P.PORTO

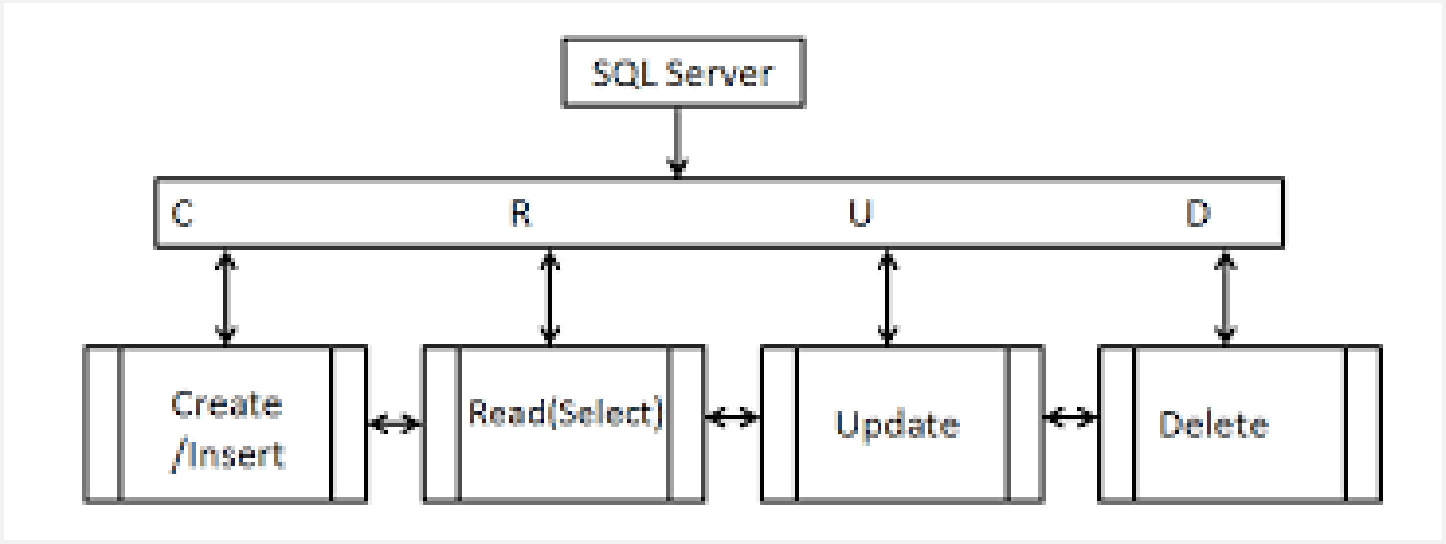
POLITÉCNICO
DO PORTO
ESMAD

PROGRAMAÇÃO WEB II

TSIW

SUMMARY

- Simple CRUD operations using Node + MySQL



Node: simple MySQL operations

- To use a MySQL database in Node, you need to install the **`mysql`** module, a MySQL driver for Node
  - Create a new diretory and initialize a Node project using NPM
    ```
    mkdir mysqlexperiment && cd mysqlexperiment
    ```
  - Create a `package.json` file with default values (to be changed afterwards)
    ```
    npm init -y
    ```
  - Install the `mysql` and `nodemon` (if not already) node modules
    ```
    npm install --save mysql
    npm install --save-dev nodemon
    ```
  - Create an `connect.js` file and copy/paste the code shown in next slide
  - Run the code from next slide to check if you have a MySQL server running on your computer

Node: simple MySQL operations

File **connect.js**

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

//Node.js MySQL database connection
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'yourusername',
    password: 'yourpassword',
});

connection.connect((err) => {
    if (err) throw err;
    console.log('Connected to MySQL Server!');
});
```

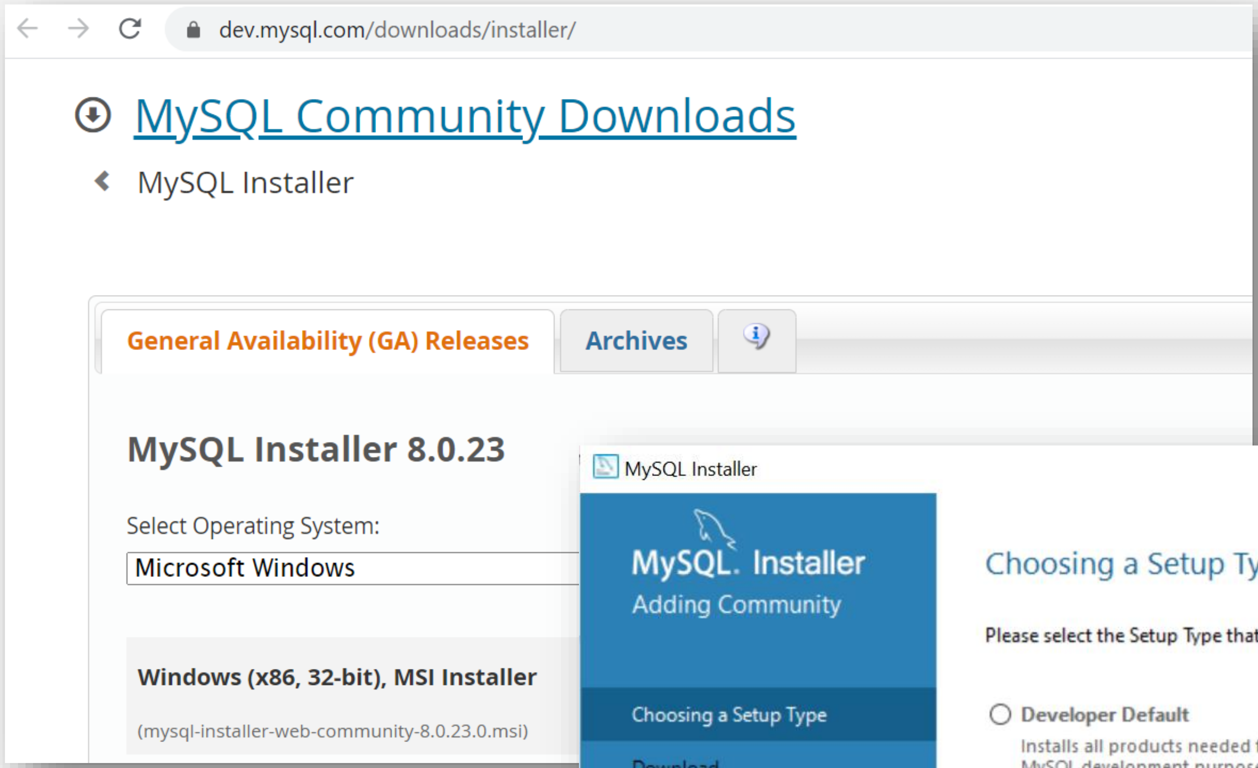
Replace *yourusername* and *yourpassword* with your SQL user credentials

IF you do not have MySQL server installed and configured on your machine, please consult the installation instructions on their home page

If you have the latest MySQL server installed, you might end up getting an error saying: "Client does not support authentication …". To tackle this issue, create a new user in your MySQL server with 'mysql_native_password' authentication mechanism:

1. log in to the MySQL server using root access and type:
2. CREATE USER '***newuser***'@'localhost' IDENTIFIED WITH 'mysql_native_password' BY '***newpassword***';
3. GRANT ALL PRIVILEGES ON *.* TO '***newuser***'@'localhost';
4. FLUSH PRIVILEGES;

Install MySQL


dev.mysql.com/downloads/installer/

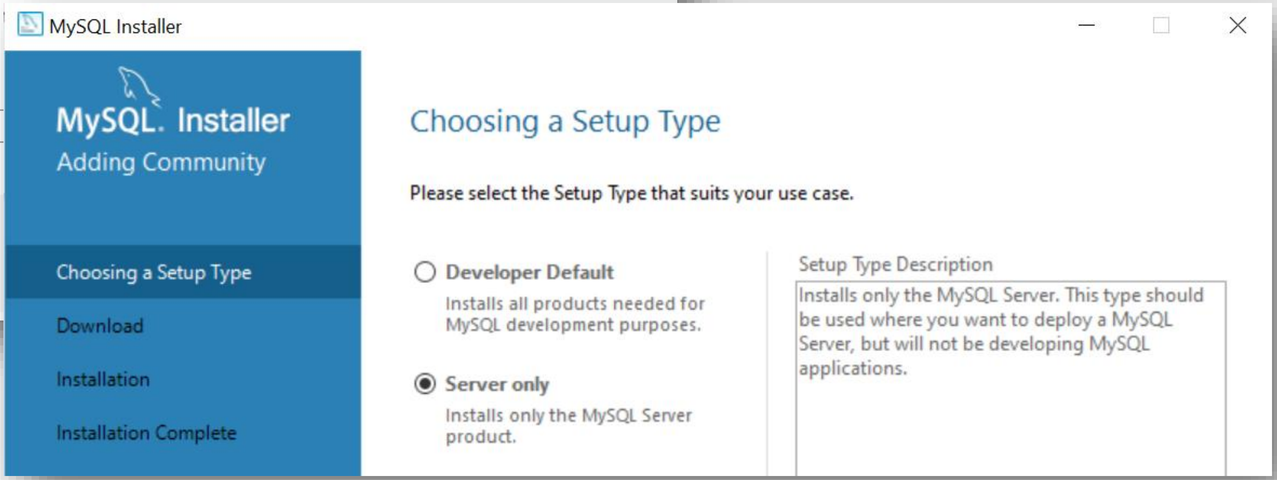
MySQL Community Downloads

MySQL Installer

MySQL Installer 8.0.23

Select Operating System:
Microsoft Windows

Windows (x86, 32-bit), MSI Installer
(mysql-installer-web-community-8.0.23.0.msi)



Choosing a Setup Type

Please select the Setup Type that suits your use case.

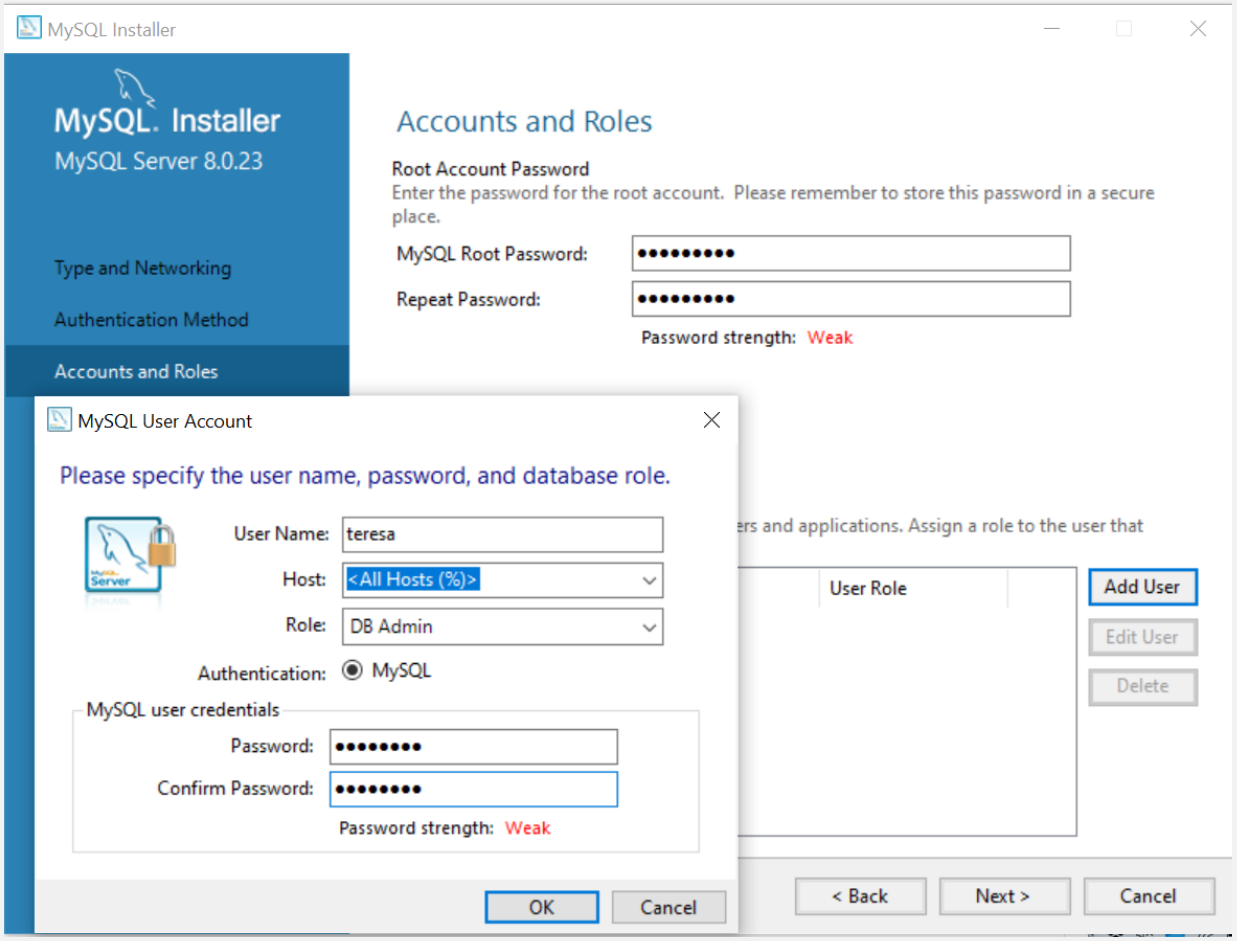
Developer Default
Installs all products needed for MySQL development purposes.

[✓ Server only]
Installs only the MySQL Server product.

Setup Type Description
Installs only the MySQL Server. This type should be used where you want to deploy a MySQL Server, but will not be developing MySQL applications.

IF you do not have MySQL server installed and configured on your machine, please consult the installation instructions on [their home page]

Install MySQL



MySQL Installer

MySQL. Installer
MySQL Server 8.0.23

Type and Networking
Authentication Method
Accounts and Roles

Accounts and Roles

Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: ●●●●●●●●●
Repeat Password: ●●●●●●●●●
Password strength: Weak

ers and applications. Assign a role to the user that

User Role

Add User
Edit User
Delete

< Back
Next >
Cancel

MySQL User Account

Please specify the user name, password, and database role.

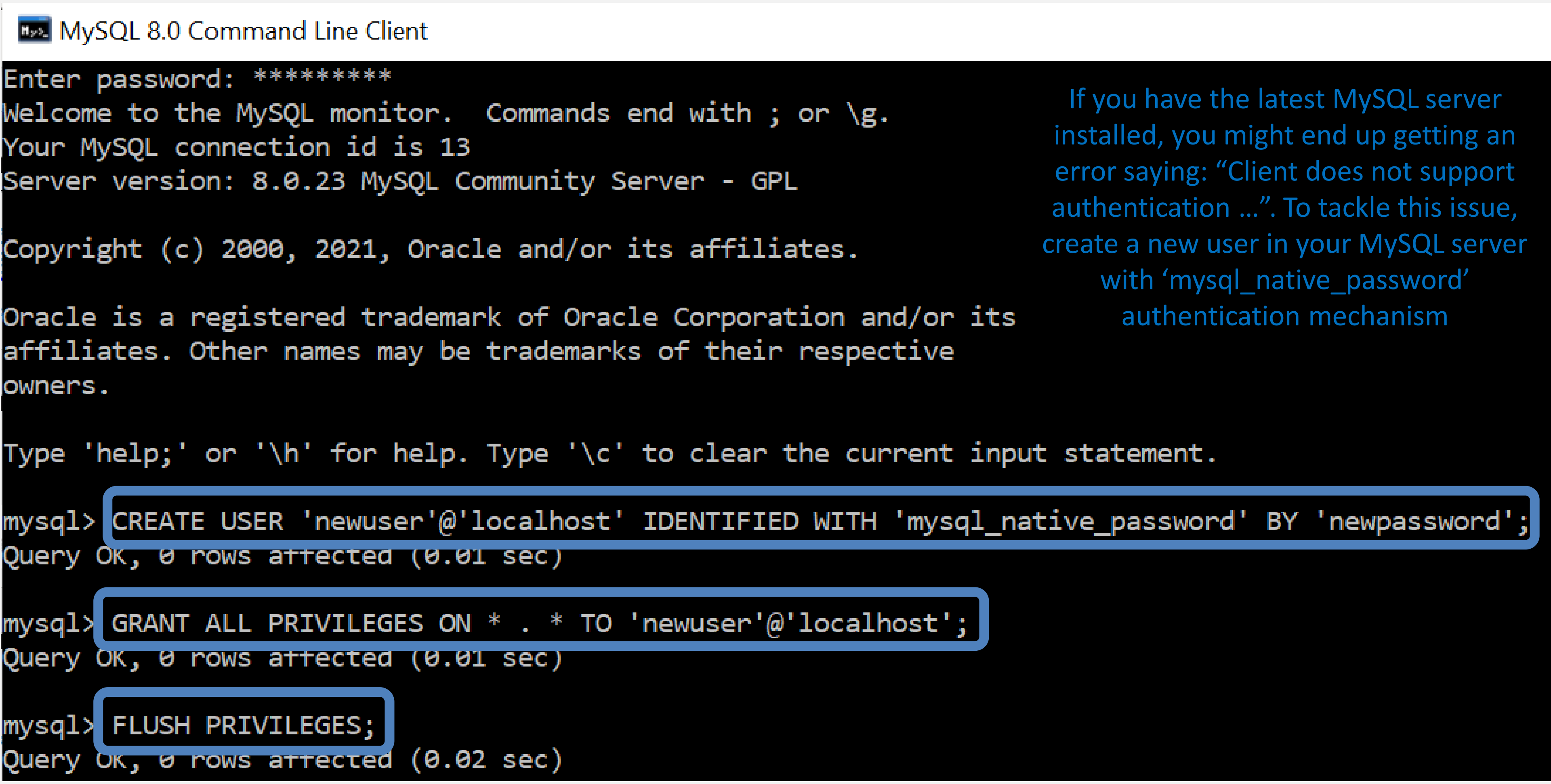
User Name: teresa
Host: <All Hosts (%)>
Role: DB Admin
Authentication: ● MySQL

MySQL user credentials
Password: ●●●●●●●●
Confirm Password: ●●●●●●●●
Password strength: Weak

OK
Cancel

Remember all the inputs provided at this point!

MySQL: create new user



MySQL 8.0 Command Line Client

```
Enter password: *********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED WITH 'mysql_native_password' BY 'newpassword';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.02 sec)
```

If you have the latest MySQL server installed, you might end up getting an error saying: “Client does not support authentication …”. To tackle this issue, create a new user in your MySQL server with ‘mysql_native_password’ authentication mechanism

Use other names for *newuser* and *newpassword* and use them into your Node projects!

Create a new database

- Use a MySQL GUI, or the MySQL command line, to create a new database:

```
CREATE DATABASE bookstore CHARACTER SET utf8 COLLATE
utf8_general_ci;
USE bookstore;

CREATE TABLE authors (
  id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(50),
  city varchar(50),
  PRIMARY KEY (id)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=5 ;

INSERT INTO authors (id, name, city) VALUES
(1, 'Michaela Lehr', 'Berlin'),
(2, 'Michael Wanyoike', 'Nairobi'),
(3, 'James Hibbard', 'Munich'),
(4, 'Almeida Garret', 'Oporto');
```

Node + MySQL

- Create a NODE web server and serve an HTML page with a table containing the database response from table `authors`, when user access the following URL http://localhost:3000/authors
- For better code understanding, separate the **server side coding** from the **SQL client connection**, by using the file `connect.js` as a module to expose the BD connection
  - ➢ Create a new file and name it as `index.js` or `server.js` to create and run the Node server
  - ➢ In this new file - the main application file (where the server is created) - import the module to connect the database server:

```
const connection = require('./connect');
```

Node + MySQL

File **connect.js**

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

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'newuser',
    password: 'newpassword',
    database: 'bookstore'
});

connection.connect(function (err) {
    if (err) throw err;
    console.log('Database is connected successfully !');
});

module.exports = connection;
```

Open a connection to database 'bookstore'

Export the database connection

```
…
// import module to connect with BD bookstore
const connection = require('./connect');
…
```

File **server.js:**
Main application file (where the server is created)
Complete it with the necessary code to run a server

Node + MySQL

- The most basic way to perform a query is to call the `.query()` method of the object connection

    `connection.query(sqlString, callback)`

```
connection.query('SELECT * FROM `books` WHERE `author` = "David"',
    function (error, results, fields) {
        // error will be an Error if one occurred during the query
        // results will contain the results of the query
        // fields will contain information about the returned results fields (if any)
});
```

- Learn more about how to perform queries: https://www.npmjs.com/package/mysql#performing-queries

```
…
const connection = require('./connect'); // import module to connect with BD bookstore
…
//check the URL of the current request
if (request.url == '/authors') {

     //start building the HTML response
     let txt = "<html><title>AUTHORS</title><body>";

     //query the BD with a SELECT statement
     connection.query('SELECT * FROM authors', (err, results) => {
          if (err) {
                //set response for DB acess error
                response.writeHead(500, { 'Content-Type': 'application/json' });
                response.end(JSON.stringify({ "ERROR": "Internal server error!" }));
          }
          // build the HTML table
          txt += `<table class='table' style='width:50%' border='1'>
                    <tr><th>Name</th><th>City</th></tr>`;

          // `results` is an array with one element for every row retrieved
          results.forEach(res => {
               txt += `<tr><td style='text align:right'>${res.name}
                         </td><td>${res.city}</td></tr>`; });
          txt += "</table></html></body>";
          response.writeHead(200, { 'Content-Type': 'text/html' });
          response.write(txt);
          response.end();
     });
} else … //set response (status code: 404) for invalid URL's
```

File **server.js**
(main application file)

Node + MySQL + JSON

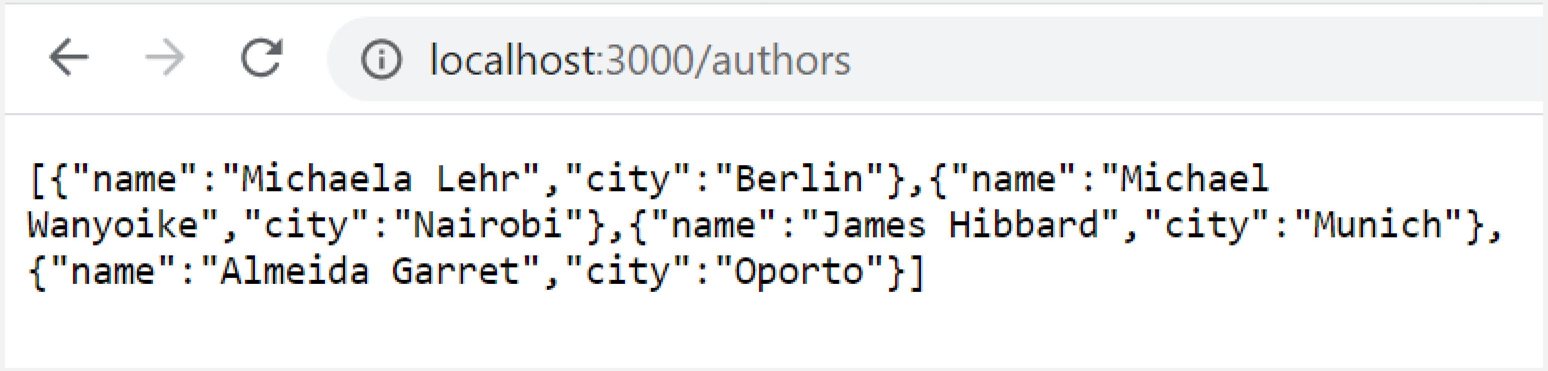
- Database query response can also be provided in a JSON format
  - ➢ Most used format in a **REST API client/server communication**
- Alter the server response to serve in a JSON (instead of HTML)

```
…
connection.query('SELECT name, city FROM authors', (err, results) => {
     if (err) …

     // wrap result-set as JSON string
     let json = JSON.stringify(results);

     // define HTTP head parameters
     response.writeHead(200, { 'Content-Type': 'application/json' });
     // set response to client
     response.end(json);
});
…
```

File **server.js**
(main application file)

Node + MySQL + JSON

- Test the application using the browser

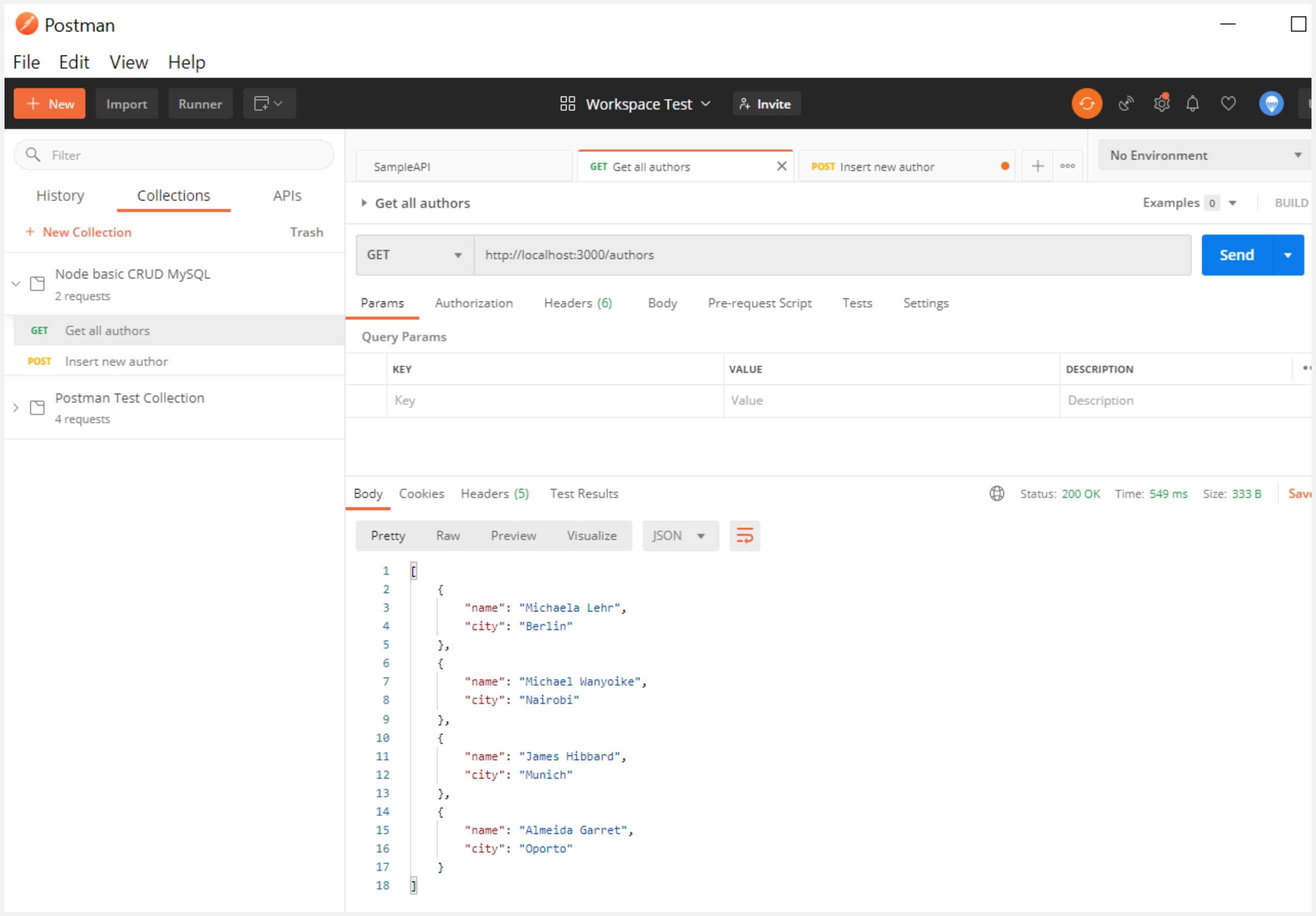


```
[{"name":"Michaela Lehr","city":"Berlin"},{"name":"Michael
Wanyoike","city":"Nairobi"},{"name":"James Hibbard","city":"Munich"},
{"name":"Almeida Garret","city":"Oporto"}]
```

- Test again using Postman
  - ➢ Great tool to help developing APIs (made by others or our ones), querying and/or test them
  - ➢ How to install and use Postman: https://learning.postman.com/docs/getting-started/installation-and-updates/

Node + MySQL + JSON

- Test again using Postman



GET http://localhost:3000/authors

Status: 200 OK Time: 549 ms Size: 333 B

```
[
    {
        "name": "Michaela Lehr",
        "city": "Berlin"
    },
    {
        "name": "Michael Wanyoike",
        "city": "Nairobi"
    },
    {
        "name": "James Hibbard",
        "city": "Munich"
    },
    {
        "name": "Almeida Garret",
        "city": "Oporto"
    }
]
```

Node + MySQL + JSON

- The second form to query is to call the `.query()` method using **placeholder** values (using **?** characters)
  - Multiple placeholders are mapped to values in the same order as passed

```
connection.query('SELECT * FROM table WHERE foo = ? AND bar = ?', ["a", "b"],
     function (error, results, fields) {
          …
});
```

- The third form comes when using various advanced options on the query, like timeouts, …

`connection.query(`*`options, callback`*`)`

```
connection.query({
     sql: 'SELECT * FROM book WHERE author = ?',
     timeout: 40000, // 40s
     values: ['David’]
     }, function (error, results, fields) {  …
});
```

Node + MySQL + JSON

- **Query placeholders**: different value types are placed differently
  - **Objects** are turned into `key = 'val'` pairs for each enumerable property on the object: `toString()` is called on it and the returned value is used
- This allows us to do neat things like this:

```
let post  = {id: 1, title: 'Hello MySQL'}; // object
let query = connection.query('INSERT INTO posts SET ?', post,
     function (error, results, fields) {
          …
});
console.log(query.sql); // INSERT INTO posts SET `id` = 1, `title` = 'Hello MySQL'
```

Node + MySQL + JSON

- In most real-life applications, **data** to be inserted into a server database comes from data in the **body of the HTTP request**
- Since we are not using Express (will be learned later), a bit more work is needed, as Express abstracts a lot of this
- The key thing to understand is that when the HTTP server starts using `http.createServer(callback)`, the callback is called when the server got all the HTTP headers, but **not the request body**
  - ➢ The request object passed in the connection callback is a **stream**
  - ➢ So, server must listen for the body content to be processed, and that it's processed in **chunks**

Node + MySQL + JSON

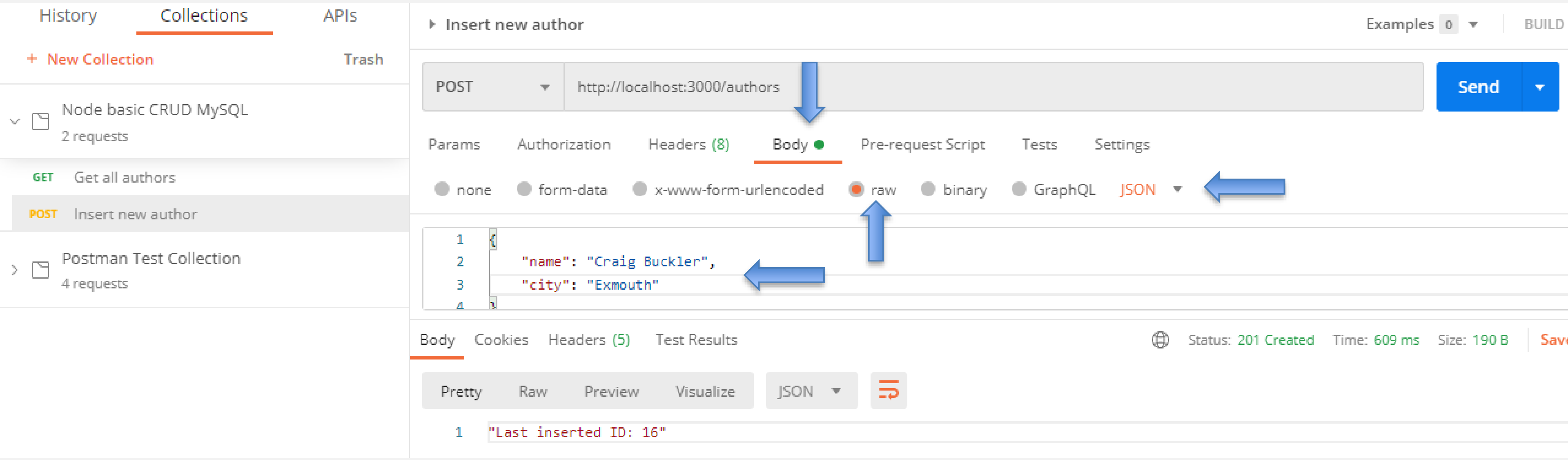
- Add a server response to **POST** requests at the same URL http://localhost:3000/authors
- It must **insert** a new author, with JSON data (provided in the request body data)
  - ➢ Server must respond with 201 status code (“Created”)
- HINTS:
  - ➢ Use the `request.method` string to retrieve the HTTP verb (GET, POST,…)
  - ➢ Print into the server console the `results` field of the callback:
    - ➢ property `results.affectedRows` gets the number of affected rows from an insert, update or delete statement
    - ➢ property `results.insertId` retrieve the insert ID of a table with auto increment primary key
  - ➢ Return to the client the new author ID (remember that in the DB this field is auto-incremental) using the `results.insertId` value

File **server.js**
(main application file)

```
…
//check the request HTTP verb
if (request.method == 'GET') { … }

else if (request.method == 'POST') {
     let data = '';
     // grab the data from the request body (it is a stream - data is sent in chunks)
     req.on('data', chunk => {
          data += chunk;
     })
     req.on('end', () => {// end of data retrieval
          // convert data (JSON string) into an object
          let newAuthor = JSON.parse(data || {});
          // send error message to client
          if ( newAuthor.name == undefined || newAuthor.city == undefined) {…}

          //query the BD with an INSERT statement
          connection.query('INSERT INTO authors SET ?', newAuthor, (err, results) => {
               if (err) … // send error message to client

               //send back to client a message with the ID of the new author
               response.writeHead(201, { 'Content-Type': 'application/json' });
               response.end(JSON.stringify({
                    "message":"author successfully inserted",
                    "newID": results.insertId}));
          }
     );
} …
```

Node + MySQL + JSON

- Test a POST request using Postman, in order to read the author data from the body request
  - ➢ See image below to check how to insert a JSON object into the HTTP POST request body



POST http://localhost:3000/authors

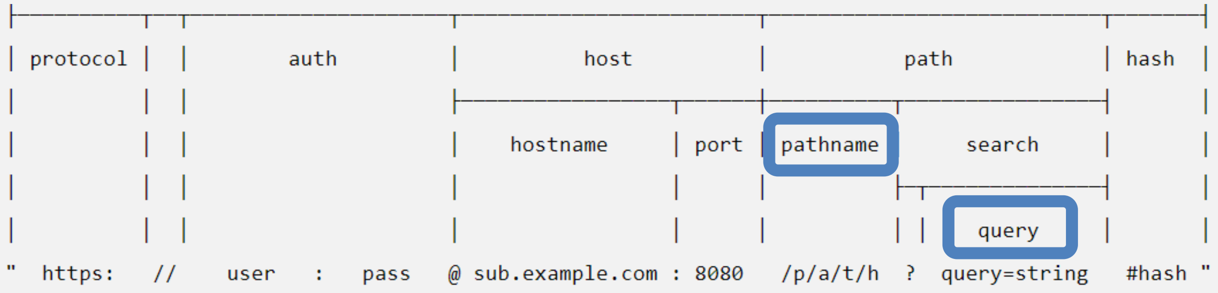
```
{
    "name": "Craig Buckler",
    "city": "Exmouth"
}
```

Status: 201 Created Time: 609 ms Size: 190 B

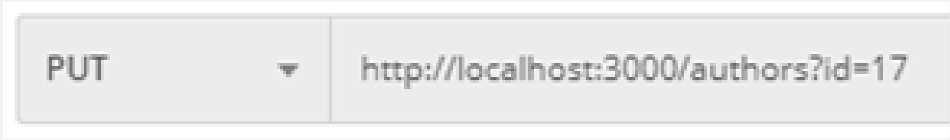
```
"Last inserted ID: 16"
```

Node + MySQL + JSON

- Now let's include na **UPDATE** query to the server
- Server database updates are usually performed via **PUT** HTTP requests by the client applications
- To know which row(s) on a table needs to be updated, the **URL** can include more information:
  - query parameters: "`/authors?id=`*`number`*"
  - pathname: "`/authors/id`"
  - ...

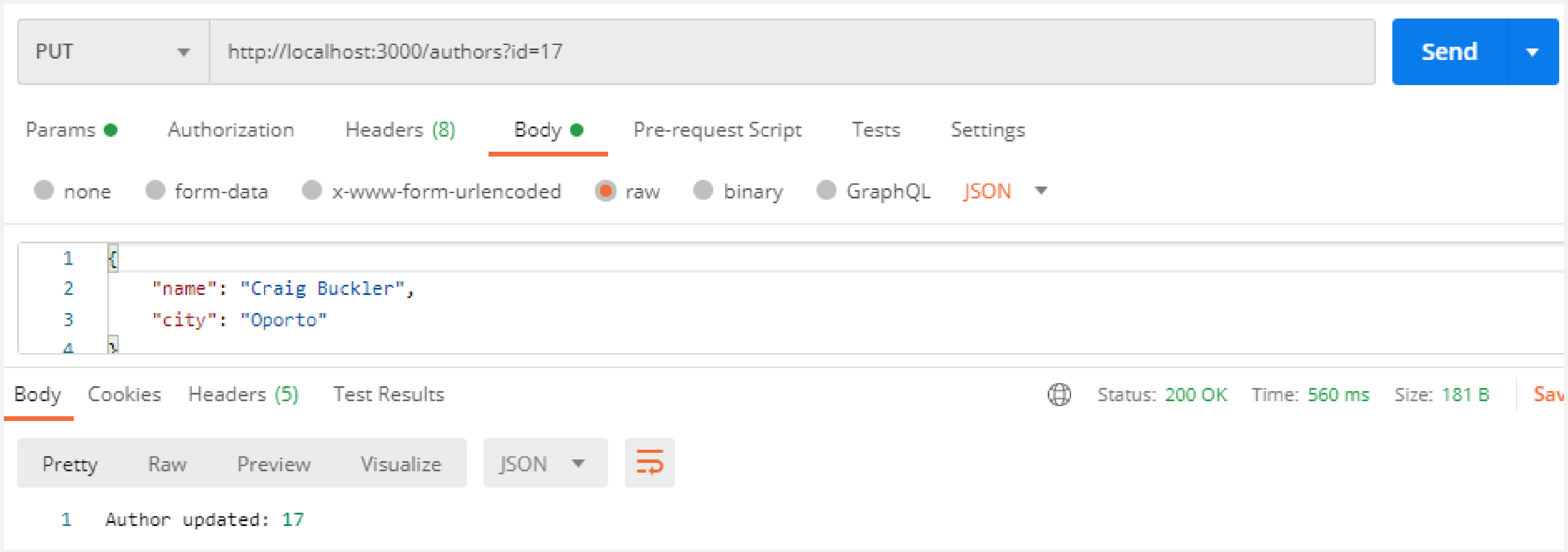

protocol
auth
host
path
hash
hostname
port
pathname
search
query
" https: // user : pass @ sub.example.com : 8080 /p/a/t/h ? query=string #hash "

Node + MySQL + JSON


PUT http://localhost:3000/authors?id=17

- Let's try with **query parameters**:
  - alter your server so that first it parses the request URL
  - the server only queries the database if the **URL pathname** is "`/authors`"
- Then add response for **PUT** requests
  - ➢ Get the `id` parameter of the URL query string, that must be "`?id=`*`number`*"
  - ➢ Get the new author parameters (name and city) from the HTTP request body (as in a POST request)
  - ➢ Query the database
    - ➢ `results.affectedRows`: number of selected rows to update (if 0, means that author ID does not exist in the database, therefore the server must respond with a 404 response)
    - ➢ `results.changedRows`: number of effectively updated rows
- Test it again with Postman

Node + MySQL + JSON



PUT http://localhost:3000/authors?id=17

Body: raw [✓ raw] JSON

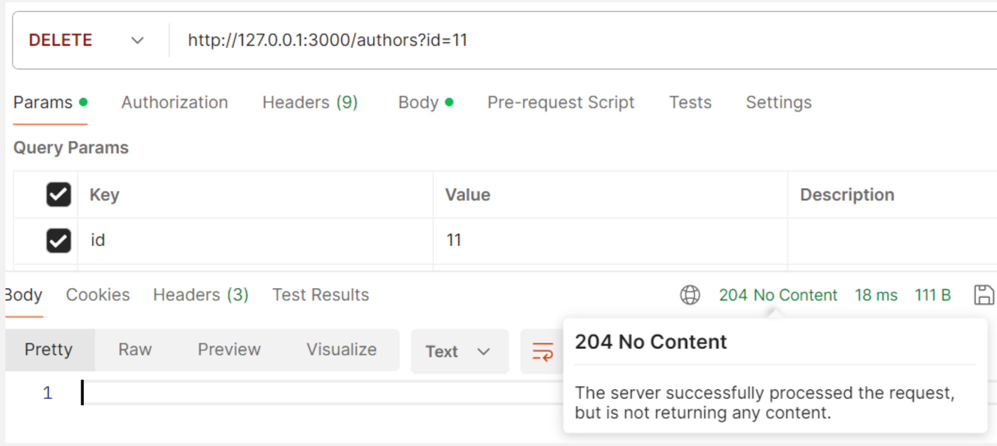
```
{
    "name": "Craig Buckler",
    "city": "Oporto"
}
```

Status: 200 OK Time: 560 ms Size: 181 B

```
Author updated: 17
```

Node + MySQL + JSON

- To **delete** one author, cliente should perform a **DELETE** HTTP request, providing the author ID in the URL query string
  - Get the `id` parameter of the URL query string, "`?id=`*`number`*", to know which author is to be deleted
- If sucessfull, the response status code should be 204 (no contente to retrieve to client)
  - `results.affectedRows`: number of selected rows to delete (if 0, means that author ID does not exist in the database, therefore the server must respond with a 404 status code)


DELETE http://127.0.0.1:3000/authors?id=11

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

	Key	Value	Description
✓	id	11	

Body Cookies Headers (3) Test Results — 204 No Content 18 ms 111 B

Pretty Raw Preview Visualize Text

204 No Content

The server successfully processed the request, but is not returning any content.