
39 // deleta um cliente
40 router.post('/deleta', function (req, res, next) {
41   var id = req.query.id;
42   req.getConnection(function (err, connection) {
43     if (err)
44       res.json({ status: 'ERRO', data: err.toString() });
45
46     connection.query("DELETE FROM cliente WHERE id = " + id, function (err, rows) {
47       if (err)
48         res.json({ status: 'ERRO', data: err.toString() });
49       else
50         res.json({ status: 'OK', data: 'Excluído com sucesso!' });
51     });
52   });
53 });
```

- O próximo serviço é o `deleta` que exclui um cliente informado. Este serviço é acessado via `post` e retorna mensagem de sucesso ou erro conforme o resultado da operação.

CRUD MySQL: Back-End - Rota Cliente: deleta

```
39 // deleta um cliente
40 router.post('/deleta', function (req, res, next) {
41   var id = req.query.id;
42   req.getConnection(function (err, connection) {
43     if (err)
44       res.json({ status: 'ERRO', data: err.toString() });
45
46     connection.query("DELETE FROM cliente WHERE id = " + id, function (err, rows) {
47       if (err)
48         res.json({ status: 'ERRO', data: err.toString() });
49       else
50         res.json({ status: 'OK', data: 'Excluído com sucesso!' });
51     });
52   });
53 });
```

- O código é semelhante ao listaCliente: recupera o id do cliente que será excluído e faz a deleção desse cliente do banco de dados.

CRUD MySQL: Back-End - Rota Cliente: insere

```
55 // insere um cliente
56 router.post('/insere', function (req, res, next) {
57   // recupera as informações enviadas
58   var input = req.body;
59   req.getConnection(function (err, connection) {
60     if (err)
61       res.json({ status: 'ERRO', data: err.toString() });
62
63     connection.query("INSERT INTO cliente SET ? ", input, function (err, rows) {
64       if (err)
65         res.json({ status: 'ERROR', data: err.toString() });
66       else
67         res.json({ status: 'OK', data: 'Incluído com sucesso!' });
68     });
69   });
70 });
```

- O próximo serviço é o `insere` que insere um cliente de acordo com os dados informados. Também é acessado via `post` e retorna mensagem de sucesso ou erro conforme o resultado da operação.

CRUD MySQL: Back-End - Rota Cliente: insere

```
55 // insere um cliente
56 router.post('/insere', function (req, res, next) {
57     // recupera as informações enviadas
58     var input = req.body;
59     req.getConnection(function (err, connection) {
60         if (err)
61             res.json({ status: 'ERRO', data: err.toString() });
62
63         connection.query("INSERT INTO cliente SET ? ", input, function (err, rows) {
64             if (err)
65                 res.json({ status: 'ERROR', data: err.toString() });
66             else
67                 res.json({ status: 'OK', data: 'Incluído com sucesso!' });
68         });
69     });
70 });
```

- Já que é um `post`, recuperamos as informações com `req.body` e armazenamos essas informações em um objeto chamado `input`.

CRUD MySQL: Back-End - Rota Cliente: insere

```
55 // insere um cliente
56 router.post('/insere', function (req, res, next) {
57   // recupera as informações enviadas
58   var input = req.body;
59   req.getConnection(function (err, connection) {
60     if (err)
61       res.json({ status: 'ERRO', data: err.toString() });
62
63     connection.query("INSERT INTO cliente SET ? ", input, function (err, rows) {
64       if (err)
65         res.json({ status: 'ERROR', data: err.toString() });
66       else
67         res.json({ status: 'OK', data: 'Incluído com sucesso!' });
68     });
69   });
70 });
```

- Depois usamos esse objeto `input` para fazer o *insert* do cliente no banco de dados.

CRUD MySQL: Back-End - Rota Cliente: altera

```
72 // altera um cliente
73 router.post('/altera', function (req, res, next) {
74   var input = req.body;
75   var id = req.query.id;
76   req.getConnection(function (err, connection) {
77     if (err)
78       res.json({ status: 'ERRO', data: err.toString() });
79
80     connection.query("UPDATE cliente set ? WHERE id = ? ", [input, id], function (err, rows) {
81       if (err)
82         res.json({ status: 'ERRO', data: err.toString() });
83       else
84         res.json({ status: 'OK', data: 'Alterado com sucesso!' });
85     });
86   });
87 });
88 };
```

- Por fim, o último serviço é o altera que altera os dados de um cliente de acordo com os dados informados. Também é acessado via `post` e retorna mensagem de sucesso ou erro conforme o resultado da operação.

CRUD MySQL: Back-End - Rota Cliente: altera

```
72 // altera um cliente
73 router.post('/altera', function (req, res, next) {
74     var input = req.body;
75     var id = req.query.id;
76     req.getConnection(function (err, connection) {
77         if (err)
78             res.json({ status: 'ERRO', data: err.toString() });
79
80         connection.query("UPDATE cliente set ? WHERE id = ? ", [input, id], function (err, rows) {
81             if (err)
82                 res.json({ status: 'ERRO', data: err.toString() });
83             else
84                 res.json({ status: 'OK', data: 'Alterado com sucesso!' });
85         });
86     });
87 });
88 };
```

- Aqui, temos uma combinação de recebimento de informações: os dados alterados do cliente são recebidos de um formulário via `post` e o `id` do cliente é recebido como uma parâmetro da própria chamada do serviço pelo *front-end*.

CRUD MySQL: Back-End - Rota Cliente: altera

```
72 // altera um cliente
73 router.post('/altera', function (req, res, next) {
74     var input = req.body;
75     var id = req.query.id;
76     req.getConnection(function (err, connection) {
77         if (err)
78             res.json({ status: 'ERRO', data: err.toString() });
79
80         connection.query("UPDATE cliente set ? WHERE id = ? ", [input, id], function (err, rows) {
81             if (err)
82                 res.json({ status: 'ERRO', data: err.toString() });
83             else
84                 res.json({ status: 'OK', data: 'Alterado com sucesso!' });
85         });
86     });
87 });
88 };
```

- Assim, os dados alterados são recebidos utilizando `req.body` e armazenado no objeto `input`.

CRUD MySQL: Back-End - Rota Cliente: altera

```
72 // altera um cliente
73 router.post('/altera', function (req, res, next) {
74   var input = req.body;
75   var id = req.query.id;
76   req.getConnection(function (err, connection) {
77     if (err)
78       res.json({ status: 'ERRO', data: err.toString() });
79
80     connection.query("UPDATE cliente set ? WHERE id = ? ", [input, id], function (err, rows) {
81       if (err)
82         res.json({ status: 'ERRO', data: err.toString() });
83       else
84         res.json({ status: 'OK', data: 'Alterado com sucesso!' });
85     });
86   });
87 });
88 };
```

- E o id do cliente que será alterado é recebido via `req.query` e armazenado em `id`.

CRUD MySQL: Back-End - Rota Cliente: altera

```
72 // altera um cliente
73 router.post('/altera', function (req, res, next) {
74   var input = req.body;
75   var id = req.query.id;
76   req.getConnection(function (err, connection) {
77     if (err)
78       res.json({ status: 'ERRO', data: err.toString() });
79
80     connection.query("UPDATE cliente set ? WHERE id = ? ", [input, id], function (err, rows) {
81       if (err)
82         res.json({ status: 'ERRO', data: err.toString() });
83       else
84         res.json({ status: 'OK', data: 'Alterado com sucesso!' });
85     });
86   });
87 });
88 };
```

- Em seguida, usamos `input` e `id` como parâmetros do comando `update` para atualização do cliente no banco de dados.



CRUD MySQL: *Back-End* - Rota Cliente

- Por fim, precisamos incluir a operação criada nos módulos.

```
module.exports = router;
```

- Com isso, fechamos a rota cliente com todas as operações do CRUD e concluímos o *back-end*.



CRUD MySQL

Passos para implementação:

Banco de Dados:

- ✓ Criar banco de dados.

Programação:

- ✓ *Back-End:* serviços CRUD;
- Front-End:* listagem e manipulação dados.



CRUD MySQL: *Front-End* - listaCliente

- Vamos criar uma página inicial contendo apenas um link para incluir um cliente (lembrem-se que o nosso foco não é o *layout* das páginas e sim a integração dos serviços).

[INCLUIR CLIENTE](#)

- Essa página também irá listar todos os clientes que estão cadastrados no banco de dados.

CRUD MySQL: *Front-End* - listaCliente

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Node + MySQL</title>
6      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
7      <script type="text/javascript" src="javascripts/cliente.js"></script>
8      <script>
9          $(document).ready(function () { buscaClientes(); });
10     </script>
11 </head>
12
13 <body>
14     <a href="formCliente.html">INCLUIR CLIENTE</a><br><br>
15     <div id="result"></div>
16 </body>
17
18 </html>
```

- Primeiro vamos criar a página listaCliente.html

CRUD MySQL: *Front-End* - listaCliente

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Node + MySQL</title>
6      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
7      <script type="text/javascript" src="javascripts/cliente.js"></script>
8      <script>
9          $(document).ready(function () { buscaClientes(); });
10     </script>
11 </head>
12
13 <body>
14     <a href="formCliente.html">INCLUIR CLIENTE</a><br><br>
15     <div id="result"></div>
16 </body>
17
18 </html>
```

- Vamos mais um vez utilizar a biblioteca jquery e importar cliente.js que conterá nossas funções de acesso aos serviços.

CRUD MySQL: *Front-End* - listaCliente

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Node + MySQL</title>
6      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
7      <script type="text/javascript" src="javascripts/cliente.js"></script>
8      <script>
9          $(document).ready(function () { buscaClientes(); });
10     </script>
11 </head>
12
13 <body>
14     <a href="formCliente.html">INCLUIR CLIENTE</a><br><br>
15     <div id="result"></div>
16 </body>
17
18 </html>
```

- Quando a página finalizar o carregamento de todos os elementos, chamamos a função **buscaClientes** que acessará o serviço `listar` e exibirá as informações.

CRUD MySQL: *Front-End* - listaCliente

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Node + MySQL</title>
6      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
7      <script type="text/javascript" src="javascripts/cliente.js"></script>
8      <script>
9          $(document).ready(function () { buscaClientes(); });
10     </script>
11 </head>
12
13 <body>
14     <a href="formCliente.html">INCLUIR CLIENTE</a><br><br>
15     <div id="result"></div>
16 </body>
17
18 </html>
```

- Temos o link para incluir um cliente que chamará a página formCliente.html.

CRUD MySQL: *Front-End - listaCliente*

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Node + MySQL</title>
6      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
7      <script type="text/javascript" src="javascripts/cliente.js"></script>
8      <script>
9          $(document).ready(function () { buscaClientes(); });
10     </script>
11 </head>
12
13 <body>
14     <a href="formCliente.html">INCLUIR CLIENTE</a><br><br>
15     <div id="result"></div>
16 </body>
17
18 </html>
```

- E a div **result** que exibirá os dados dos clientes cadastrados no banco de dados.

CRUD MySQL: *Front-End* - formCliente

- O formulário formCliente.html de inserção/alteração de cliente terá o seguinte *layout*:

Formulário de Cliente

Nome:

Endereço:

Email:

Telefone:

salva

cancela



CRUD MySQL: *Front-End* - formCliente

- É importante observar que esse formulário será utilizado tanto para inclusão de um novo cliente quanto para a alteração de um cliente já existente.
- Essa estratégia é muito interessante muito nos permite ter uma única página de formulário para as duas operações, evitando a necessidade de criar dois arquivos separados (um por operação) e posterior retrabalho.

CRUD MySQL: *Front-End* - formCliente

```
1  <!DOCTYPE html>
2
3  </html>
4
5  <head>
6      <title>Cliente</title>
7      <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
8      <script type="text/javascript" src="javascripts/cliente.js"></script>
9      <script>
10         $(document).ready(function () { alteraCliente(); });
11     </script>
12 </head>
```

- Quando a página finalizar o carregamento de todos os elementos, chamamos a função `alteraCliente` que acessará o serviço `listaCliente` e exibirá as informações nos respectivos inputs de campos.

CRUD MySQL: *Front-End* - formCliente

```
14 <body>
15   <h2>Formulário de Cliente</h2>
16   <form name="formCliente" method="post" action="">
17     Nome: <br><input type="text" name="nome" required><br>
18     Endereço: <br><textarea name="endereco" required cols="30" rows="5"></textarea><br>
19     Email: <br><input type="text" name="email" required><br>
20     Telefone: <br><input type="text" name="telefone" required><br>
21     <br>
22     <input type="button" name="salva" value="salva" onClick="salvaCliente()">
23     <input type="button" name="cancela" value="cancela" onClick="window.location.href = '/listaCliente.html';">
24   </form>
25 </body>
26
27 </html>
```

- Por fim, temos o formulário com os botões salva e cancela (que retorna para a lista de clientes).



CRUD MySQL: *Front-End* – cliente.js

- O último passo é criar o nosso arquivo cliente.js que vai conter todas as funções de chamada dos serviços e exibição dos dados para o usuário.

CRUD MySQL: *Front-End* - formCliente

```
1 // chama o serviço que retorna todos os clientes cadastrados
2 function buscaClientes() {
3     $.get('/cliente/lista',
4         // tratamento de sucesso de processamento do post
5         function (returnedData, statusRequest) {
6             // se ocorreu algum erro no processamento da API
7             if (returnedData.status === 'ERRO')
8                 alert('Erro:' + returnedData.data);
9             // caso os dados tenham retornado com sucesso
10            // exibe as informações
11            else
12                exibeClientes(returnedData.data);
13        }
14    )
15    // tratamento de erro do get
16    .fail(function (returnedData) {
17        alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
18    });
19 }
```

- Nossa primeira função do arquivo é a **buscaClientes** que é chamada logo ao carregar o html da listagem de clientes.

CRUD MySQL: *Front-End* - formCliente

```
1 // chama o serviço que retorna todos os clientes cadastrados
2 function buscaClientes() {
3     $.get('/cliente/lista',
4         // tratamento de sucesso de processamento do post
5         function (returnedData, statusRequest) {
6             // se ocorreu algum erro no processamento da API
7             if (returnedData.status === 'ERRO')
8                 alert('Erro:' + returnedData.data);
9             // caso os dados tenham retornado com sucesso
10            // exibe as informações
11            else
12                exibeClientes(returnedData.data);
13        }
14    )
15    // tratamento de erro do get
16    .fail(function (returnedData) {
17        alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
18    });
19 }
```

- A função faz um `get` para o serviço `lista` e envia os dados retornados pelo serviço para a função `exibeClientes`.

CRUD MySQL: *Front-End* - formCliente

```
21 // exibe os clientes retornados na tela
22 function exibeClientes(clientes) {
23     for (i = 0; i < clientes.length; i++) {
24         var cliente = clientes[i];
25         var dadosCliente = '<div id="' + cliente.id + '"> ' +
26             'ID: ' + cliente.id +
27             '<br>Nome: ' + cliente.nome +
28             '<br>Endereço: ' + cliente.endereco +
29             '<br>Telefone: ' + cliente.telefone +
30             '<br>Email: ' + cliente.email +
31             '<br><a href="formCliente.html?id=' + cliente.id + '">ALTERAR</a>' +
32             ' - <a href="#" onclick="deletaCliente(' + cliente.id + ')">EXCLUIR</a>' +
33             '<br><br></div>';
34         document.getElementById('result').innerHTML += dadosCliente;
35     }
36 }
```

- A função **exibeClientes** recebe o json retornado pelo serviço e lista as informações na tela para o usuário. Utilizamos a div **result** criada na página html para exibir as informações.

CRUD MySQL: *Front-End* - formCliente

```
21 // exibe os clientes retornados na tela
22 function exibeClientes(clientes) {
23     for (i = 0; i < clientes.length; i++) {
24         var cliente = clientes[i];
25         var dadosCliente = '<div id="' + cliente.id + '"> ' +
26             'ID: ' + cliente.id +
27             '<br>Nome: ' + cliente.nome +
28             '<br>Endereço: ' + cliente.endereco +
29             '<br>Telefone: ' + cliente.telefone +
30             '<br>Email: ' + cliente.email +
31             '<br><a href="formCliente.html?id=' + cliente.id + '">ALTERAR</a>' +
32             ' - <a href="#" onclick="deletaCliente(' + cliente.id + ');">EXCLUIR</a>' +
33             '<br><br></div>' +
34         document.getElementById('result').innerHTML += dadosCliente;
35     }
36 }
```

- Observem que ao criar os links para alteração e para exclusão do cliente enviamos o **id** desse cliente como parâmetro.

CRUD MySQL: *Front-End* - formCliente

```
21 // exibe os clientes retornados na tela
22 function exibeClientes(clientes) {
23     for (i = 0; i < clientes.length; i++) {
24         var cliente = clientes[i];
25         var dadosCliente = '<div id="' + cliente.id + '"> ' +
26             'ID: ' + cliente.id +
27             '<br>Nome: ' + cliente.nome +
28             '<br>Endereço: ' + cliente.endereco +
29             '<br>Telefone: ' + cliente.telefone +
30             '<br>Email: ' + cliente.email +
31             '<br><a href="formCliente.html?id=' + cliente.id + '">ALTERAR</a>' +
32             ' - <a href="#" onclick="deletaCliente(' + cliente.id + ');">EXCLUIR</a>' +
33             '<br><br></div>' +
34         document.getElementById('result').innerHTML += dadosCliente;
35     }
36 }
```

- Para a alteração chamamos o formulário de cliente e enviamos o **id**.

CRUD MySQL: *Front-End* - formCliente

```
21 // exibe os clientes retornados na tela
22 function exibeClientes(clientes) {
23     for (i = 0; i < clientes.length; i++) {
24         var cliente = clientes[i];
25         var dadosCliente = '<div id="' + cliente.id + '"> ' +
26             'ID: ' + cliente.id +
27             '<br>Nome: ' + cliente.nome +
28             '<br>Endereço: ' + cliente.endereco +
29             '<br>Telefone: ' + cliente.telefone +
30             '<br>Email: ' + cliente.email +
31             '<br><a href="formCliente.html?id=' + cliente.id + '">ALTERAR</a>' +
32             ' - <a href="#" onclick="deletaCliente(' + cliente.id + ');">EXCLUIR</a>' +
33             '<br><br></div>';
34         document.getElementById('result').innerHTML += dadosCliente;
35     }
36 }
```

- Para a exclusão chamamos a função **deletaCliente** e também enviamos o **id**.

CRUD MySQL: *Front-End* - formCliente

```
38 // deleta o cliente do id recebido
39 function deletaCliente(id) {
40     $.post('/cliente/deleta?id=' + id,
41         // tratamento de sucesso de processamento do post
42         function (returnedData, statusRequest) {
43             // se ocorreu algum erro no processamento da API
44             if (returnedData.status === 'ERRO')
45                 alert('Erro:' + returnedData.data);
46             // caso os dados tenham retornado com sucesso
47             // remove o cliente da exibição da página e mostra msg
48             else {
49                 var divResult = document.getElementById('result');
50                 divResult.removeChild(document.getElementById(id));
51                 alert(returnedData.data);
52             }
53         }
54     )
55     // tratamento de erro do post
56     .fail(function (returnedData) {
57         alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
58     });
59 }
```

- A função `deletaCliente` recebe o `id` do cliente a ser excluído e chama o serviço `deleta` passando esse id.

CRUD MySQL: *Front-End* - formCliente

```
38 // deleta o cliente do id recebido
39 function deletaCliente(id) {
40     $.post('/cliente/deleta?id=' + id,
41         // tratamento de sucesso de processamento do post
42         function (returnedData, statusRequest) {
43             // se ocorreu algum erro no processamento da API
44             if (returnedData.status === 'ERRO')
45                 alert('Erro:' + returnedData.data);
46             // caso os dados tenham retornado com sucesso
47             // remove o cliente da exibição da página e mostra msg
48             else {
49                 var divResult = document.getElementById('result');
50                 divResult.removeChild(document.getElementById(id));
51                 alert(returnedData.data);
52             }
53         }
54     )
55     // tratamento de erro do post
56     .fail(function (returnedData) {
57         alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
58     });
59 }
```

- Note que, em caso de sucesso, precisamos atualizar os elementos dentro da div **result**, removendo os dados do cliente deletado (atualização da página).



CRUD MySQL: *Front-End* – cliente.js

- A próxima função é a `salvaCliente`.
- Essa função é chamada no formulário de cliente tanto para alteração quanto para inclusão:
 - Quando o formulário é chamado para inclusão, não recebemos um id de cliente como parâmetro na chamada da URL.
 - Já quando o formulário é chamado para alteração, recebemos o id do cliente a ser alterado como parâmetro na chamada da URL.
- Essa distinção nos permitirá determinar se devemos chamar o serviço de inclusão ou de alteração de cliente.

CRUD MySQL: *Front-End* - formCliente

```
61 // salva os dados do cliente
62 function salvaCliente() {
63     // recupera as informações do formulário
64     var form = document.formCliente;
65     var input = {
66         nome: form.nome.value,
67         endereco: form.endereco.value,
68         email: form.email.value,
69         telefone: form.telefone.value
70     };
71
72     // recupera os parâmetros da URL da página
73     var urlParams = new URLSearchParams(window.location.search);
74     var acao;
75     // se tiver um id na URL, o form é de alteração
76     if (urlParams.has('id')) {
77         acao = '/cliente/altera?id=' + urlParams.get('id');
78     // caso contrário, o form é de inclusão
79     } else {
80         acao = '/cliente/insere';
81     }
82 }
```

- A função `salvaCliente` está dividida nos slides em dois blocos de código.

CRUD MySQL: *Front-End* - formCliente

```
61 // salva os dados do cliente
62 function salvaCliente() {
63     // recupera as informações do formulário
64     var form = document.formCliente;
65     var input = {
66         nome: form.nome.value,
67         endereco: form.endereco.value,
68         email: form.email.value,
69         telefone: form.telefone.value
70     };
71
72     // recupera os parâmetros da URL da página
73     var urlParams = new URLSearchParams(window.location.search);
74     var acao;
75     // se tiver um id na URL, o form é de alteração
76     if (urlParams.has('id')) {
77         acao = '/cliente/altera?id=' + urlParams.get('id');
78     // caso contrário, o form é de inclusão
79     } else {
80         acao = '/cliente/insere';
81     }
82 }
```

- Primeiro recuperamos todas as informações preenchidas o formulário. Independente de ser uma operação de inclusão ou de alteração de clientes, precisamos recuperar todas as informações fornecidas.

CRUD MySQL: *Front-End* - formCliente

```
61 // salva os dados do cliente
62 function salvaCliente() {
63     // recupera as informações do formulário
64     var form = document.formCliente;
65     var input = {
66         nome: form.nome.value,
67         endereco: form.endereco.value,
68         email: form.email.value,
69         telefone: form.telefone.value
70     };
71
72     // recupera os parâmetros da URL da página
73     var urlParams = new URLSearchParams(window.location.search);
74     var acao;
75     // se tiver um id na URL, o form é de alteração
76     if (urlParams.has('id')) {
77         acao = '/cliente/altera?id=' + urlParams.get('id');
78     // caso contrário, o form é de inclusão
79     } else {
80         acao = '/cliente/insere';
```

- O próximo passo é recuperar os parâmetros da página e identificar se recebemos o id do cliente como parâmetro na página.

CRUD MySQL: *Front-End* - formCliente

```
61 // salva os dados do cliente
62 function salvaCliente() {
63     // recupera as informações do formulário
64     var form = document.formCliente;
65     var input = {
66         nome: form.nome.value,
67         endereco: form.endereco.value,
68         email: form.email.value,
69         telefone: form.telefone.value
70     };
71
72     // recupera os parâmetros da URL da página
73     var urlParams = new URLSearchParams(window.location.search);
74     var acao;
75     // se tiver um id na URL, o form é de alteração
76     if (urlParams.has('id')) {
77         acao = '/cliente/altera?id=' + urlParams.get('id');
78     // caso contrário, o form é de inclusão
79     } else {
80         acao = '/cliente/insere';
81     }
82 }
```

- Se identificarmos o id do cliente, vamos chamar o serviço **altera** enviando o id do cliente a ser alterado e, se não houver id do cliente, vamos chamar o serviço **insere**. A definição do serviço a ser chamado ficará gravado na variável **acao**.

CRUD MySQL: *Front-End* - formCliente

```
82 // post para o serviço de inclusão ou alteração
83 $.post(acao,
84     input,
85     // tratamento de sucesso de processamento do post
86     function (returnedData, statusRequest) {
87         // se ocorreu algum erro no processamento da API
88         if (returnedData.status === 'ERRO')
89             alert('Erro:' + returnedData.data);
90         // caso os dados tenham retornado com sucesso
91         // exibe mensagem de confirmação
92         else
93             alert(returnedData.data);
94     }
95 )
96 // tratamento de erro do post
97 .fail(function (returnedData) {
98     alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
99 });
100 }
```

- No segundo bloco da função, fazemos o post para o serviço definido e exibimos mensagem de acordo com o retorno.

CRUD MySQL: *Front-End* - formCliente

```
82 // post para o serviço de inclusão ou alteração
83 $.post(acao,
84     input,
85     // tratamento de sucesso de processamento do post
86     function (returnedData, statusRequest) {
87         // se ocorreu algum erro no processamento da API
88         if (returnedData.status === 'ERRO')
89             alert('Erro:' + returnedData.data);
90         // caso os dados tenham retornado com sucesso
91         // exibe mensagem de confirmação
92         else
93             alert(returnedData.data);
94     }
95 )
96 // tratamento de erro do post
97 .fail(function (returnedData) {
98     alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
99 });
100 }
```

- Note que passamos como parâmetro para a chamada do serviço a variável **acao** que contém a URL do serviço adequado (inclusão ou alteração) e o objeto **input** com os dados do cliente.

CRUD MySQL: *Front-End* - formCliente

```
102 // carrega os dados do cliente no formulário para alteração
103 function alteraCliente() {
104     // recupera os parâmetros da URL da página para obter o id
105     var urlParams = new URLSearchParams(window.location.search);
106     if (urlParams.has('id')) {
107         $.get('/cliente/listaCliente?id=' + urlParams.get('id'),
108             // tratamento de sucesso de processamento do post
109             function (returnedData, statusRequest) {
110                 // se ocorreu algum erro no processamento da API
111                 if (returnedData.status === 'ERRO')
112                     alert('Erro:' + returnedData.data);
113                 // caso os dados tenham retornado com sucesso
114                 // atribui as informações retornadas aos campos
115                 else {
116                     var form = document.formCliente;
117                     form.nome.value = returnedData.data[0].nome;
118                     form.endereco.value = returnedData.data[0].endereco;
119                     form.email.value = returnedData.data[0].email;
120                     form.telefone.value = returnedData.data[0].telefone;
121                 }
122             }
123     }
}
```

- A nossa última função é a `alteraCliente` que recupera as informações do cliente que será alterado e exibe essas informações no formulário de cliente

CRUD MySQL: *Front-End* - formCliente

```
102 // carrega os dados do cliente no formulário para alteração
103 function alteraCliente() {
104     // recupera os parâmetros da URL da página para obter o id
105     var urlParams = new URLSearchParams(window.location.search);
106     if (urlParams.has('id')) {
107         $.get('/cliente/listaCliente?id=' + urlParams.get('id'),
108             // tratamento de sucesso de processamento do post
109             function (returnedData, statusRequest) {
110                 // se ocorreu algum erro no processamento da API
111                 if (returnedData.status === 'ERRO')
112                     alert('Erro:' + returnedData.data);
113                 // caso os dados tenham retornado com sucesso
114                 // atribui as informações retornadas aos campos
115                 else {
116                     var form = document.formCliente;
117                     form.nome.value = returnedData.data[0].nome;
118                     form.endereco.value = returnedData.data[0].endereco;
119                     form.email.value = returnedData.data[0].email;
120                     form.telefone.value = returnedData.data[0].telefone;
121                 }
122             }
123     }
}
```

- Primeiro recuperamos o id do cliente a ser alterado e enviamos esse id para o serviço listaCliente.

CRUD MySQL: *Front-End* - formCliente

```
102 // carrega os dados do cliente no formulário para alteração
103 function alteraCliente() {
104     // recupera os parâmetros da URL da página para obter o id
105     var urlParams = new URLSearchParams(window.location.search);
106     if (urlParams.has('id')) {
107         $.get('/cliente/listaCliente?id=' + urlParams.get('id'),
108             // tratamento de sucesso de processamento do post
109             function (returnedData, statusRequest) {
110                 // se ocorreu algum erro no processamento da API
111                 if (returnedData.status === 'ERRO')
112                     alert('Erro:' + returnedData.data);
113                 // caso os dados tenham retornado com sucesso
114                 // atribui as informações retornadas aos campos
115                 else {
116                     var form = document.formCliente;
117                     form.nome.value = returnedData.data[0].nome;
118                     form.endereco.value = returnedData.data[0].endereco;
119                     form.email.value = returnedData.data[0].email;
120                     form.telefone.value = returnedData.data[0].telefone;
121                 }
122             }
123     }
}
```

- Em caso de sucesso, atribuímos cada uma das informações retornadas do cliente aos respectivos campos do formulário.

CRUD MySQL: *Front-End* - formCliente

```
124     // tratamento de erro do get
125     .fail(function (returnedData) {
126         alert('Erro: ' + returnedData.status + ' ' + returnedData.statusText);
127     });
128 }
129 }
```

- Para fechar a função o tratamento em caso de falha no post.



CRUD MySQL

Passos para implementação:

✓ **Banco de Dados:**

- ✓ Criar banco de dados.

✓ **Programação:**

- ✓ *Back-End*: serviços CRUD;
- ✓ *Front-End*: listagem e manipulação dados.



Exemplo de Aplicação CRUD com MySQL

- Pronto! Já temos um CRUD completo para tratamento de dados de clientes usando node e MySQL hospedado no Microsoft Azure!