Chapter 1

RESTFUL Web Services with Node js: Basic

What Are RESTful Web Services?

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically web url links. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

- **Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.

- **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, but JSON is the most popular one.

- **Stateless interactions**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

  HTTP Methods

Method	Operation performed on server	Quality
GET	Read a resource.	Safe
PUT	Insert a new resource or update if the resource already exists.	Idempotent
POST	Insert a new resource. Also can be used to update an existing resource.	N/A
DELETE	Delete a resource.	Idempotent

A Safe operation is an operation that does not have any effect on the original value of the resource. A read operation only fetches data, so it is safe and does not modify the original data in any way. An Idempotent operation is an operation that gives the same result no matter how many times you perform it. For example, if you delete a user with id 10, the result will still be the same regardless of how many times it is performed.

Difference between PUT and POST

The key difference between PUT and POST is that PUT is idempotent while POST is not. No matter how many times you send a PUT request, the results will be same. POST is not an idempotent method. Making a POST multiple times may result in multiple resources getting created on the server.

Another difference is that, with PUT, you must always specify the complete URI of the resource. This implies that the client should be able to construct the URI of a resource even if it does not yet exist on the server. This is possible when it is the client's job to choose a unique name or ID for the resource, just like creating a user on the server requires the client to choose a user ID. If a client is not able to guess the complete URI of the resource, then you have no option but to use POST.

Request	**Operation**
PUT http://.../User/1	Insert a new user with **Userid=1** if it does not already exist, or else update the existing resource
POST http://.../User/	Insert a new person every time this request is made and generate a new **Userid**.
POST http://.../User/1	Update the existing person where **Userid =1**

From the above table, we can see that the PUT request will not modify or create more than one resource no matter how many times it is called. There is no difference between PUT and POST if the resource already exists, as both update the existing resource. The third request (POST http://.../User/) will create a new resource each time it is called.

Setting up the persistence storage source

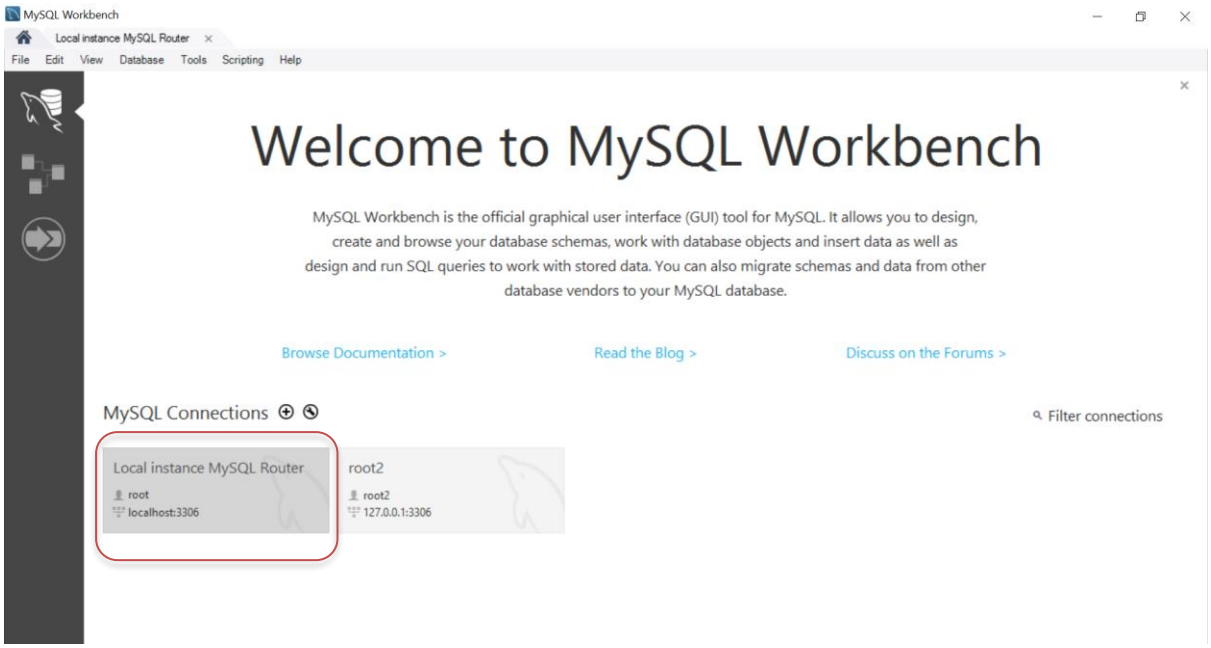
Now that we understand what a RESTful web service can do, let's proceed to create a RESTful web service.

We will create a web service that can handle create, read, update, delete (CRUD) operations on a user database table. We will use MySQL, an open source database management system to create the tables.

Let's create a table called **User** which contains the following fields:

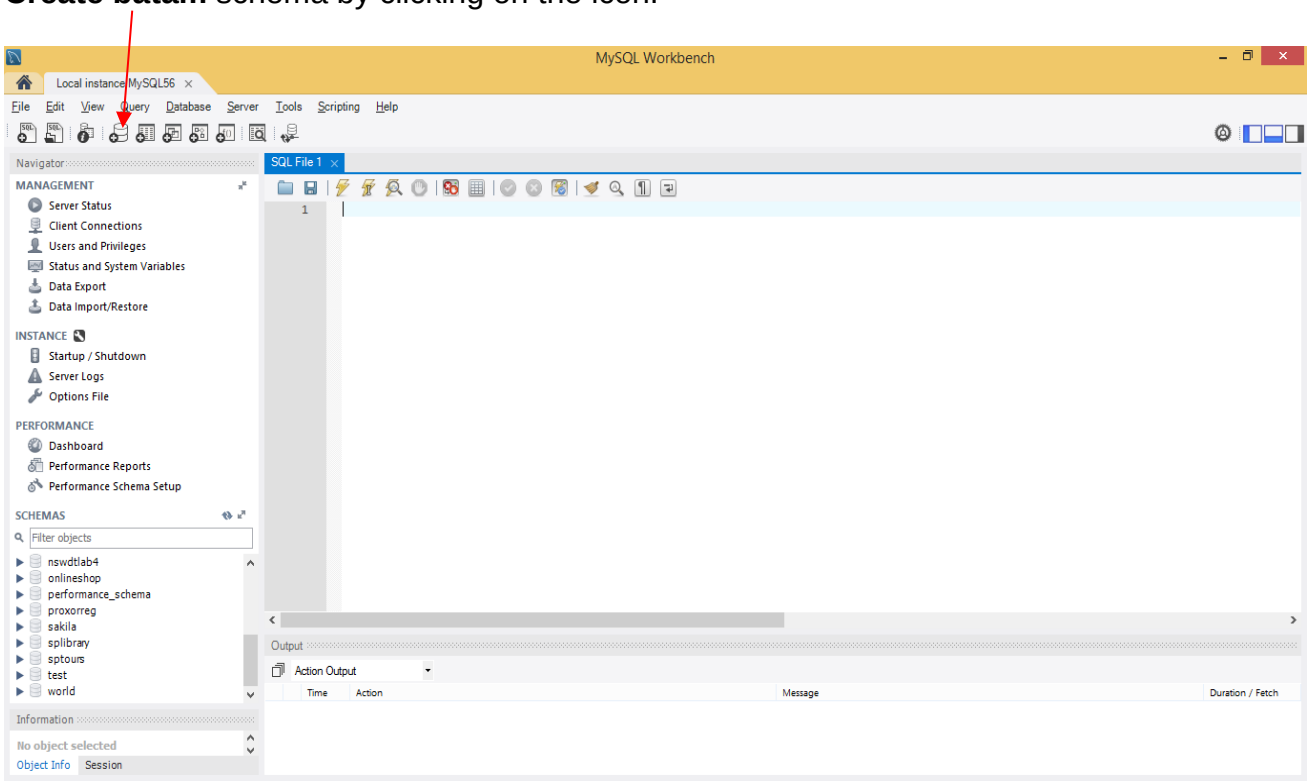
- Userid (Data Type: varchar) [Primary Key, Auto Increment]
- Username (Data Type: varchar)
- Password (Data Type: varchar)
- Email (Data Type: varchar)
- Role (Data Type: varchar)

Startup workbench and log in to MySQL with the default root account

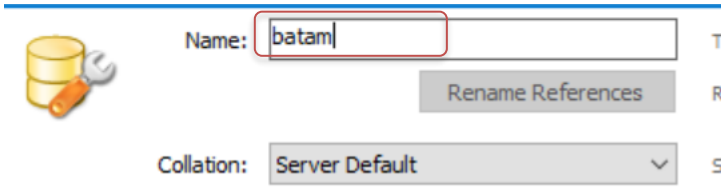


Enter the password for the root account according to what you setup during installation.

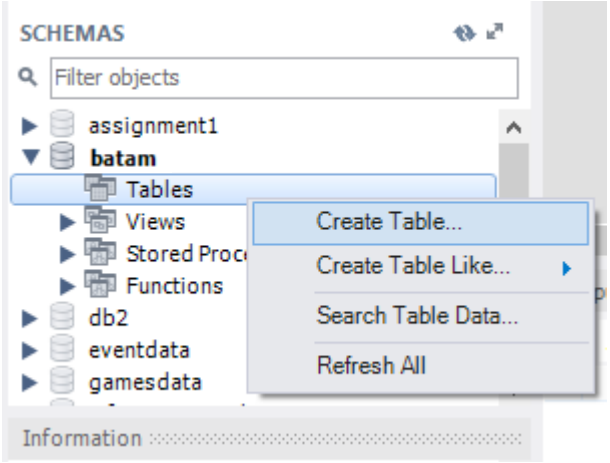
Create batam schema by clicking on the icon.



Enter the schema name as batam and click apply. Click apply again when the sql statement is show for creation of the schema

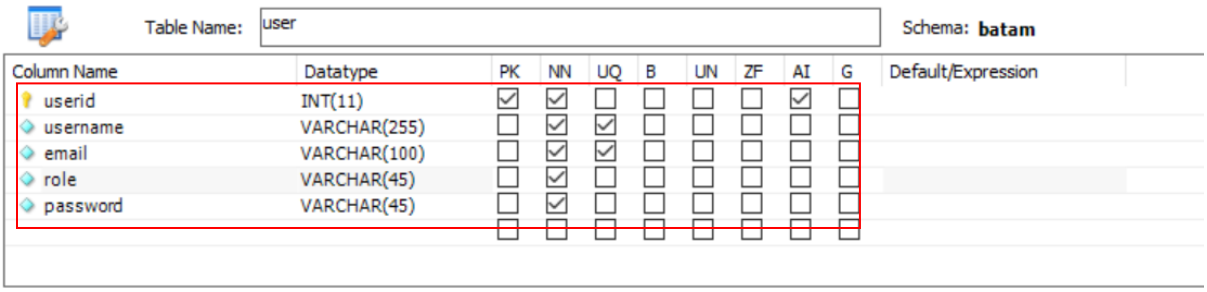


Left-click **batam** schema to open the menu, right click on Tables and choose **Create Table.**

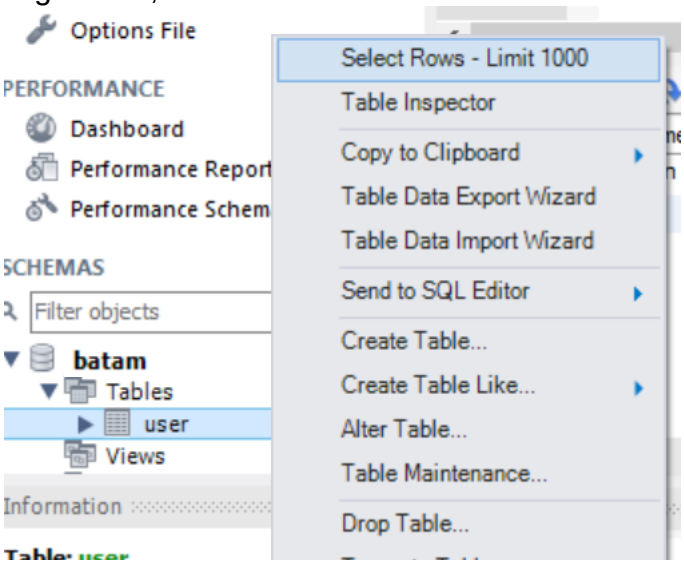


Enter Table Name (e.g. **user**), Columns Name and Datatype of the column, and then click **Apply Changes**. E.g. The Table Name is user, and the Columns Names are userid(Primary key, AUTOINCREMENT INTEGER), username (VARCHAR), email(VARCHAR), role(VARCHAR) and Password(VARCHAR).

Once done with the data entry, click Apply, review the script and apply the script to create the table.



Right click, then left click on select Rows



Enter the 2 records in the table and click apply once done

userid	username	email	role	password
1	Samsudin	samsudin@bmail.com	user	abc123
2	hidayat	hidayat@imail.com	admin	a12112x
NULL	NULL	NULL	NULL	NULL

Now we have the necessary data and table in the MySql Database.