MySQL
ORACLE

Building SQL-Free Applications
with MySQL REST Service

Frédéric Descamps
Community Manager
Oracle MySQL
preFOSDEM MySQL Belgian Days - January 2026

Who am I ?

about.me/lefred

Frédéric Descamps

- 𝕏 *@lefred*
- *@lefredbe.bsky.social*
- *@lefred@fosstodon.org*
- [matrix] *@lefred14:matrix.org*
- *MySQL Evangelist*
- *using MySQL since version 3.20*
- *devops believer*
- *living in* 🇧🇪
- *https://lefred.be*

MySQL REST Service

Agenda

Agenda

- *About MySQL REST Service (MRS)*
- *Installation of MySQL Labs with MRS*
- *Installation of MySQL Shell for Visual Studio Code*
- *Loading sample database*
- *Building RESTful APIs with MRS*
  - *with MySQL Shell for VS Code*
  - *with SQL in MySQL Shell*
- *Using the Python SDK*
- *Q&A Session*

MySQL REST Service

About

MySQL REST Service (MRS)

Fast, Secure HTTPS Access for MySQL Data

MRS is built on the concepts and architecture of Oracle REST Data Services (ORDS)

It's available in three different deployment options:

- *as middleware using MySQL Router*
- *as part of MySQL HeatWave*
- *as a component to run in the MySQL Server (Lab release)*

MySQL REST Service (MRS)

Features Overview

RESTful Web Services

- *auto REST for tables, view, procedures and functions*
- *{JSON} responses with paged results*
- *developer support (GUI, CLI, API)*
- *support for popular OAuth2 services*

Powerful REST Data Mapping

- *full SQL support for joins and nested JSON REST endpoints*
- *Visual Data Mapping Editor*
  - *build complex JSON structures with a flew clicks*
- *SQL & SDK interface preview*

MySQL REST Service (MRS)

Features Overview (2)

Full SQL Support & SDK API

- *fully manageable through SQL*
- *tailored SDK for all RESTful Endpoints*
- *popular, Prisma-like API, like prototyping*

TypeScript SDK API whith live prototyping of REST queries

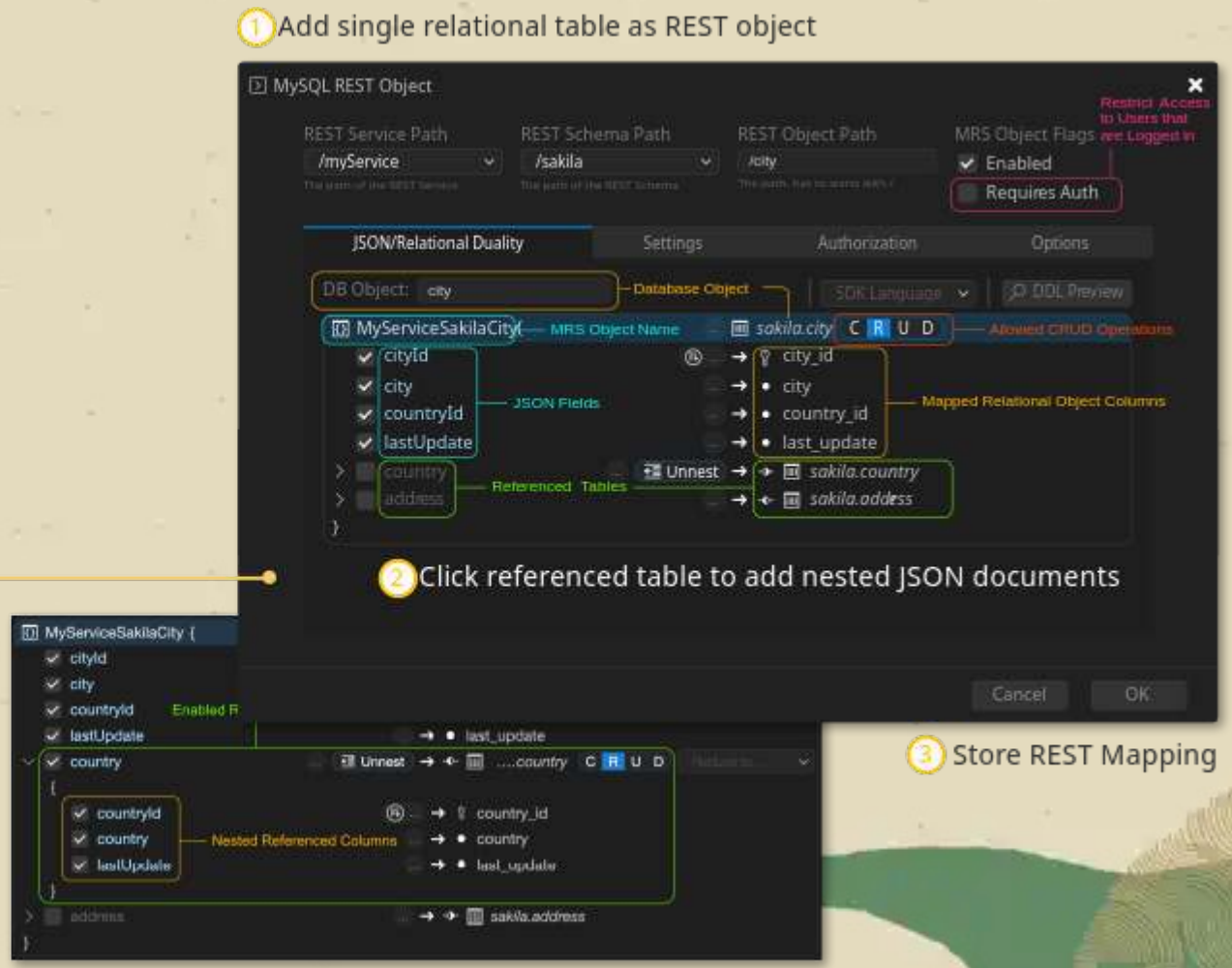
Fully manageable through SQL

MySQL REST Service (MRS)

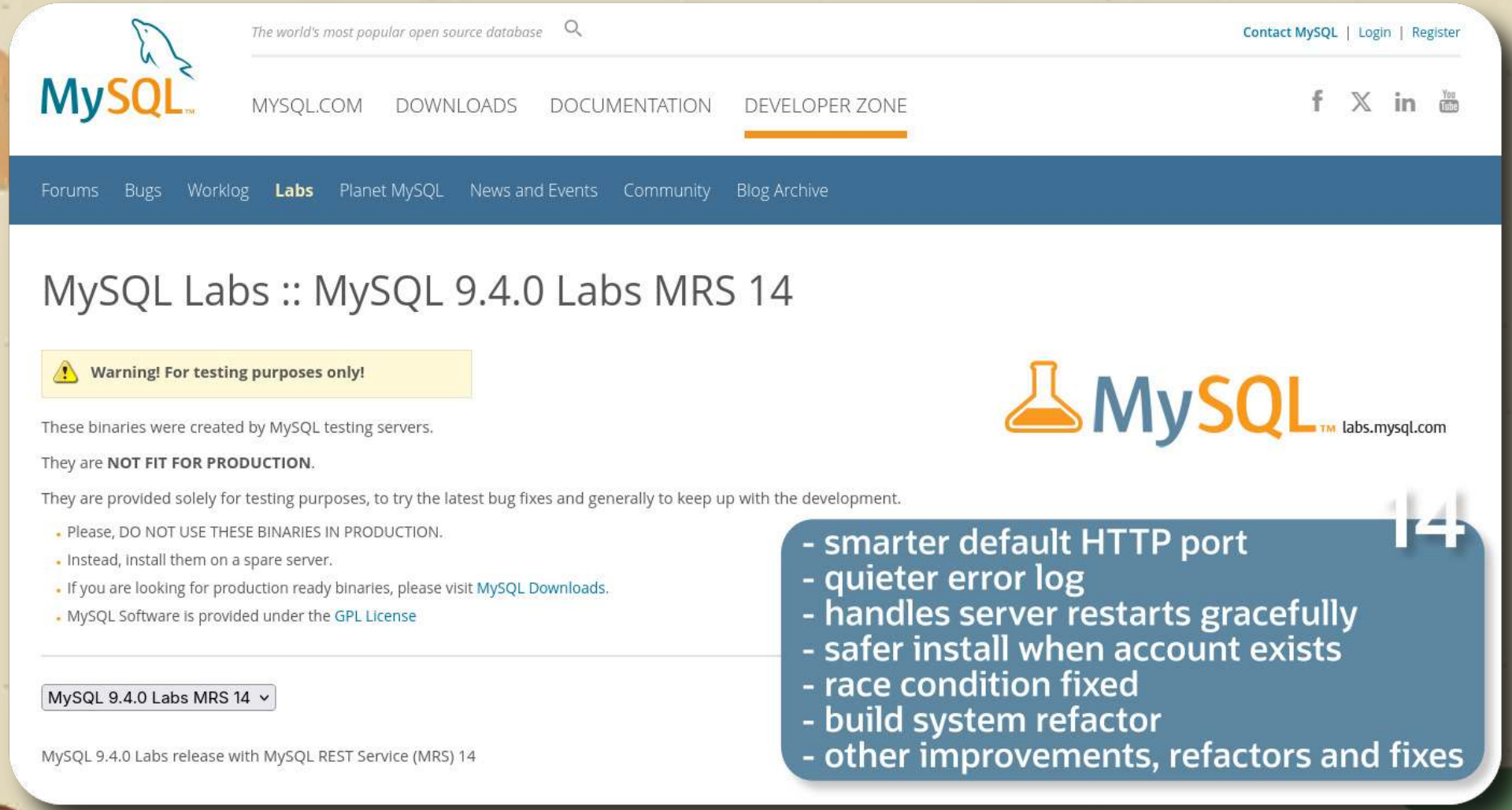
Visual REST Mapping View Editor

Intuitive WYSIWYG Editor to design REST Mappings

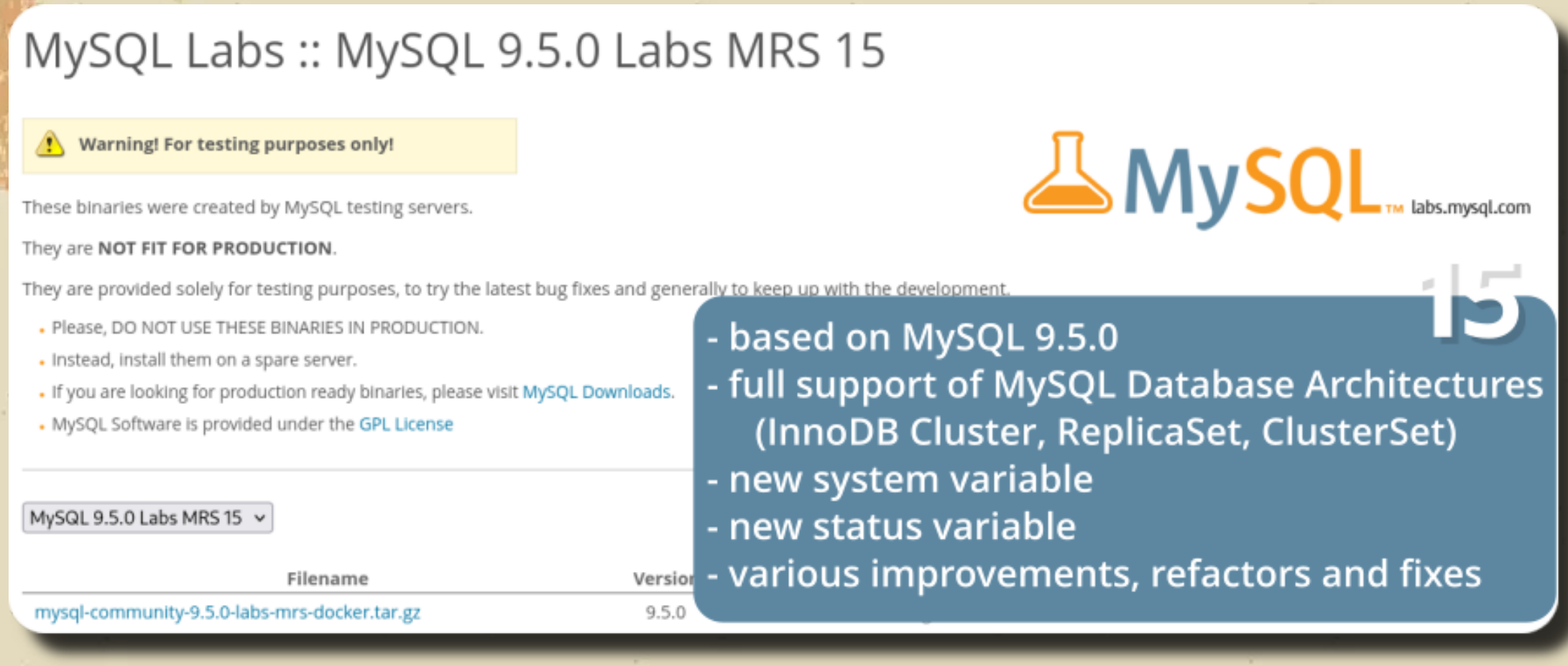
- *creation of complex mappings with a few clicks*
- *automatic database schema analysis*
- *SQL Preview*



MySQL REST Service is part of the Server

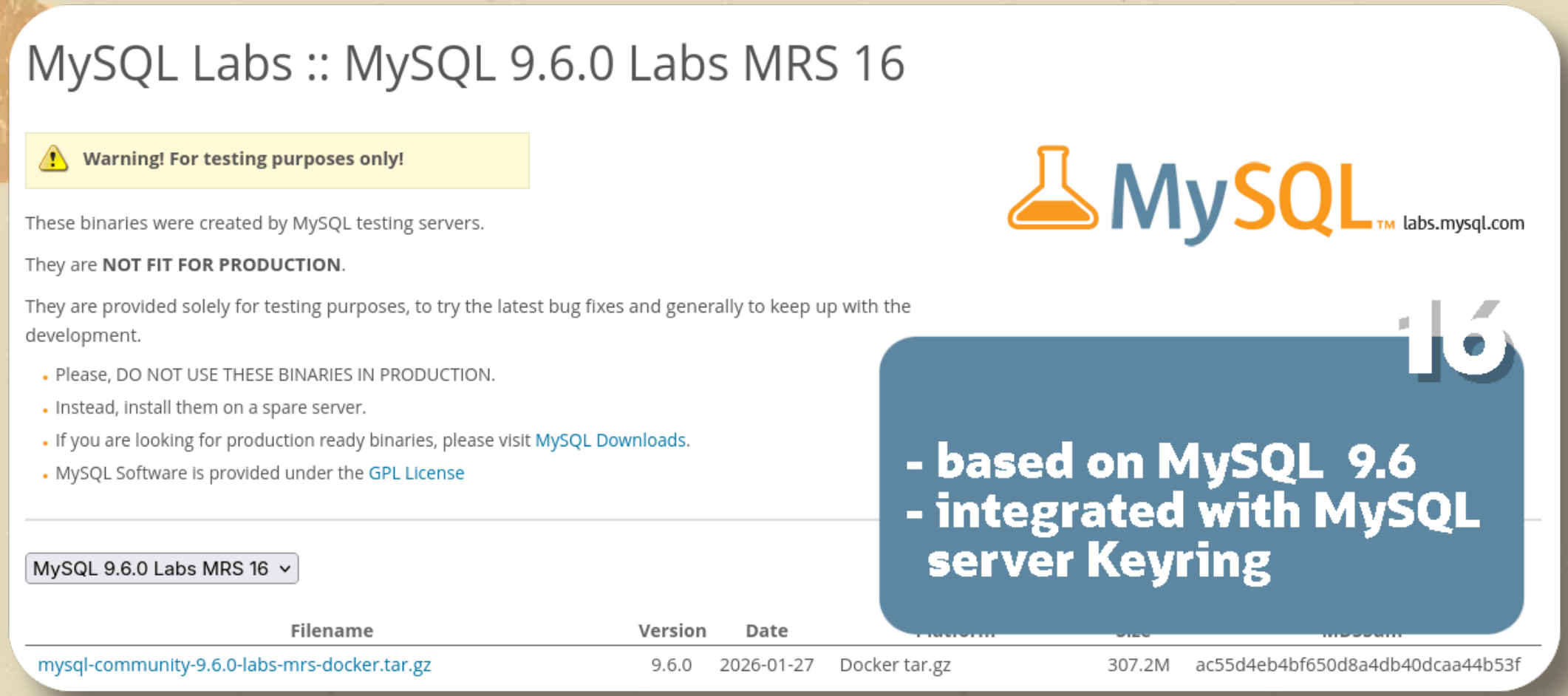


MySQL REST Service is part of the Server



https://blogs.oracle.com/mysql/post/uplevel-the-mysql-rest-service

MySQL REST Service is part of the Server



MySQL Server & MySQL Shell

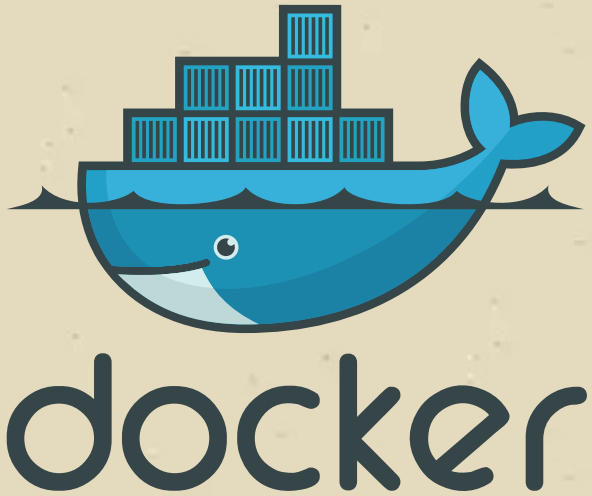
Installation

Installation: MySQL Server

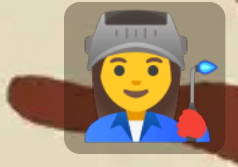
Demo using MySQL Labs 9.5.0 with MRS included that you can download from:

https://labs.mysql.com/

And we will use docker in the examples but you can install it natively on your OS as well.



Installation: MySQL Server (using docker)



```
$ docker load --input mysql-community-9.6.0-labs-mrs-docker.tar.gz

$ docker image list
REPOSITORY                      TAG          IMAGE ID      CREATED        SIZE
cont[...]ql/community-server    9.6-amd64    3cbcdeb1a24d  5 hours ago    1.1 GB
```

MySQL Server Keyring - NEW

The MRS component is now (since 9.6.0) using the MySQL Server Keyring to store passwords of the internal accounts used by MRS.

We need then to configure the keyring manifest when starting the container.

We use **`keyring-file`**.

mysqld.my

```
{
    "components": "file://component_keyring_file"
}
```

MySQL Server Keyring (2) - NEW

component_keyring_file.cnf

```
{
  "path": "/var/lib/mysql-keyring/component_keyring_file",
  "read_only": false
}
```

And we create the directory on the host machine:

```
$ mkdir -p ./mysql-keyring
$ chmod 777 ./mysql-keyring
```

Lauching the MySQL Server in Docker

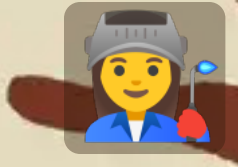
```
$ docker run -d --name mrs -e MYSQL_ROOT_PASSWORD=F0sd3M \
 -p 3306:3306 -p 33060:33060 -p 33061:33061 \
 --mount type=bind,src="mysqld.my",dst=/usr/sbin/mysqld.my,ro \
 --mount type=bind,src="component_keyring_file.cnf",dst=/usr/lib64/mysql/plugin/component_keyring_file.cnf,ro \
 --mount type=bind,src="mysql-keyring",dst=/var/lib/mysql-keyring \
 container-registry.oracle.com/mysql/community-server:9.6-amd64
```

Lauching the MySQL Server in Docker

```
$ docker run -d --name mrs -e MYSQL_ROOT_PASSWORD=F0sd3M \
 -p 3306:3306 -p 33060:33060 -p 33061:33061 \
 --mount type=bind,src="mysqld.my",dst=/usr/sbin/mysqld.my,ro \
 --mount type=bind,src="component_keyring_file.cnf",dst=/usr/lib64/mysql/plugin/component_keyring_file.cnf,ro \
 --mount type=bind,src="mysql-keyring",dst=/var/lib/mysql-keyring \
 container-registry.oracle.com/mysql/community-server:9.6-amd64
```

```
$ printf "%-12s %-10s %-10s %-20s %-12s\n" ID NAME COMMAND CREATED SIZE
  docker ps -sa --format \
 "{{printf \"%-12s %-10s %-10s %-20s %-12s\" .ID .Names .Command .RunningFor .Size}}"
ID           NAME       COMMAND    CREATED              SIZE
68d33d0dc412 mrs        mysqld     2 minutes ago        215MB (virtual 1.31GB)
```

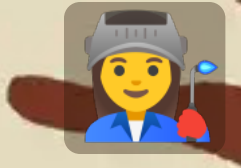
Installation: MySQL Server (using docker) (2)



Now we need to create a user that will be able to connect from your machine to to database server in the container:

```
$ docker exec -it mrs mysql -uroot -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 9.6.0-labs-mrs-16 MySQL Community Server - GPL

mysql> create user fosdem identified by 'F0sd3M';
Query OK, 0 rows affected (0.018 sec)

mysql> grant all privileges on *.* to fosdem with grant option;
Query OK, 0 rows affected (0.014 sec)
```

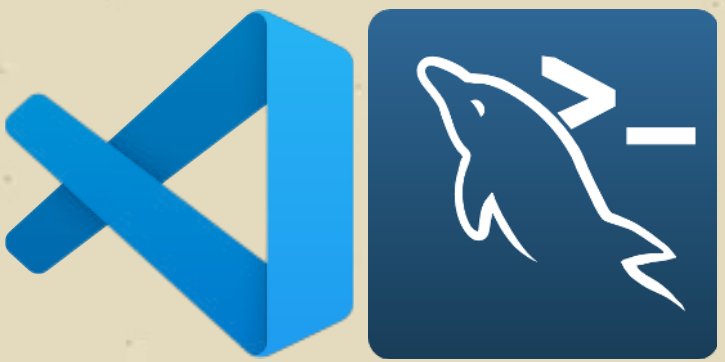
Installation: MySQL Shell for Visual Studio Code



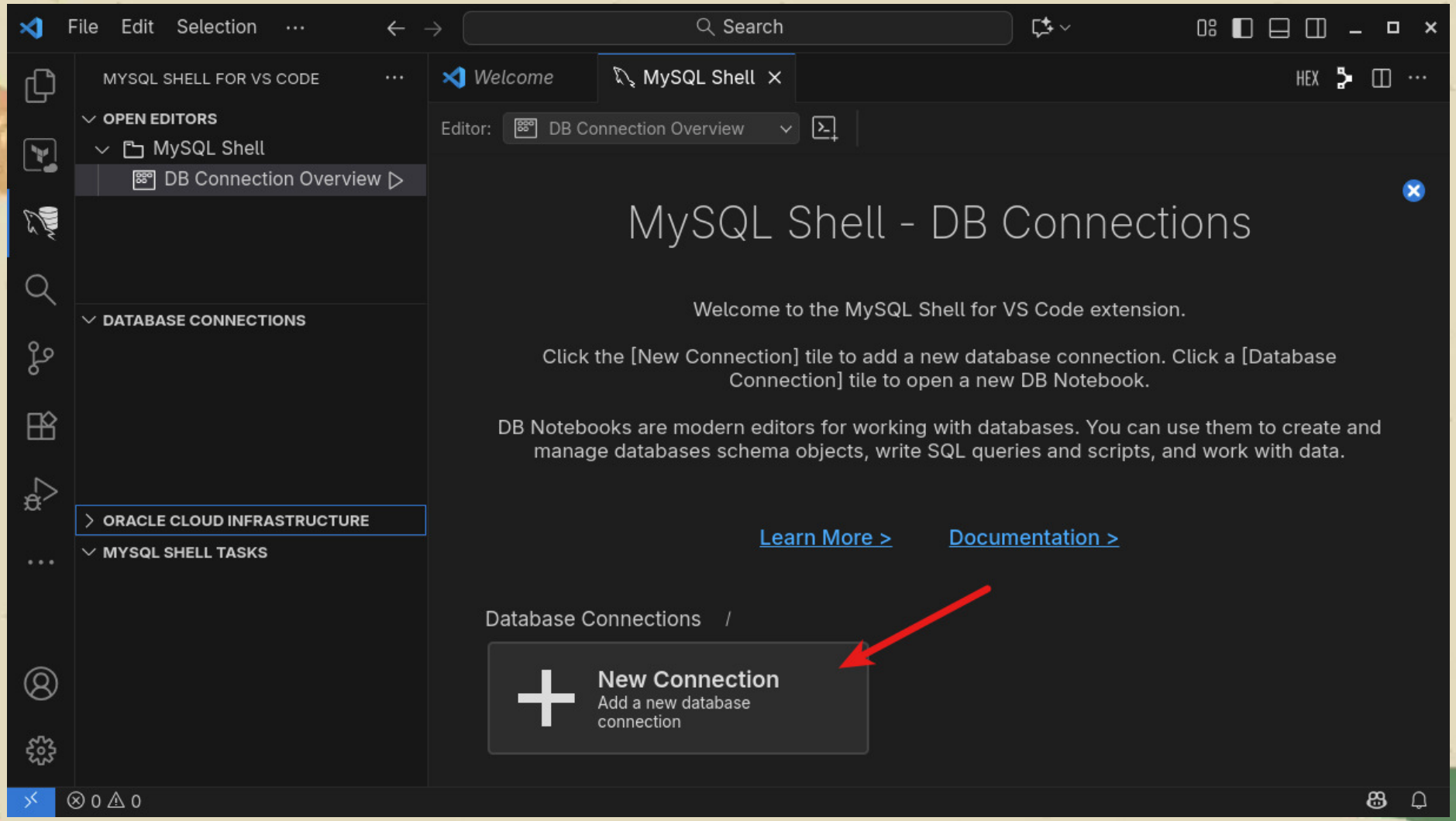
You need to install Visual Studio Code and then the MySQL Shell extension from the marketplace.

Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter:

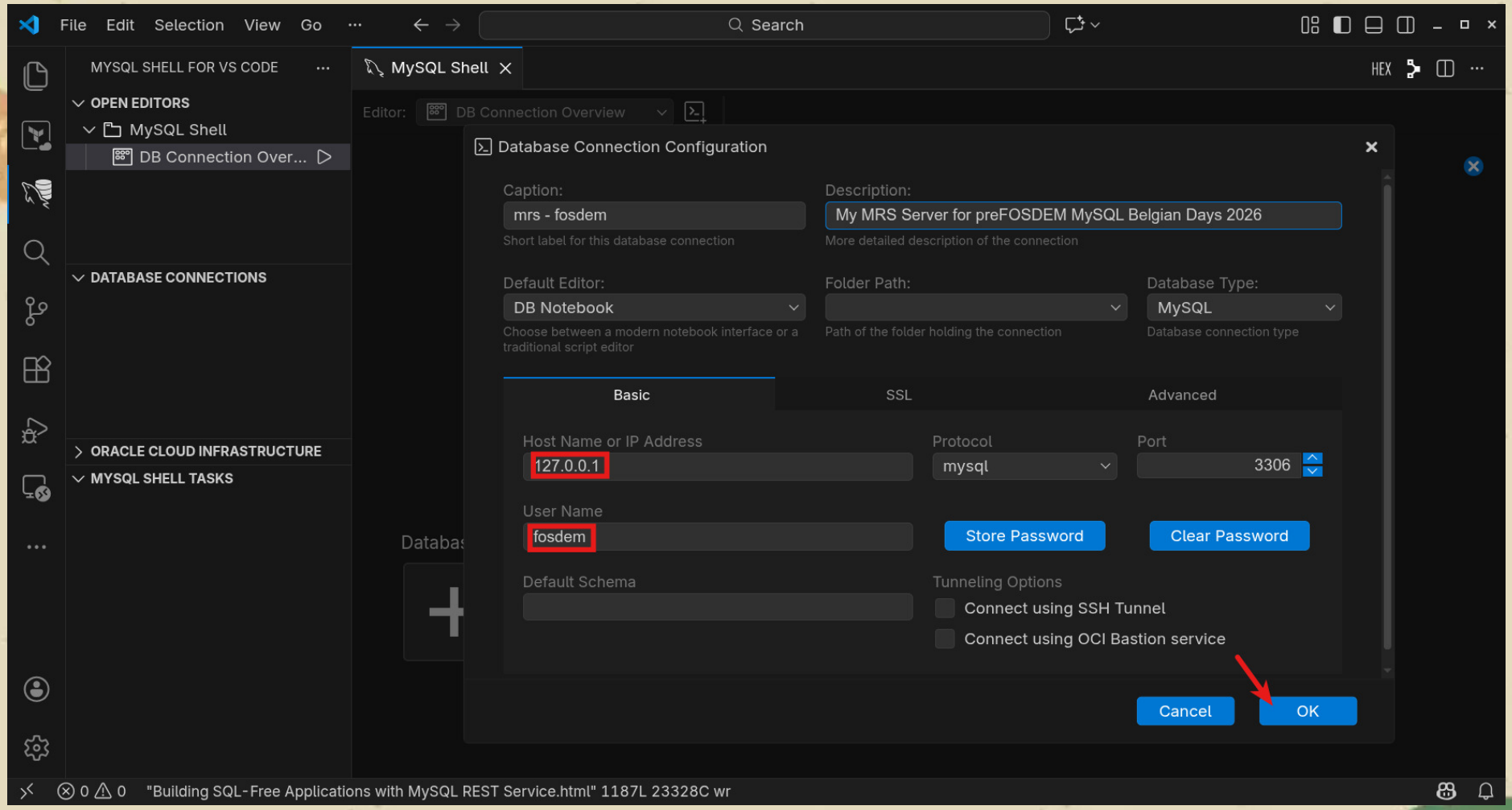
```
ext install Oracle.mysql-shell-for-vs-code
```



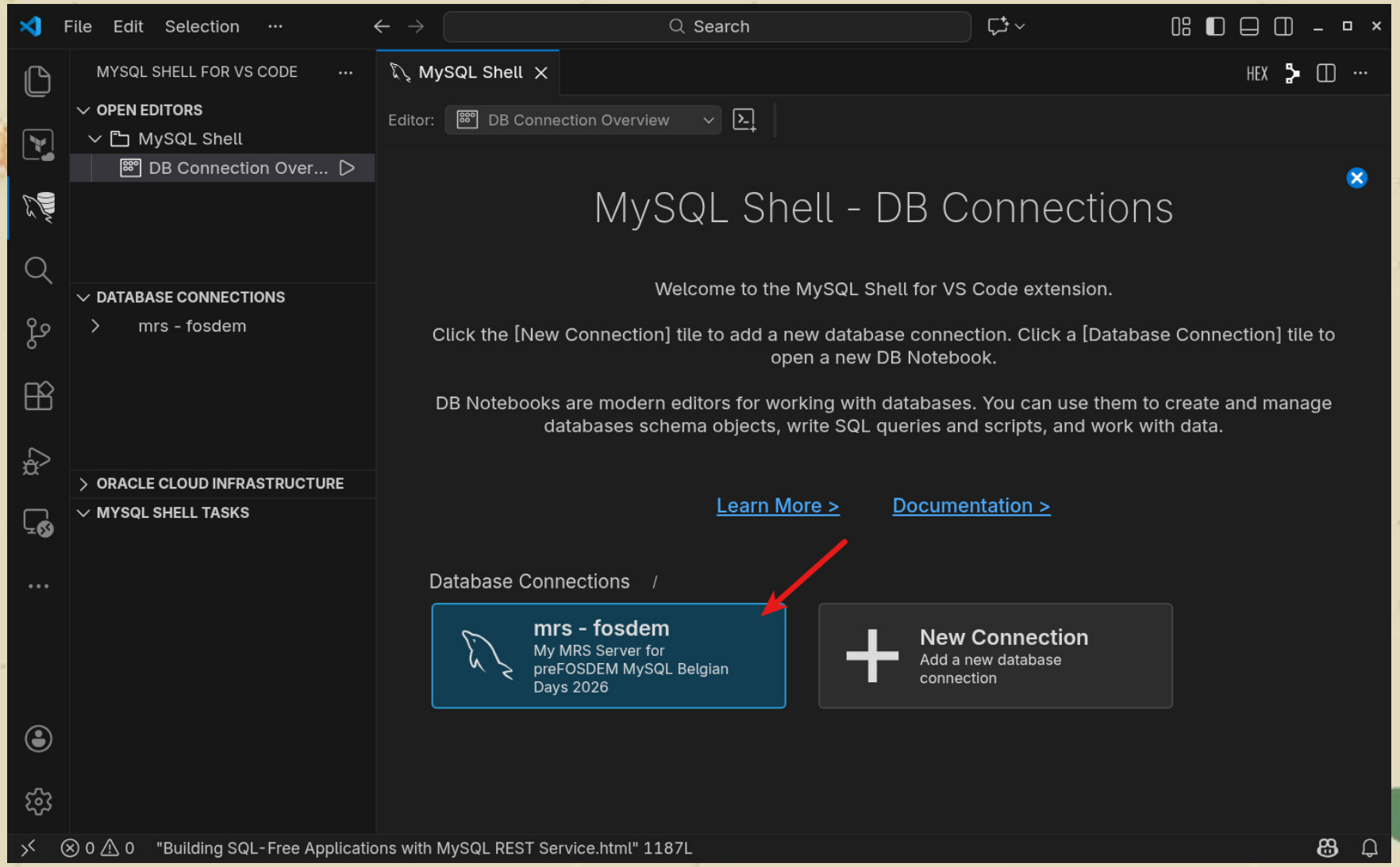
Connecting to MySQL Server



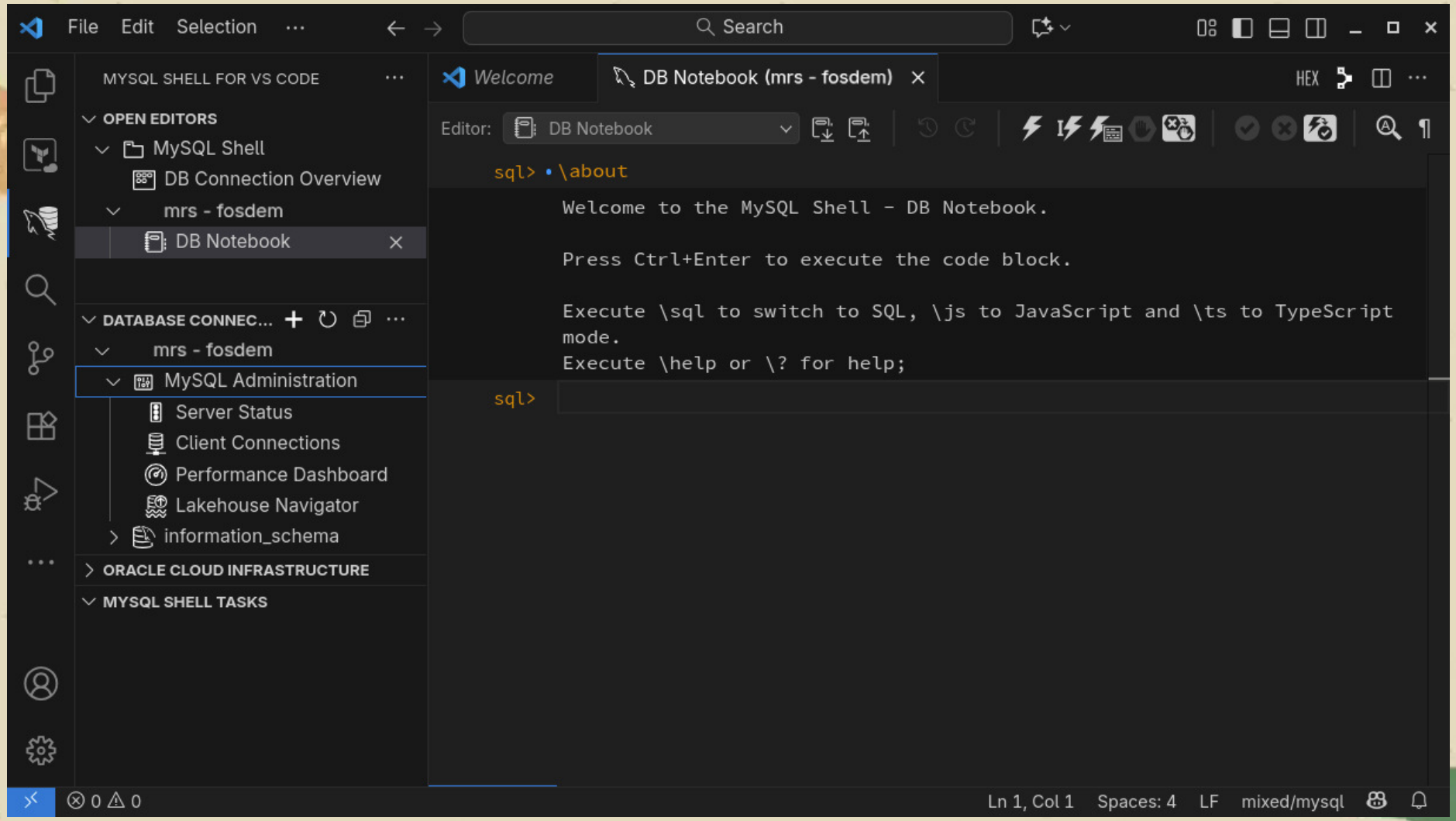
Connecting to MySQL Server



Connecting to MySQL Server



Connecting to MySQL Server



Connecting to MySQL Server (2)

If we prefer to use MySQL Shell from command line, we can do it like this if we have it installed (from https://dev.mysql.com/downloads/shell/):

```
$ mysqlsh mysql://fosdem@127.0.0.1:3306
Please provide the password for 'fosdem@127.0.0.1:3306': *******
Save password for 'fosdem@127.0.0.1:3306'? [Y]es/[N]o/Ne[v]er (default No): yes
MySQL Shell 9.6.0

Type '\help' or '\?' for help; '\quit' to exit.
Creating a Classic session to 'fosdem@127.0.0.1:3306'
Fetching global names for auto-completion... Press ^C to stop.
Your MySQL connection id is 12
Server version: 9.6.0-labs-mrs-16 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
 MySQL>
```

Sample Database

Sakila

Loading Sample Database: Sakila

We will use the Sakila sample database that you can download from:

https://downloads.mysql.com/docs/sakila-db.zip

The Sakila sample database was initially developed by Mike Hillyer, a former member of the MySQL AB documentation team. It is intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so forth. The Sakila sample database also serves to highlight features of MySQL such as Views, Stored Procedures, and Triggers.

Loading Sample Database: Sakila

We will use the Sakila sample database that you can download from:

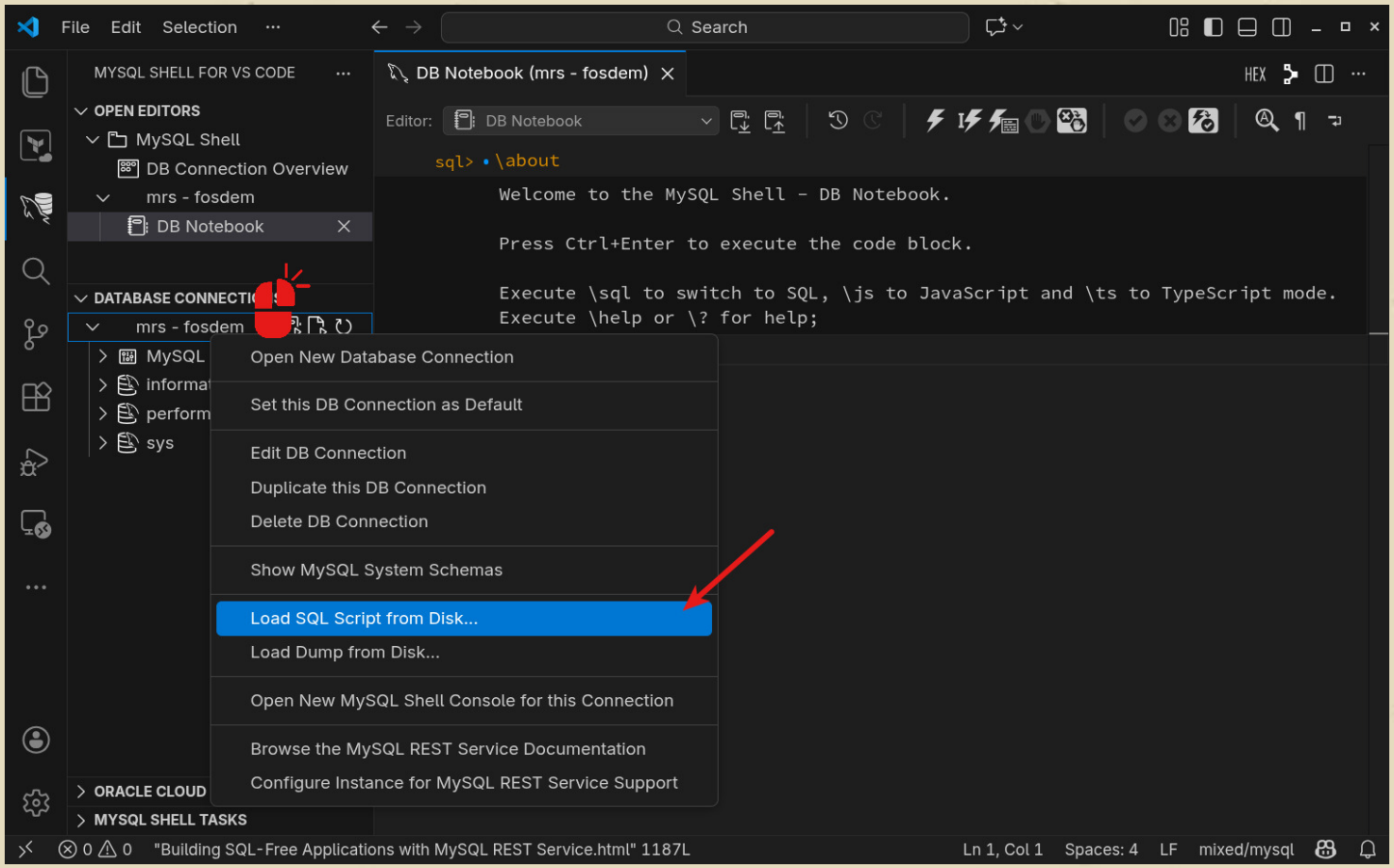
https://downloads.mysql.com/docs/sakila-db.zip

The Sakila sample database was initially developed by Mike Hillyer, a former member of the MySQL AB documentation team. It is intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so forth. The Sakila sample database also serves to highlight features of MySQL such as Views, Stored Procedures, and Triggers.

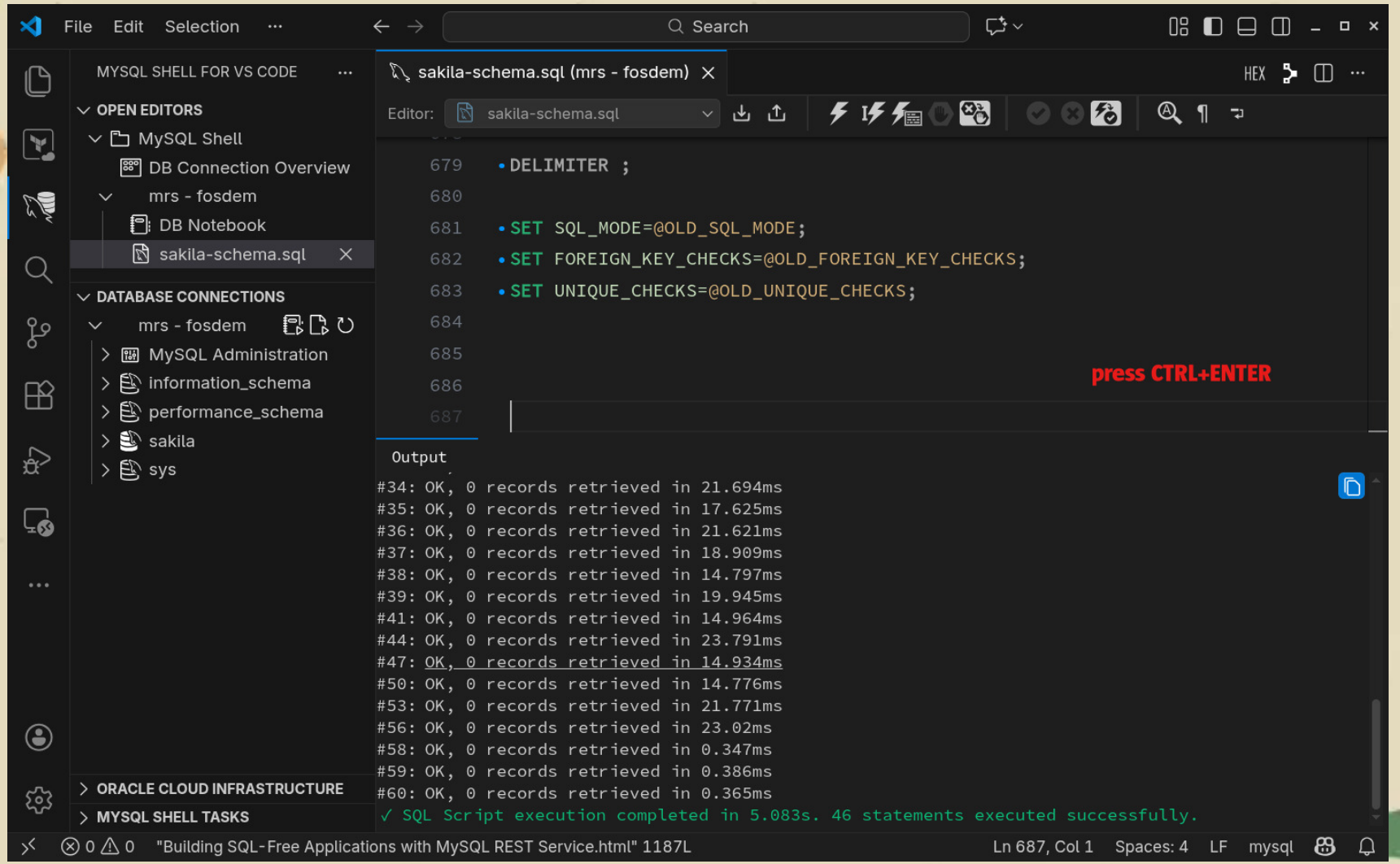
Download it and unzip it!

Loading Sample Database: Sakila (2)

In MySQL Shell for VS Code, open the `sakila-schema.sql` *file:*

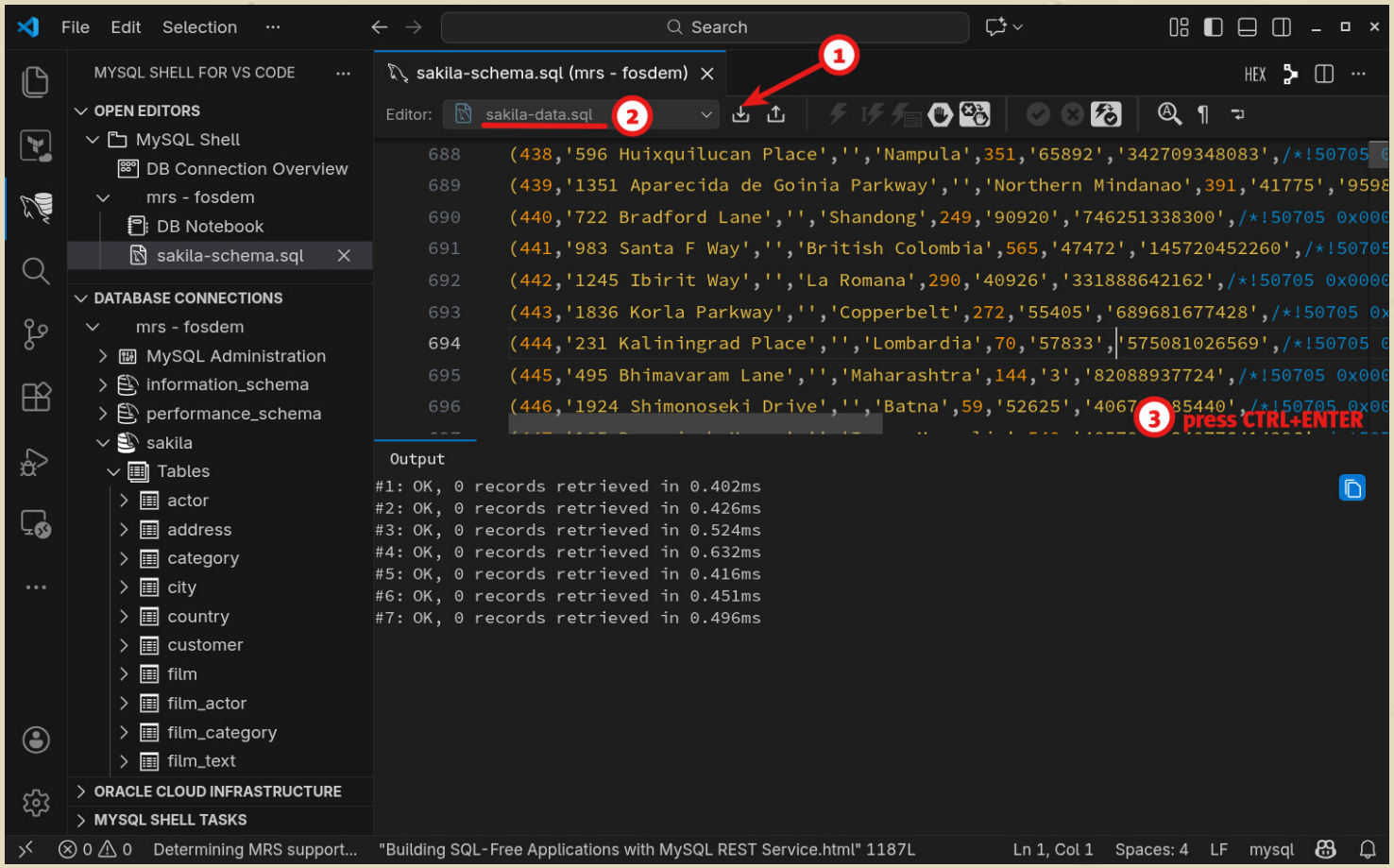


Loading Sample Database: Sakila (2)

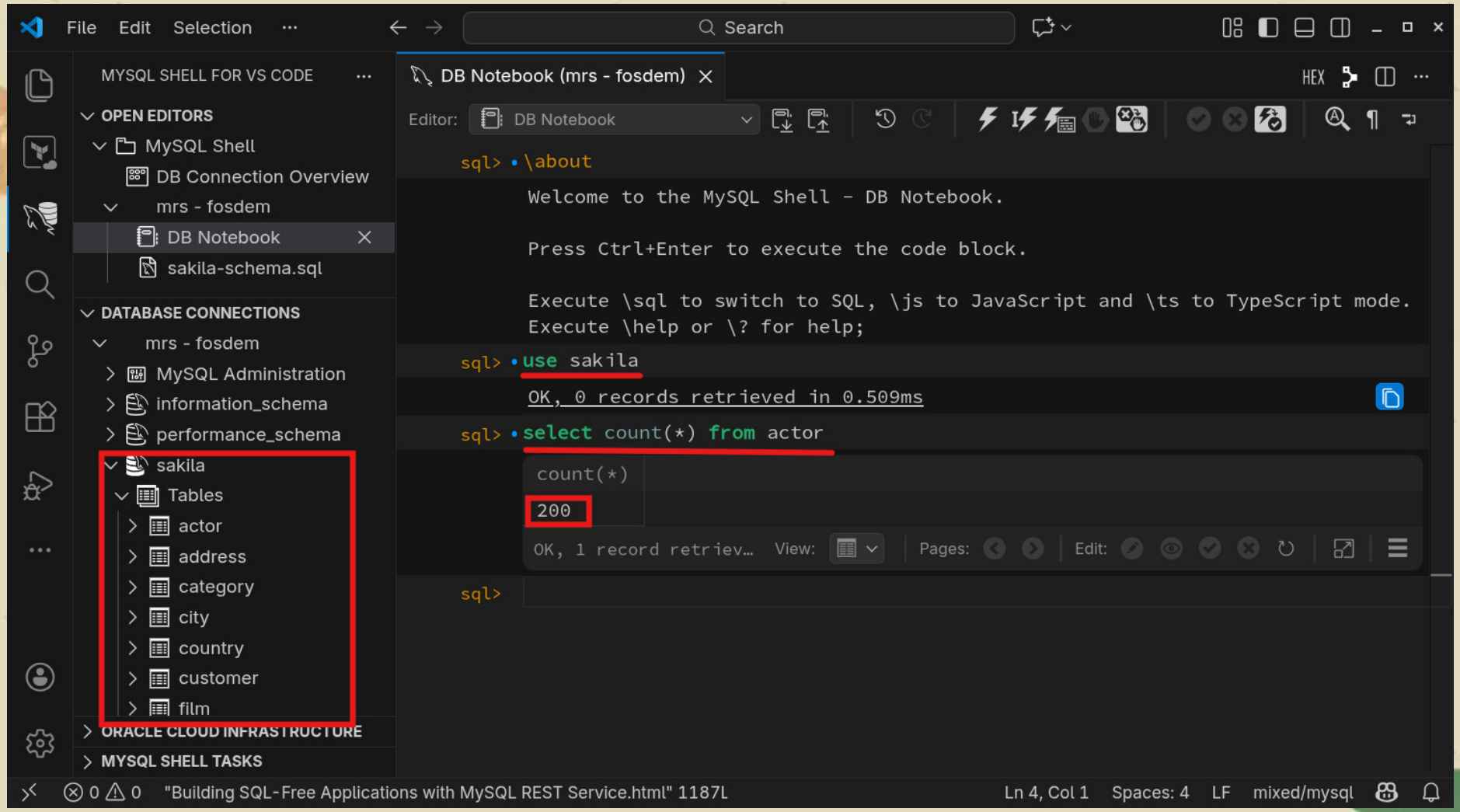


Loading Sample Database: Sakila (2)

Then load the `sakila-data.sql` *file:*



Loading Sample Database: Sakila (2)



Building RESTful APIs with MRS

Preparing the Server

Creating the REST metadata

The very first step for the MySQL REST Service component to work is to create the metadata schema.

We perform this operation even before loading the component.

Creating the REST metadata

The very first step for the MySQL REST Service component to work is to create the metadata schema.

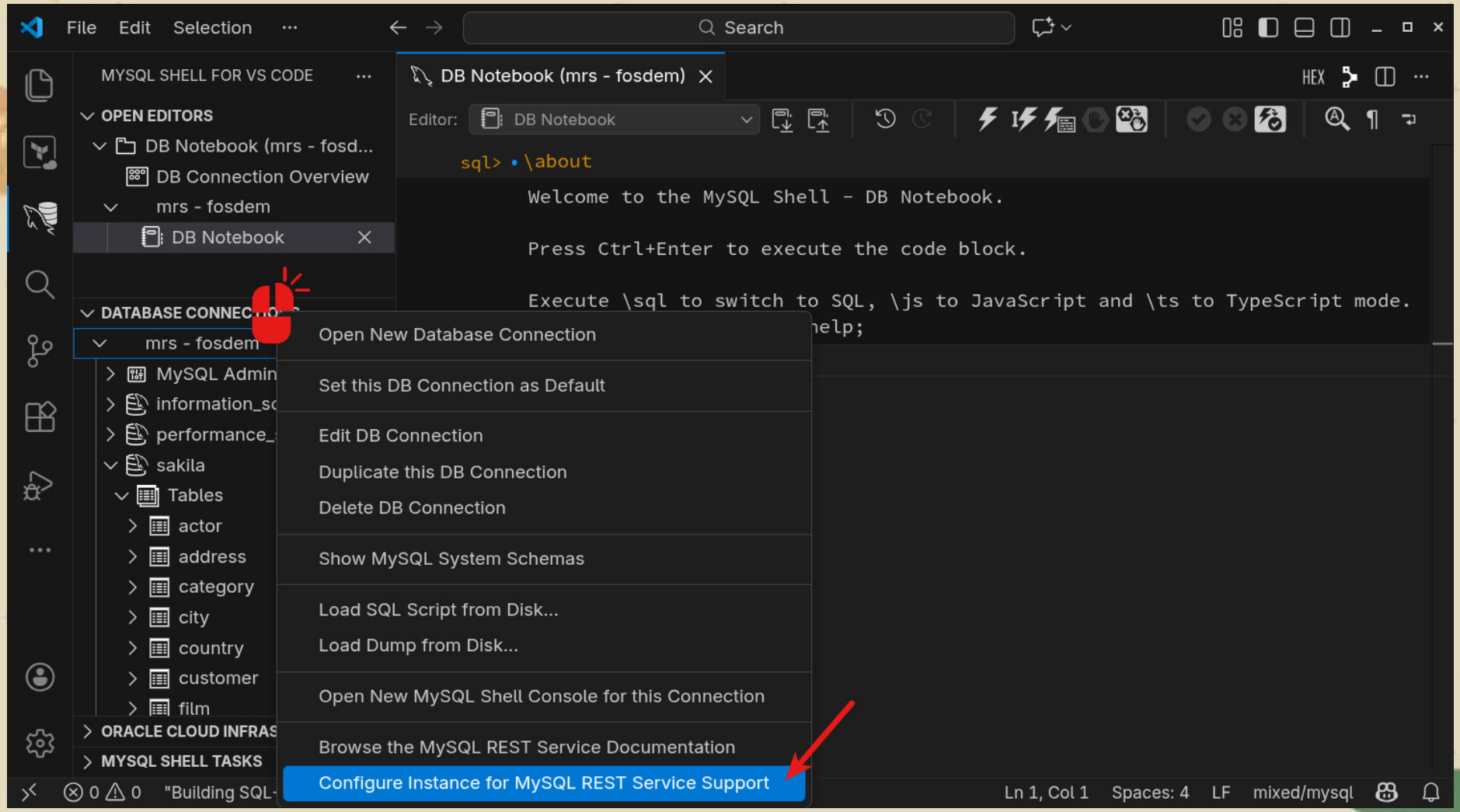
We perform this operation even before loading the component.

This operation can be done in SQL like this but only using MySQL Shell:

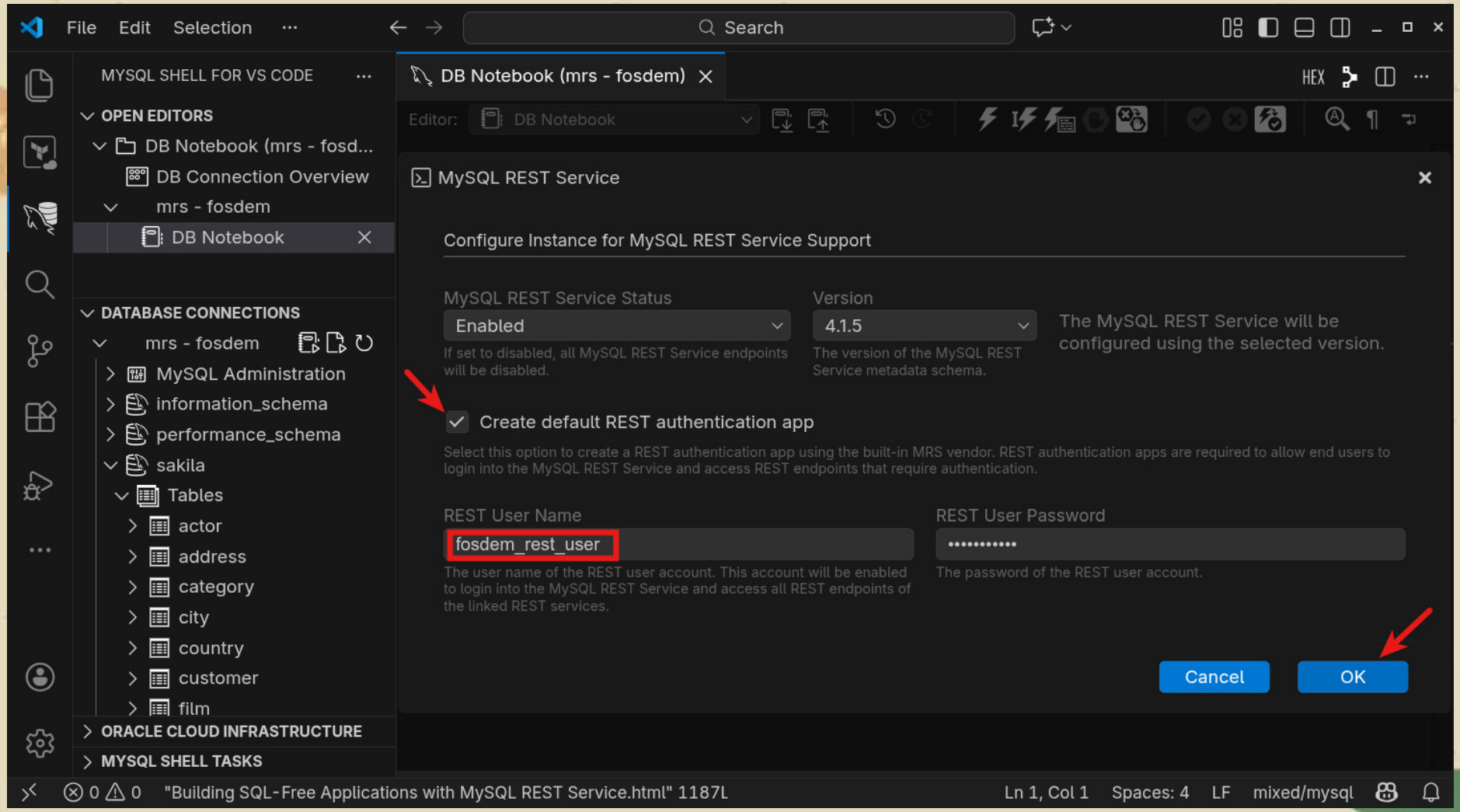
```
SQL> CONFIGURE REST METADATA;
```

But we will do it using MySQL Shell for VS Code as it automates more things for us.

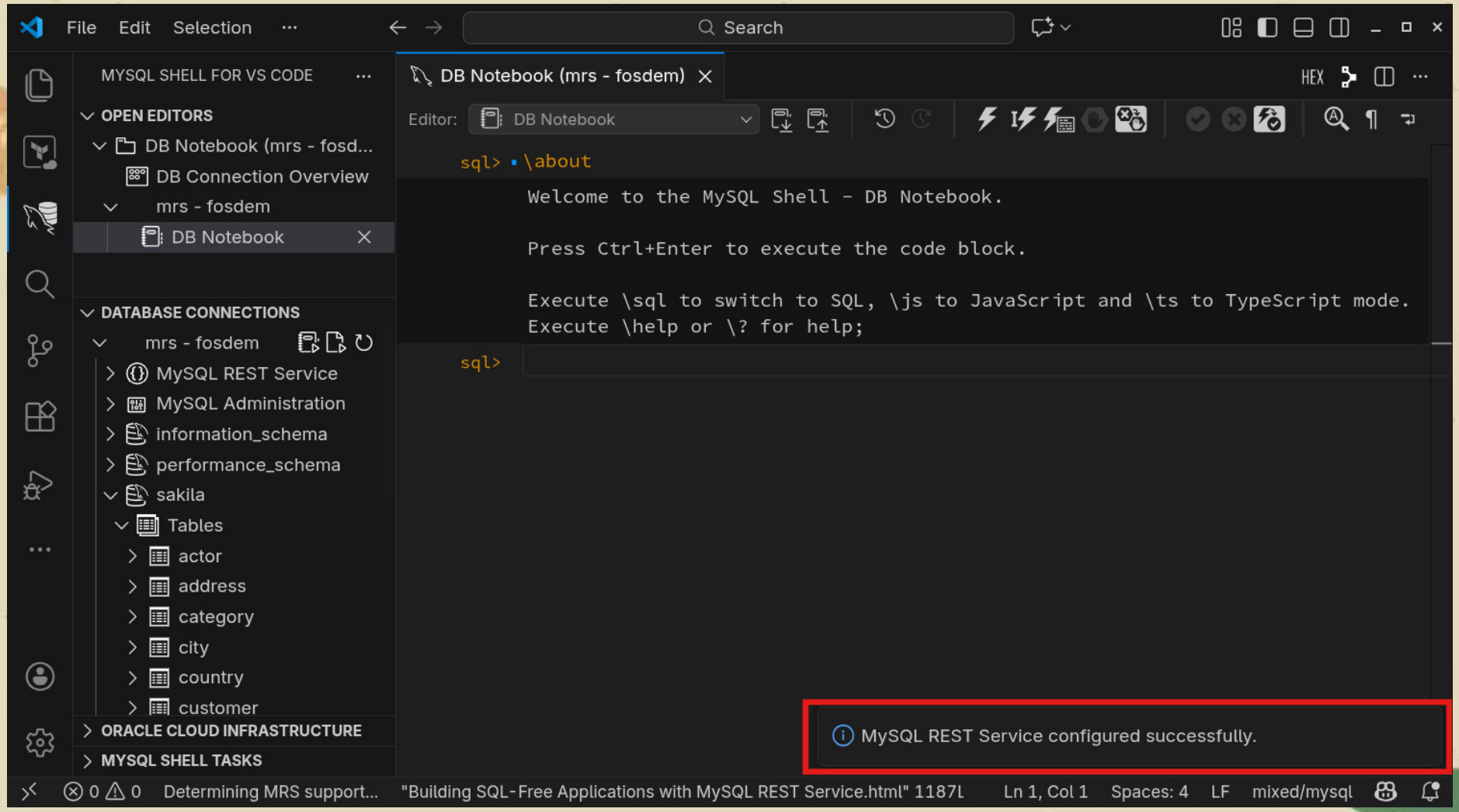
Creating the REST metadata - VS Code



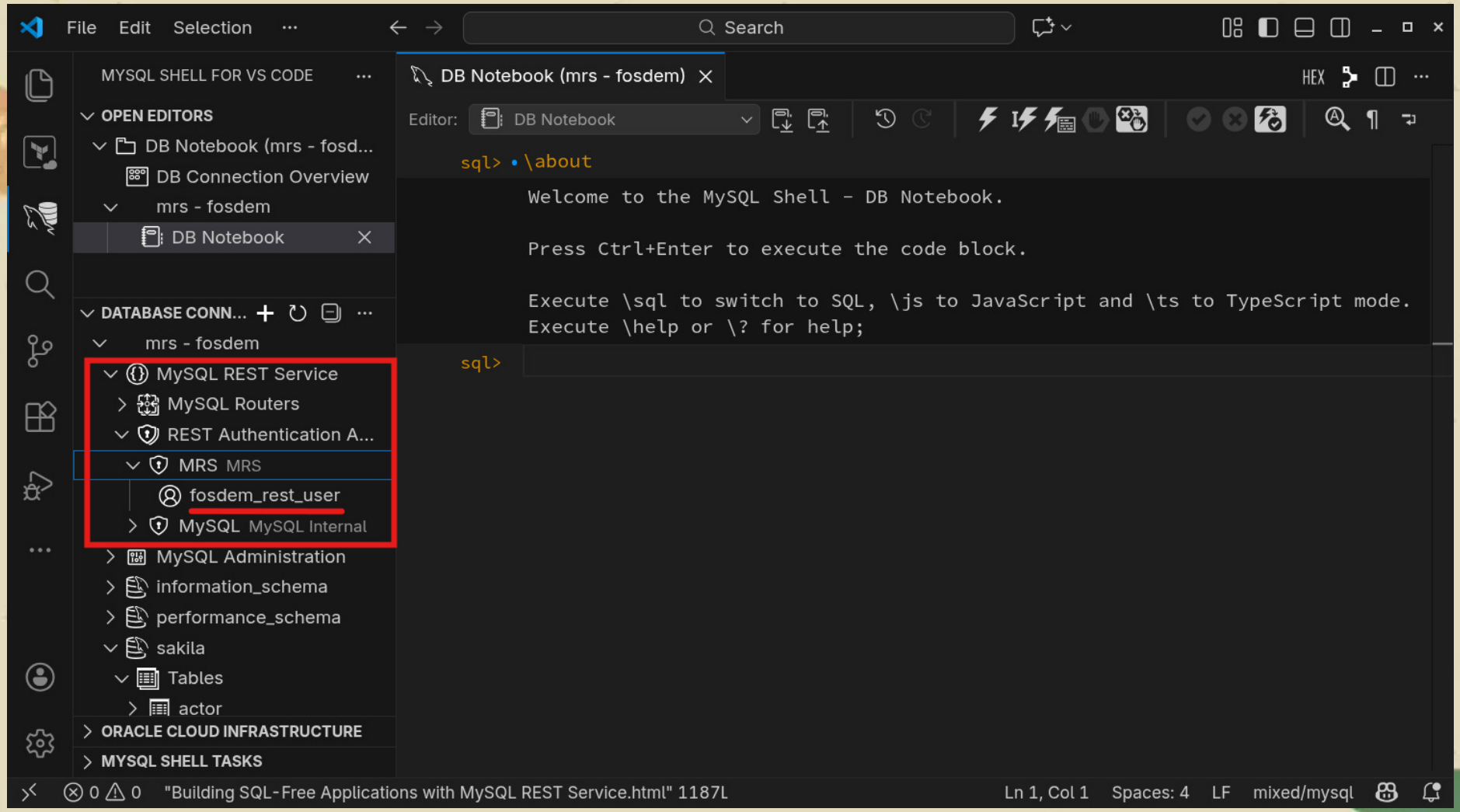
Creating the REST metadata - VS Code



Creating the REST metadata - VS Code



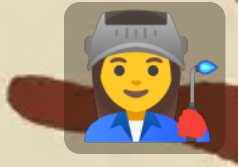
Creating the REST metadata - VS Code



Loading the MRS Component

This lab version provides a MRS component pre-installed but not yet loaded. This is a nice enhancement as you don't need to use MySQL Router as middleware between your application and the database server to provide RESTful services.

Loading the MRS Component



This lab version provides a MRS component pre-installed but not yet loaded. This is a nice enhancement as you don't need to use MySQL Router as middleware between your application and the database server to provide RESTful services.

```
MySQL > install component "file://component_mysql_rest_service";

MySQL > select * from mysql.component;
+--------------+--------------------+-----------------------------------+
| component_id | component_group_id | component_urn                     |
+--------------+--------------------+-----------------------------------+
|            1 |                  1 | file://component_mysql_rest_service |
+--------------+--------------------+-----------------------------------+
1 row in set (0.0014 sec)
```

MRS Component and Keyring - required

If the keyring service is not properly configured, you will get an error like this when loading the component:

```
MySQL > install component "file://component_mysql_rest_service";
ERROR: 6557 (HY000): Failed to initialize Keyring.
```

And in the error log:

```
MySQL Rest Service Component: Unable to initialize keyring:
MySQL Keyring is required to run MRS.
Install and initialize a supported MySQL keyring backend beforehand.
See https://dev.mysql.com/doc/refman/en/keyring.html
```

MRS Component and Keyring

Now the user for the REST Service internal operations is automatically created and its password is stored in the Keyring:

```
| System | MY-015163 | Server |
              MySQL Rest Service Component: Creating MRS account: mysql_mrs_1
```

```
SQL > SELECT to_user Users, GROUP_CONCAT(from_user SEPARATOR ', ') Roles
      FROM mysql.role_edges where to_user = 'mysql_mrs_1' GROUP BY to_user;
+-------------+-----------------------------------------------------------------+
| Users       | Roles                                                           |
+-------------+-----------------------------------------------------------------+
| mysql_mrs_1 | mysql_rest_service_data_provider, mysql_rest_service_meta_provider |
+-------------+-----------------------------------------------------------------+
1 row in set (0.0005 sec)
```

MRS Component and Keyring (2)

```
$ cat mysql-keyring/component_keyring_file | jq
{
  "version": "1.0",
  "elements": [
    {
      "user": "",
      "data_id": "mrs/rest-user/jwt_secret",
      "data_type": "SECRET",
      "data": "736563726574",
      "extension": []
    },
    {
      "user": "",
      "data_id": "mrs/mysql_mrs_1/password",
      "data_type": "SECRET",
      "data": "6D5F353355452C2C36404C416C716B534661744B",
      "extension": []
    }
  ]
}
```

Building RESTful APIs with MRS

MySQL without SQL

Our first RESTful Endpoint

We will create a first simple RESTful Endpoint that will expose the `actor` *table from the* `sakila` *database.*

We won't use any authentication for this first example to keep it simple.

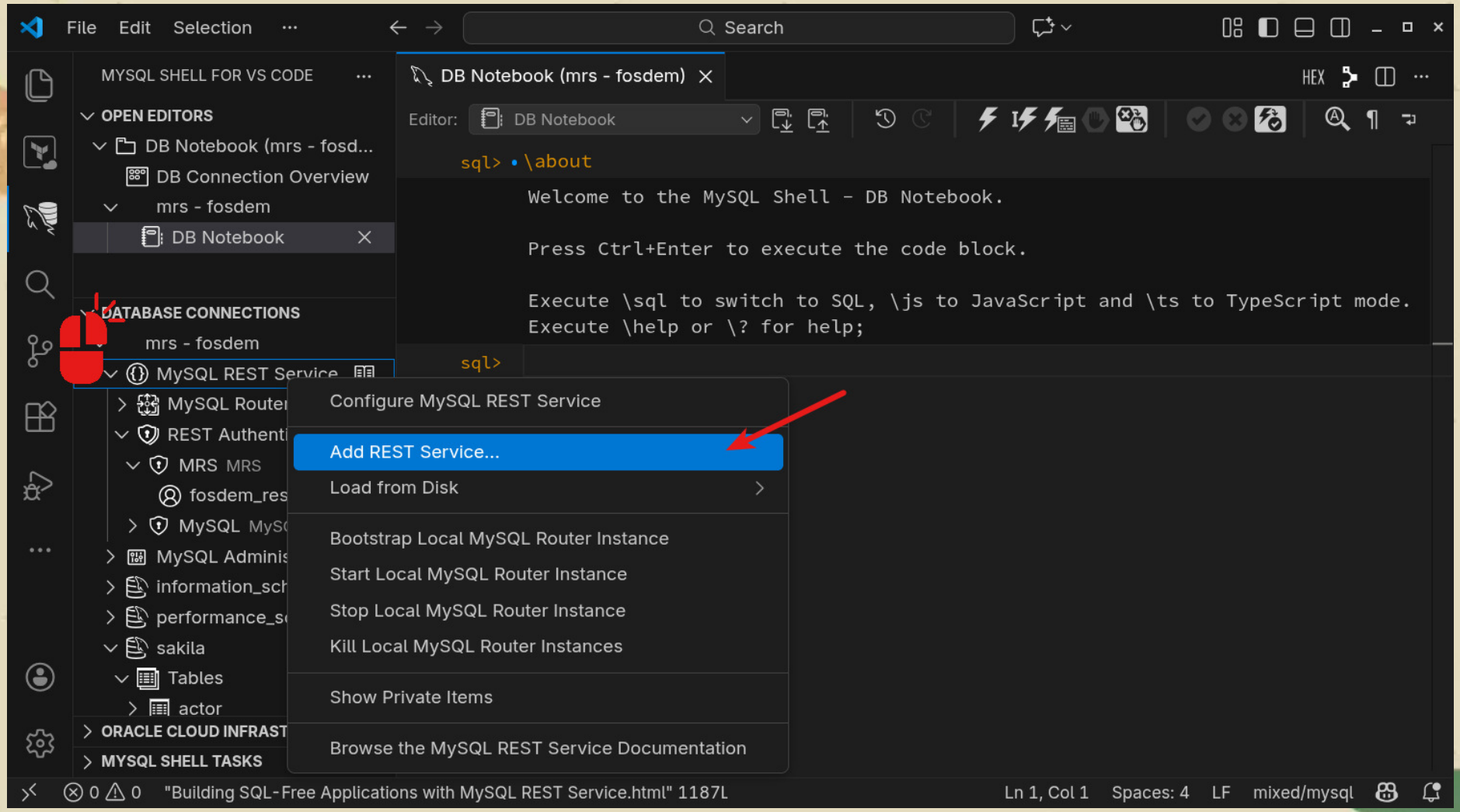
Our first RESTful Endpoint

We will create a first simple RESTful Endpoint that will expose the `actor` *table from the* `sakila` *database.*

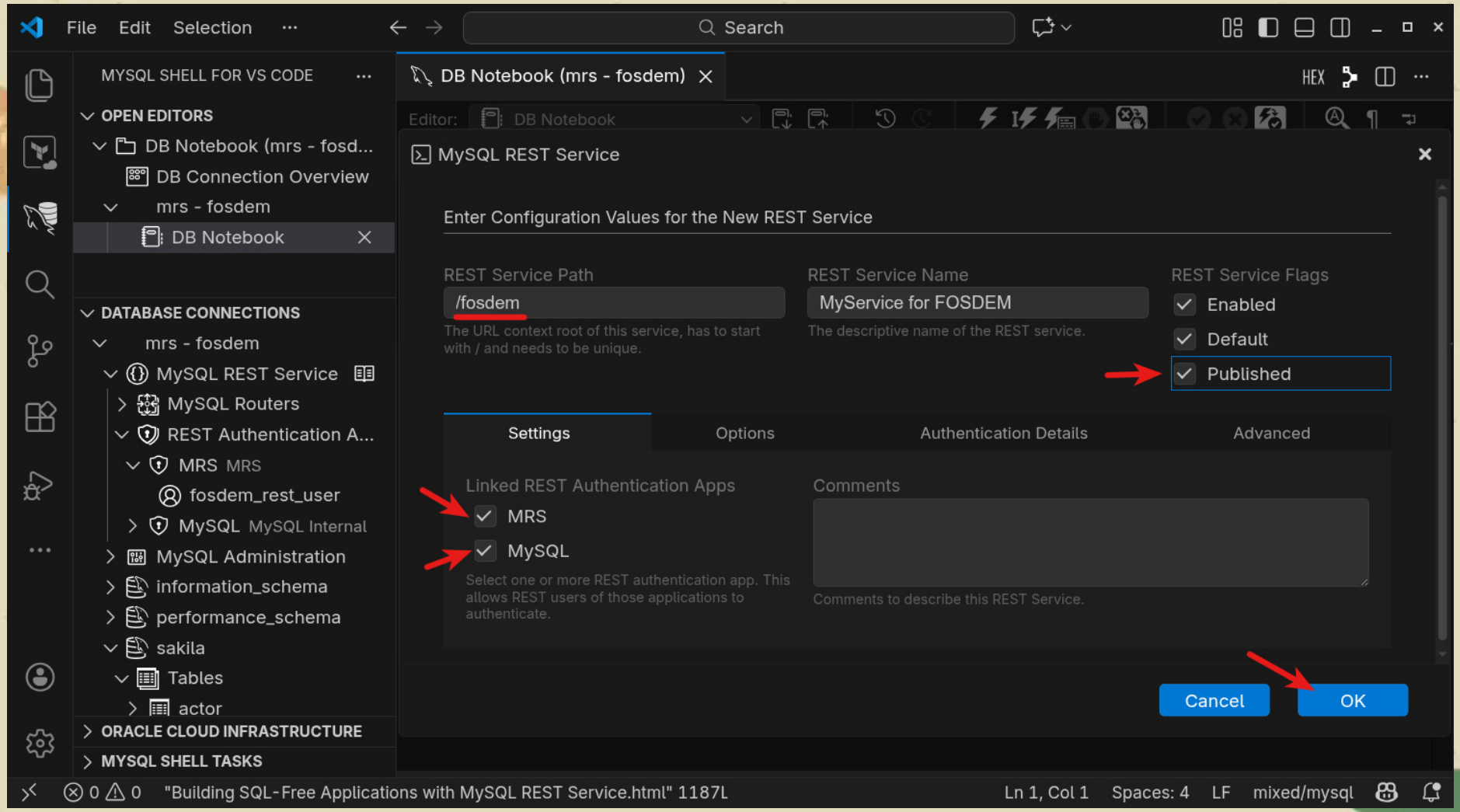
We won't use any authentication for this first example to keep it simple.

We need to create a REST Service first.

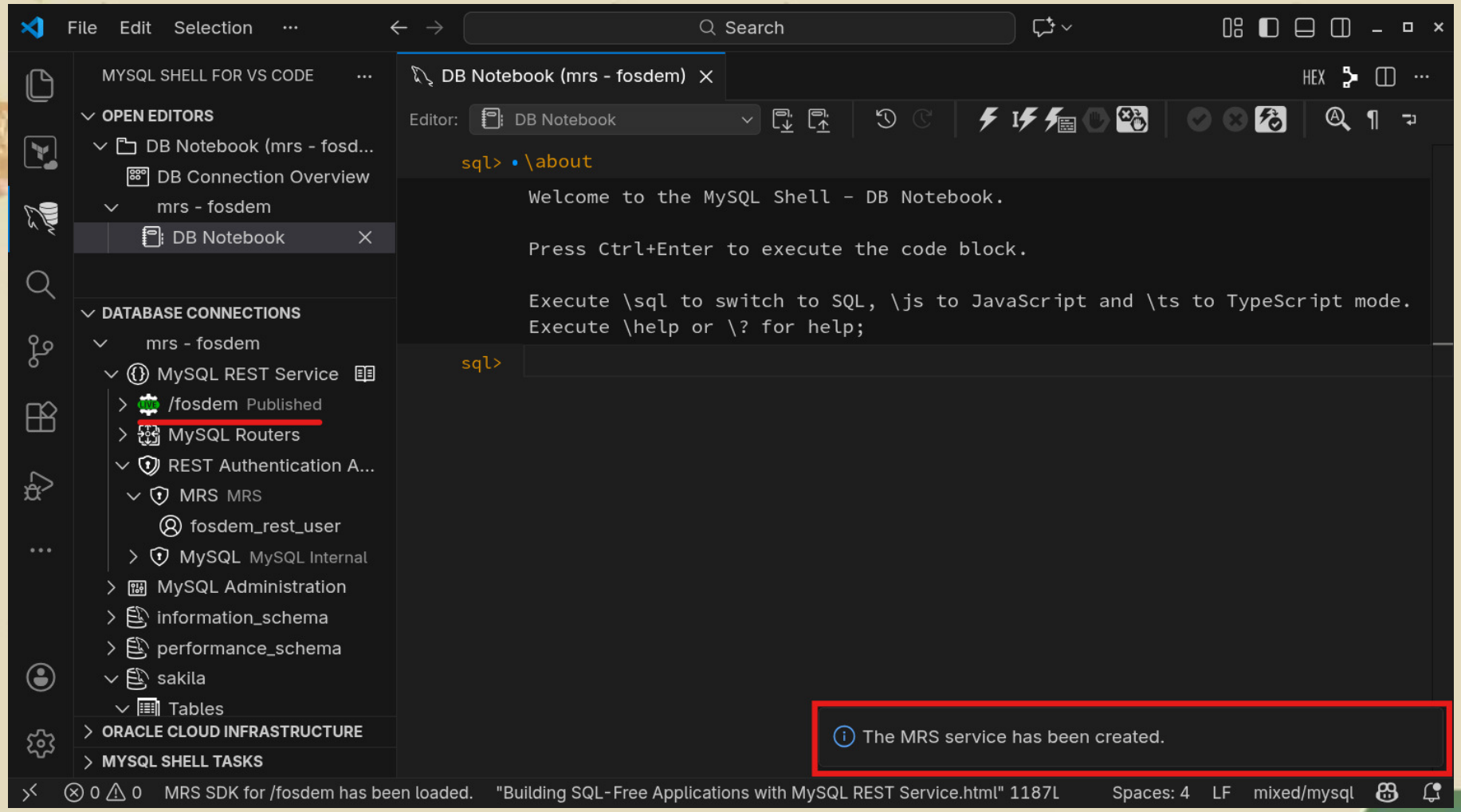
Our first REST Service



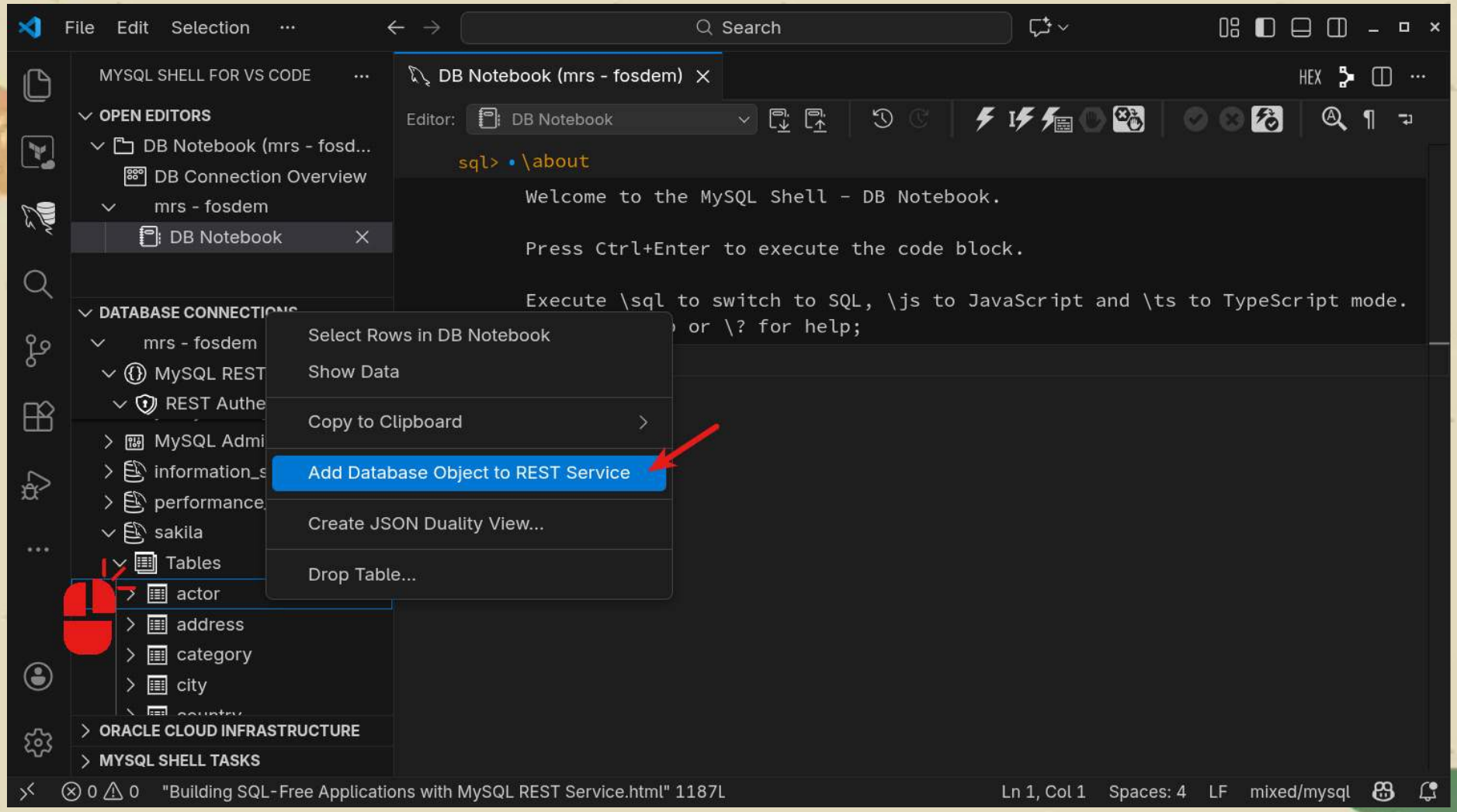
Our first REST Service



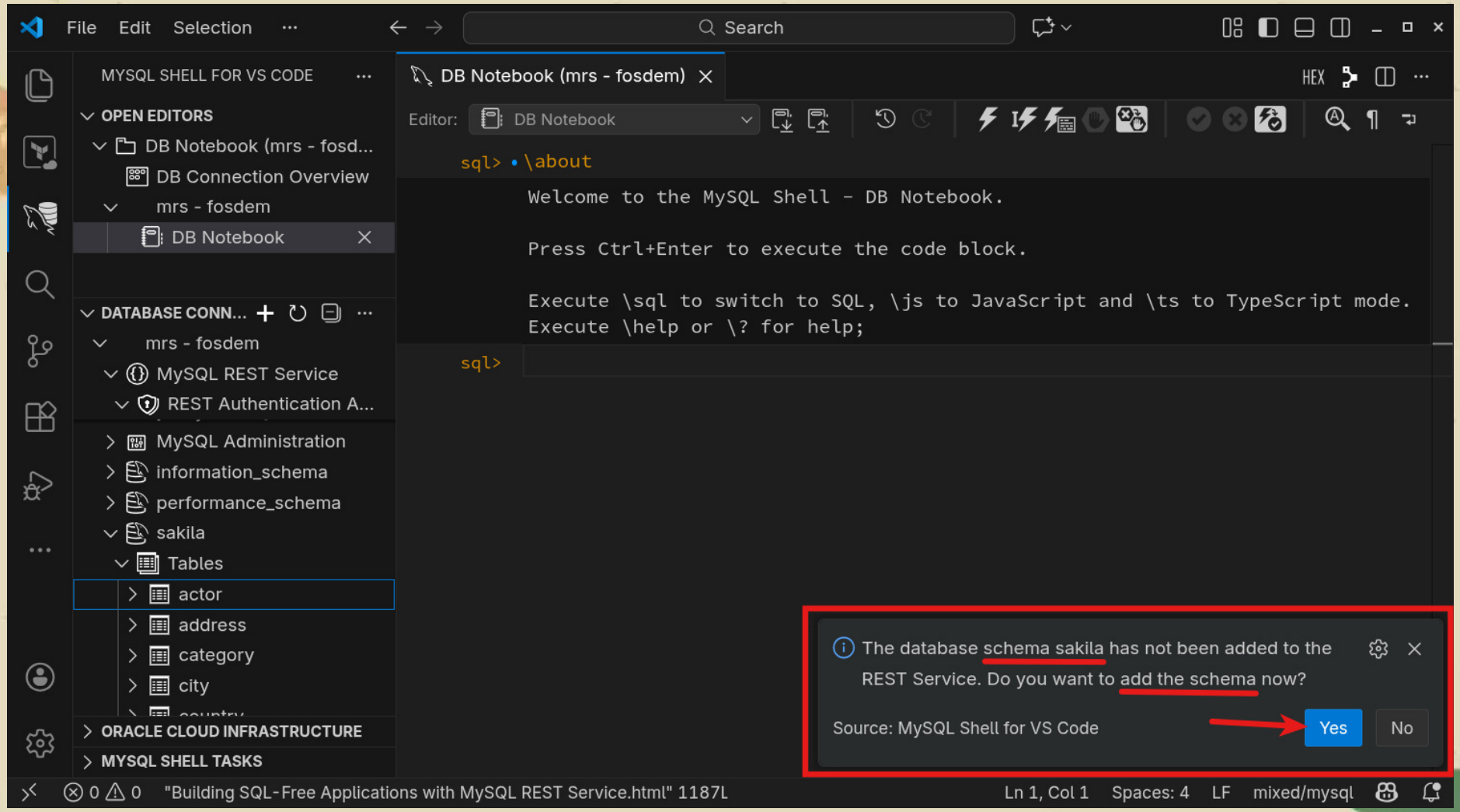
Our first REST Service



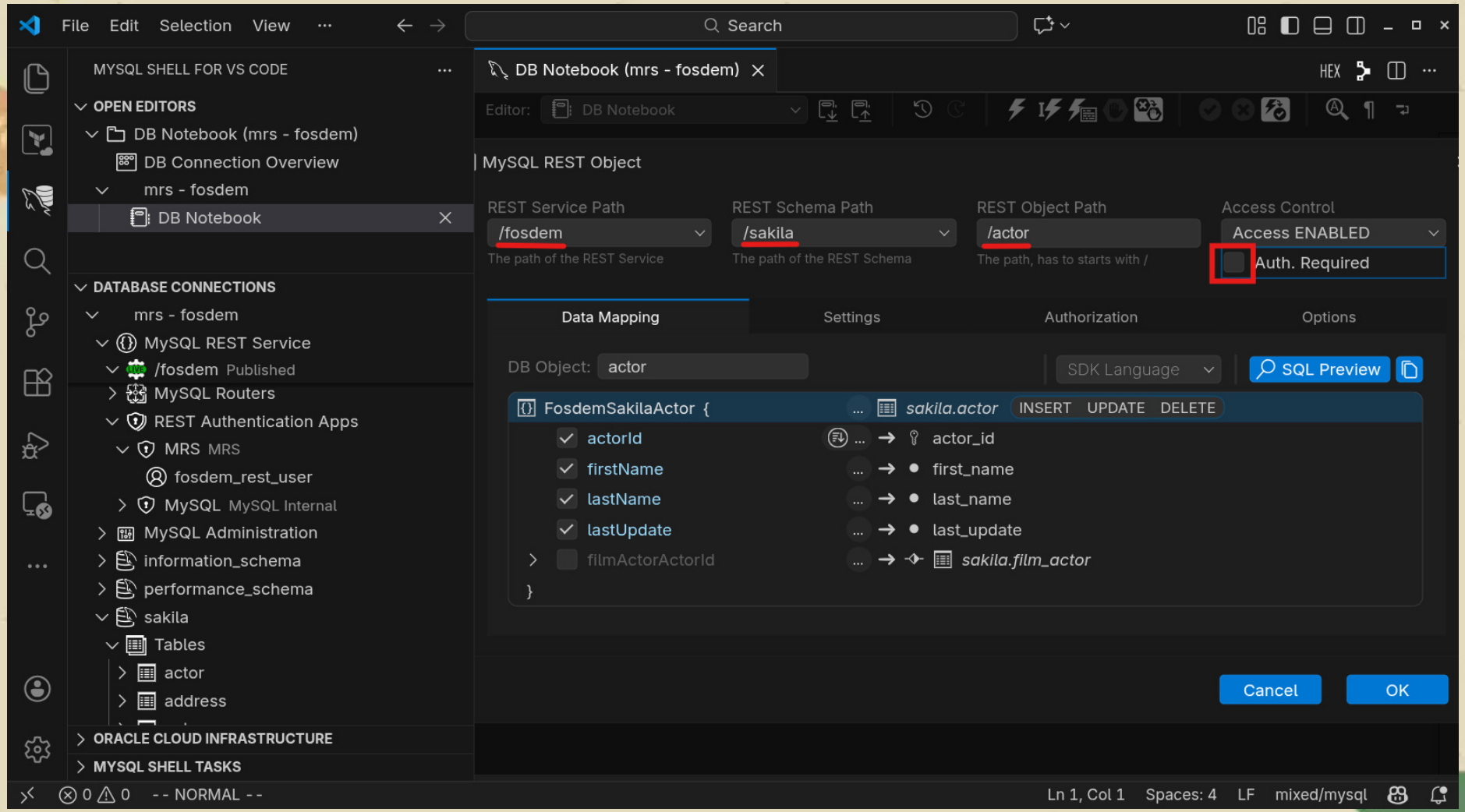
Our first RESTful Endpoint



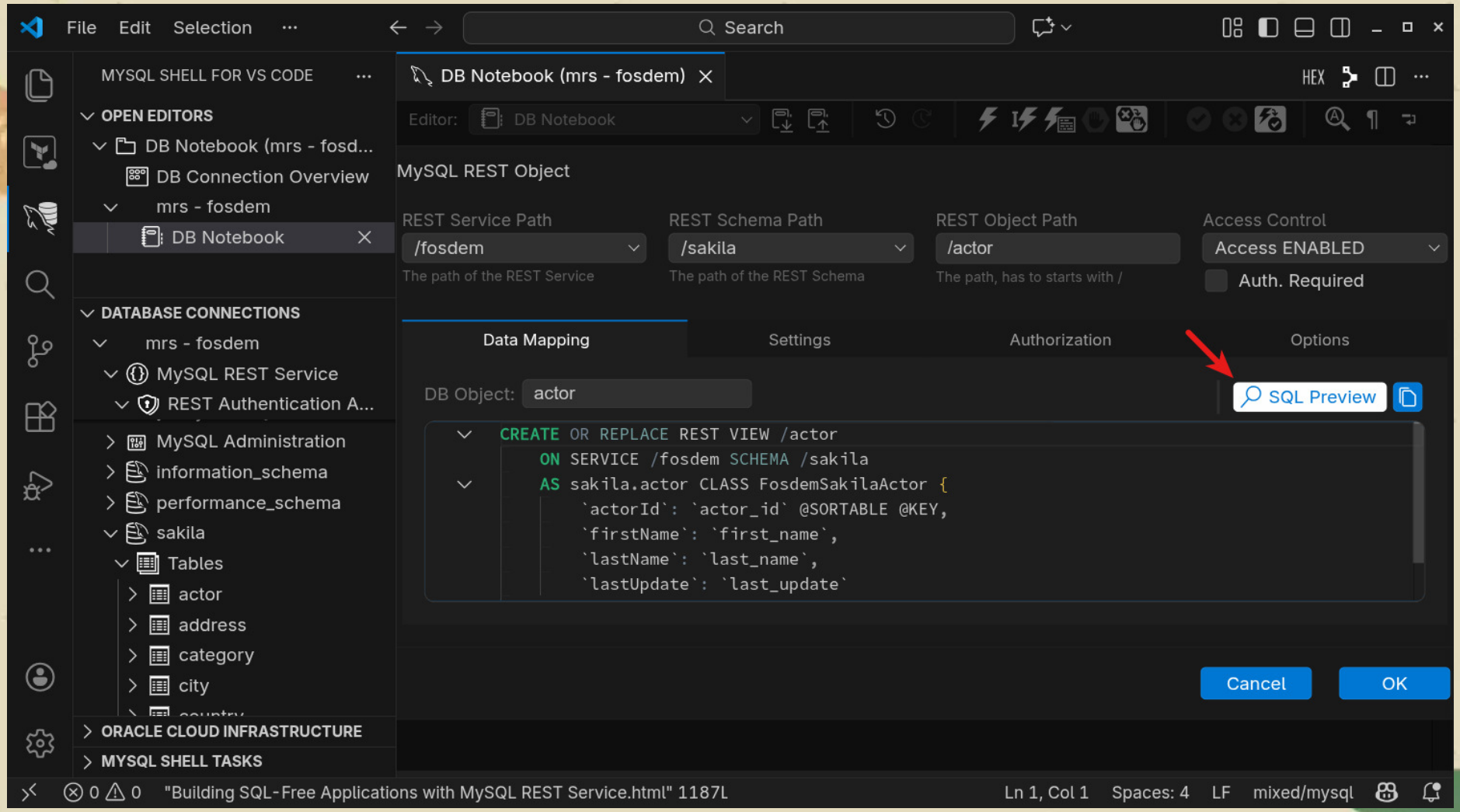
Our first RESTful Endpoint



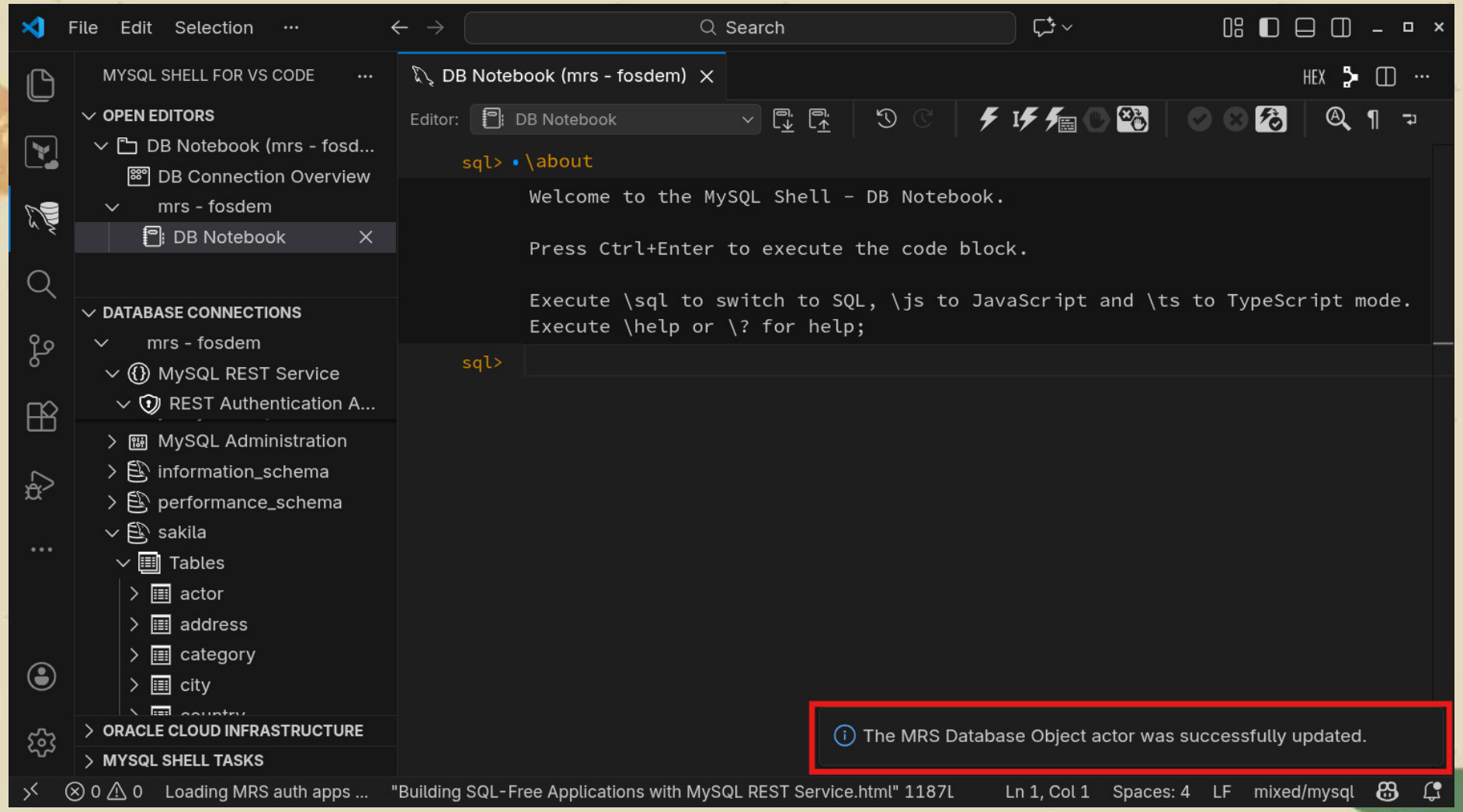
Our first RESTful Endpoint



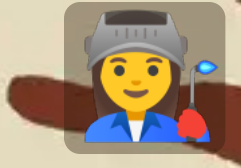
Our first RESTful Endpoint

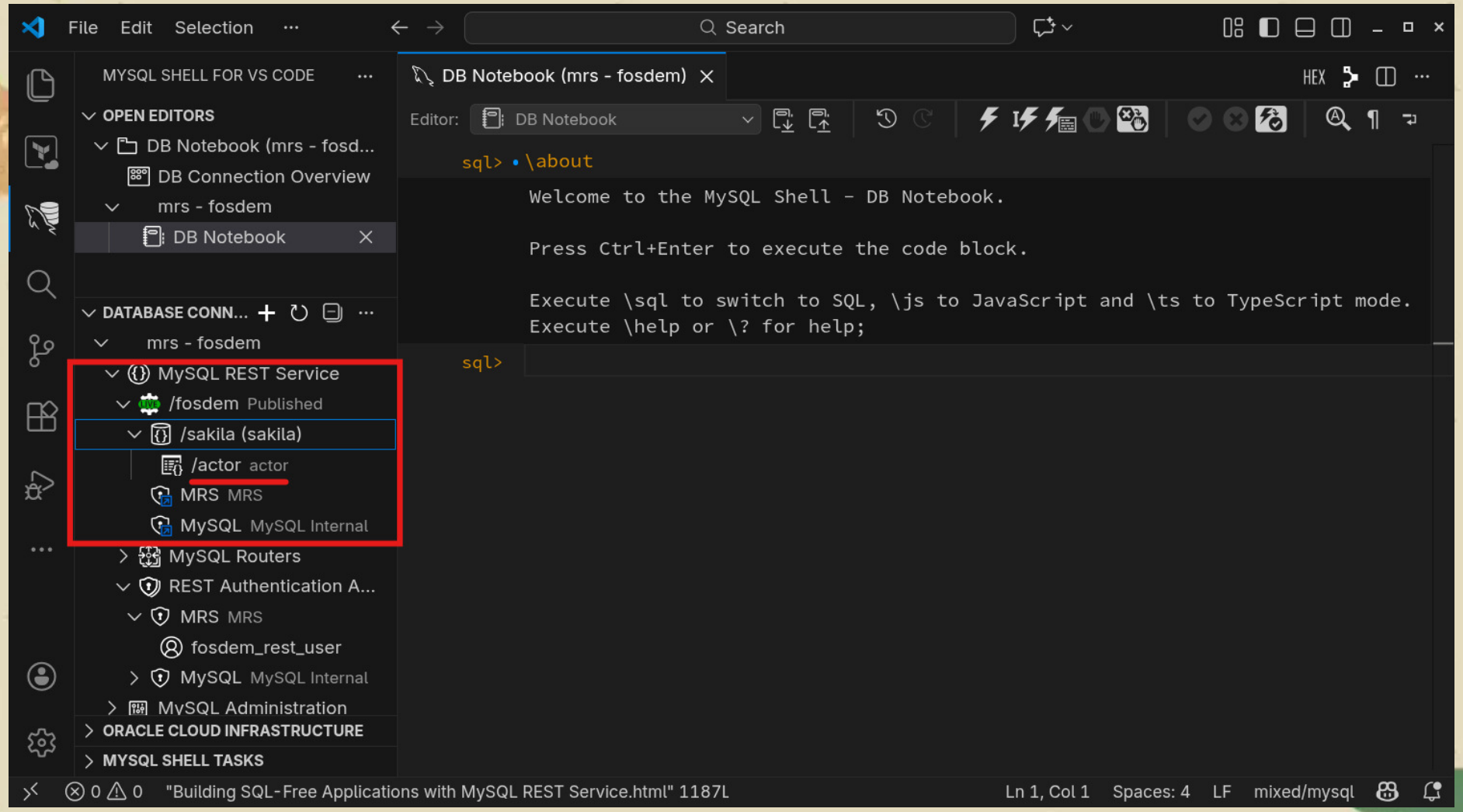


Our first RESTful Endpoint

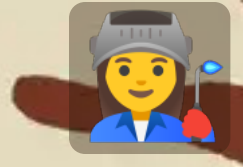


Our first RESTful Endpoint





Our first RESTful Endpoint - testing



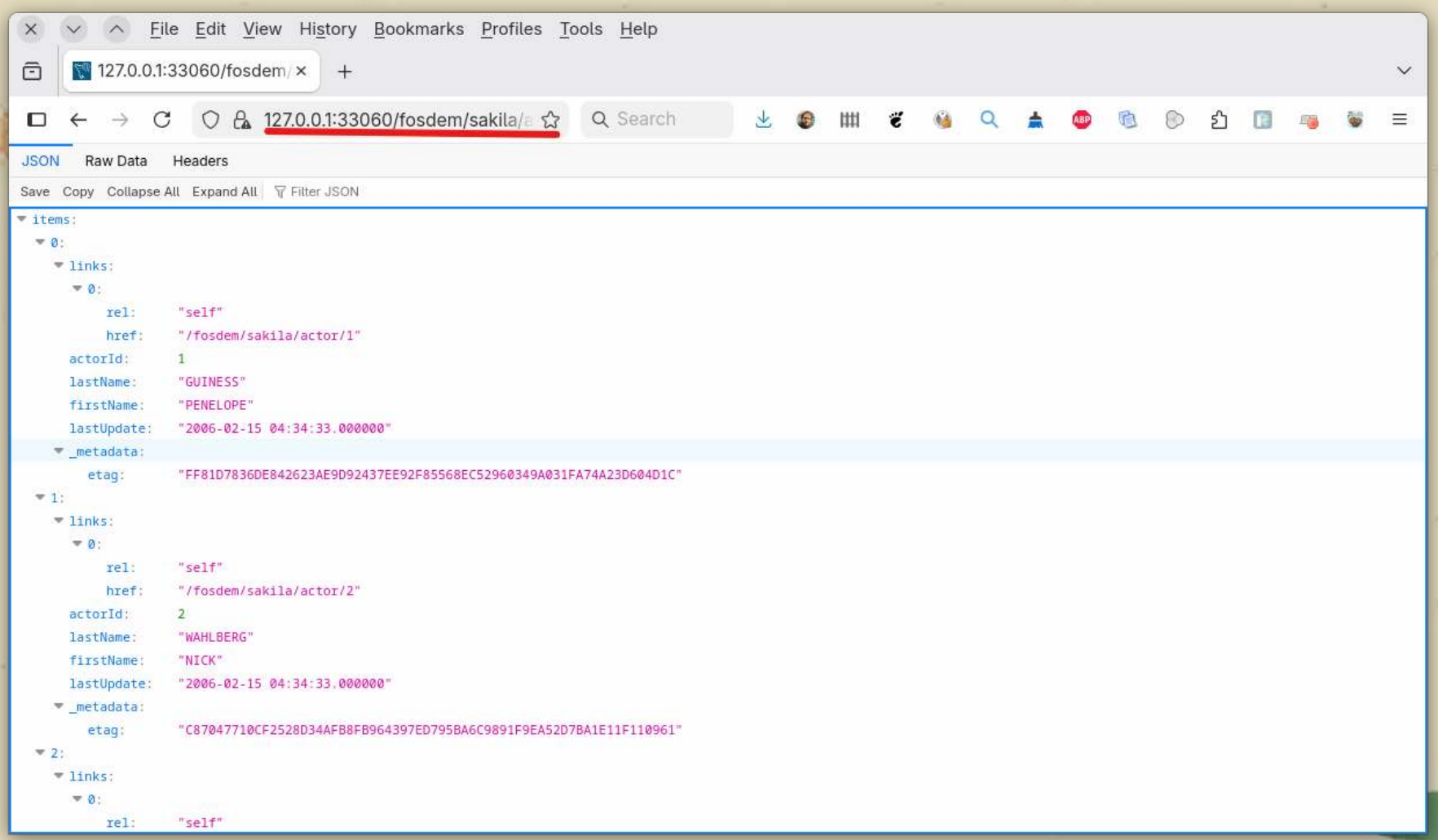
We can now try to browse to our first RESTful Endpoint using curl or a browser:

https://127.0.0.1:33060/fosdem/sakila/actor

The ceritficate is self-signed so you need to accept the risk if you use a browser.

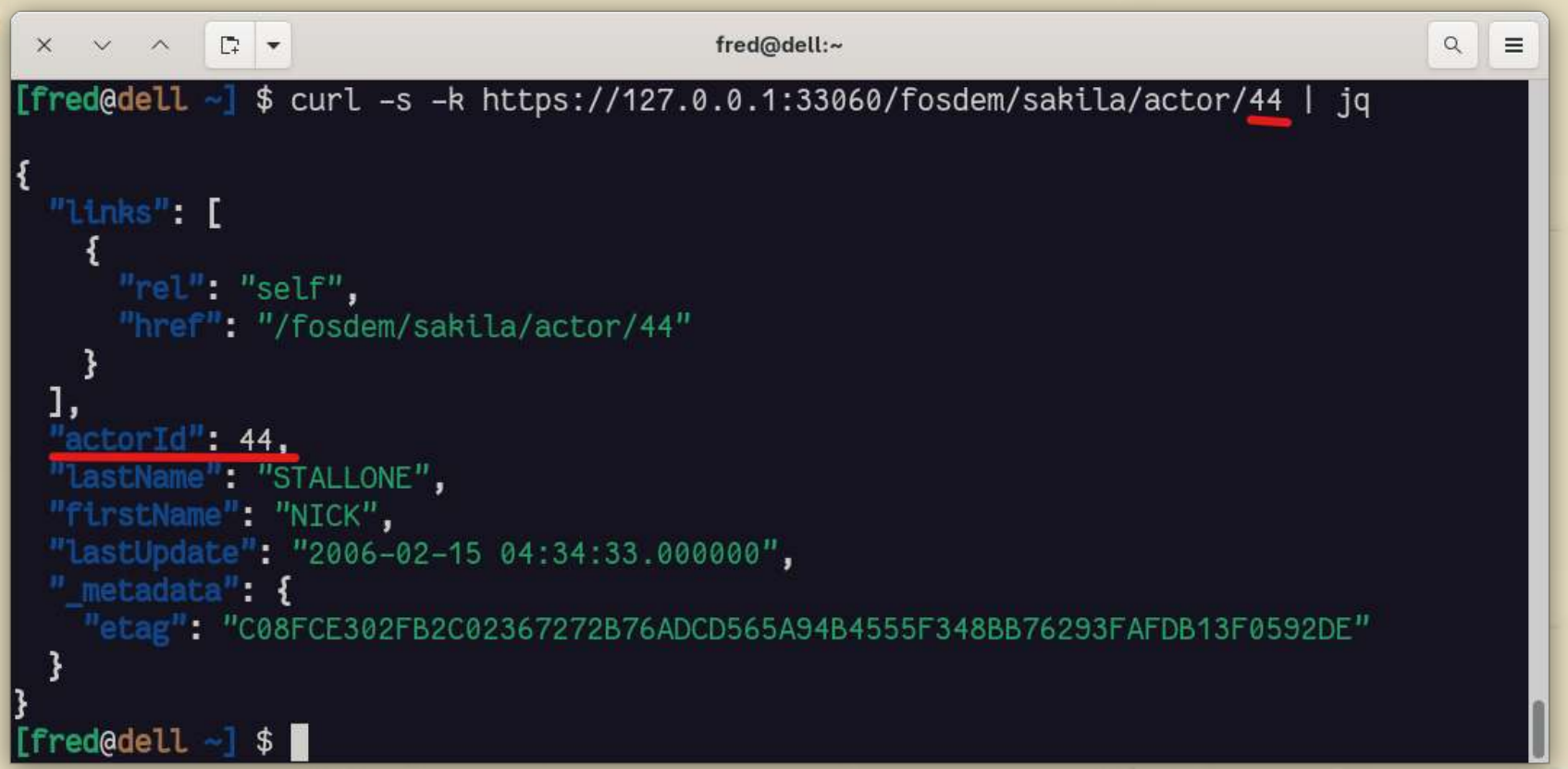
```
$ curl -s -k https://127.0.0.1:33060/fosdem/sakila/actor | jq
```

Our first RESTful Endpoint - testing



Our first RESTful Endpoint - filtering

We can request an actor by its primary key like this:



Our first RESTful Endpoint - filtering (2)

Or even filter out on fields like this:

```
$ curl -s -k --get 'https://127.0.0.1:33060/fosdem/sakila/actor/' \
   --data-urlencode 'q={"firstName":"NICK"}' | \
   jq -r '.items[] | "\(.actorId)\t\(.firstName)\t\(.lastName)"'
2   NICK    WAHLBERG
44  NICK    STALLONE
166 NICK    DEGENERES
```

Our first RESTful Endpoint - filtering (2)

Or even filter out on fields like this:

```
$ curl -s -k --get 'https://127.0.0.1:33060/fosdem/sakila/actor/' \
   --data-urlencode 'q={"firstName":"NICK"}' | \
   jq -r '.items[] | "\(.actorId)\t\(.firstName)\t\(.lastName)"'
2   NICK    WAHLBERG
44  NICK    STALLONE
166 NICK    DEGENERES
```

```
MySQL > select count(*) from actor where first_name like 'nick';
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.0005 sec)
```

Building RESTful APIs with MRS

MySQL without SQL and without GUI

All in Command Line in SQL

We can also create our RESTful Endpoint using only SQL.

This is useful if we want to automate the creation of RESTful Endpoints.

We can get all information about the RESTful Endpoints from MySQL Shell in VS Code by dumping your services:

All in Command Line in SQL (2)

```
CREATE OR REPLACE REST SERVICE /fosdem PUBLISHED
    OPTIONS {
        "http": { "allowedOrigin": "auto" },
        "headers": {
            "Access-Control-Allow-Headers": "Content-Type, Authorization, X-Requested-With, Origin, X-Auth-Token",
            "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, OPTIONS",
            "Access-Control-Allow-Credentials": "true"
        },
        "logging": {
            "request": {
                "body": true,
                "headers": true
            },
            "response": {
                "body": true,
                "headers": true
            },
            "exceptions": true
        },
        "includeLinksInResults": false,
        "returnInternalErrorDetails": true
    }
```

All in Command Line in SQL (3)

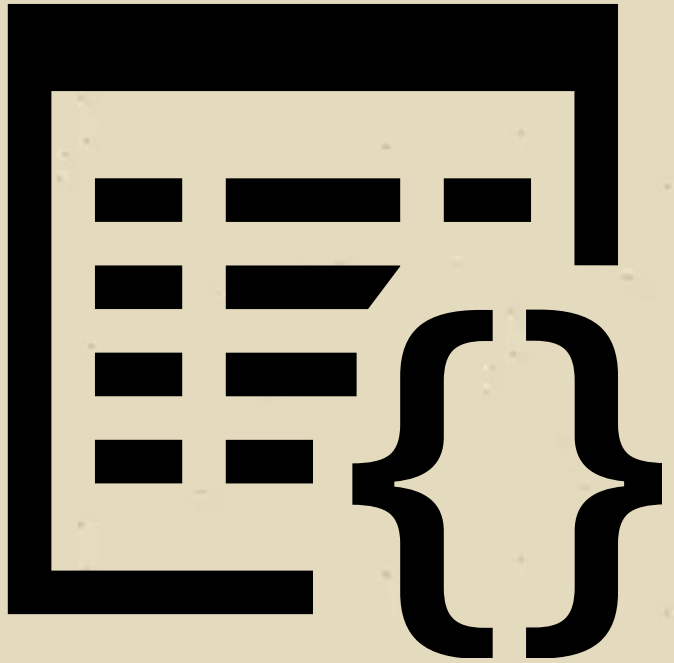
```
    ADD AUTH APP `MRS` IF EXISTS
    ADD AUTH APP `MySQL` IF EXISTS;

CREATE OR REPLACE REST SCHEMA /sakila ON SERVICE /fosdem
    FROM `sakila`
    AUTHENTICATION NOT REQUIRED;

CREATE OR REPLACE REST VIEW /actor
    ON SERVICE /fosdem SCHEMA /sakila
    AS sakila.actor CLASS FosdemSakilaActor @INSERT @UPDATE {
        firstName: first_name,
        lastName: last_name,
        actorId: actor_id @KEY @SORTABLE,
        lastUpdate: last_update
    }
    AUTHENTICATION REQUIRED;
```

Using the Python SDK

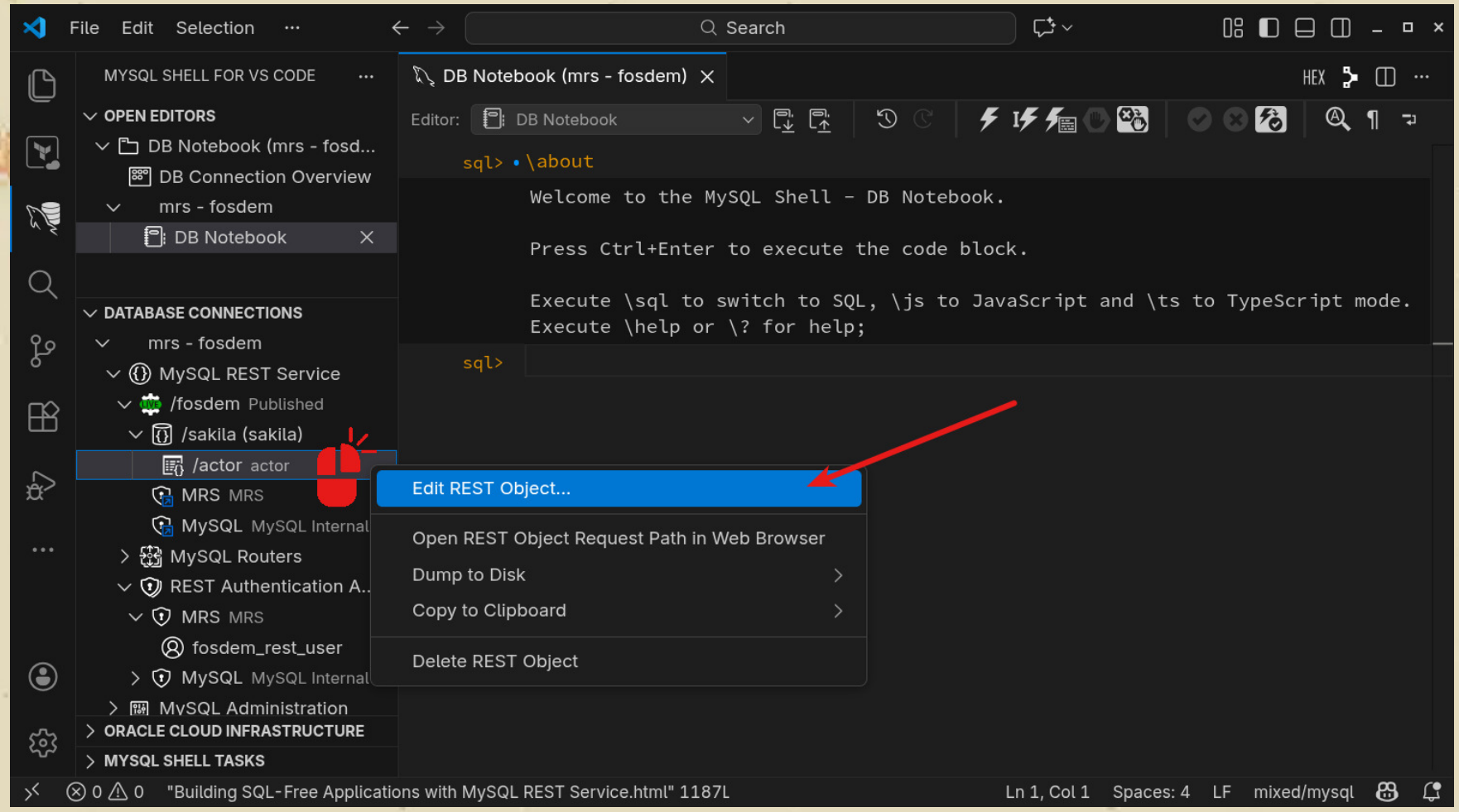
Modify our RESTful Endpoint

We will now modify our RESTful Endpoint to return a more complex JSON structure and enable authentication.

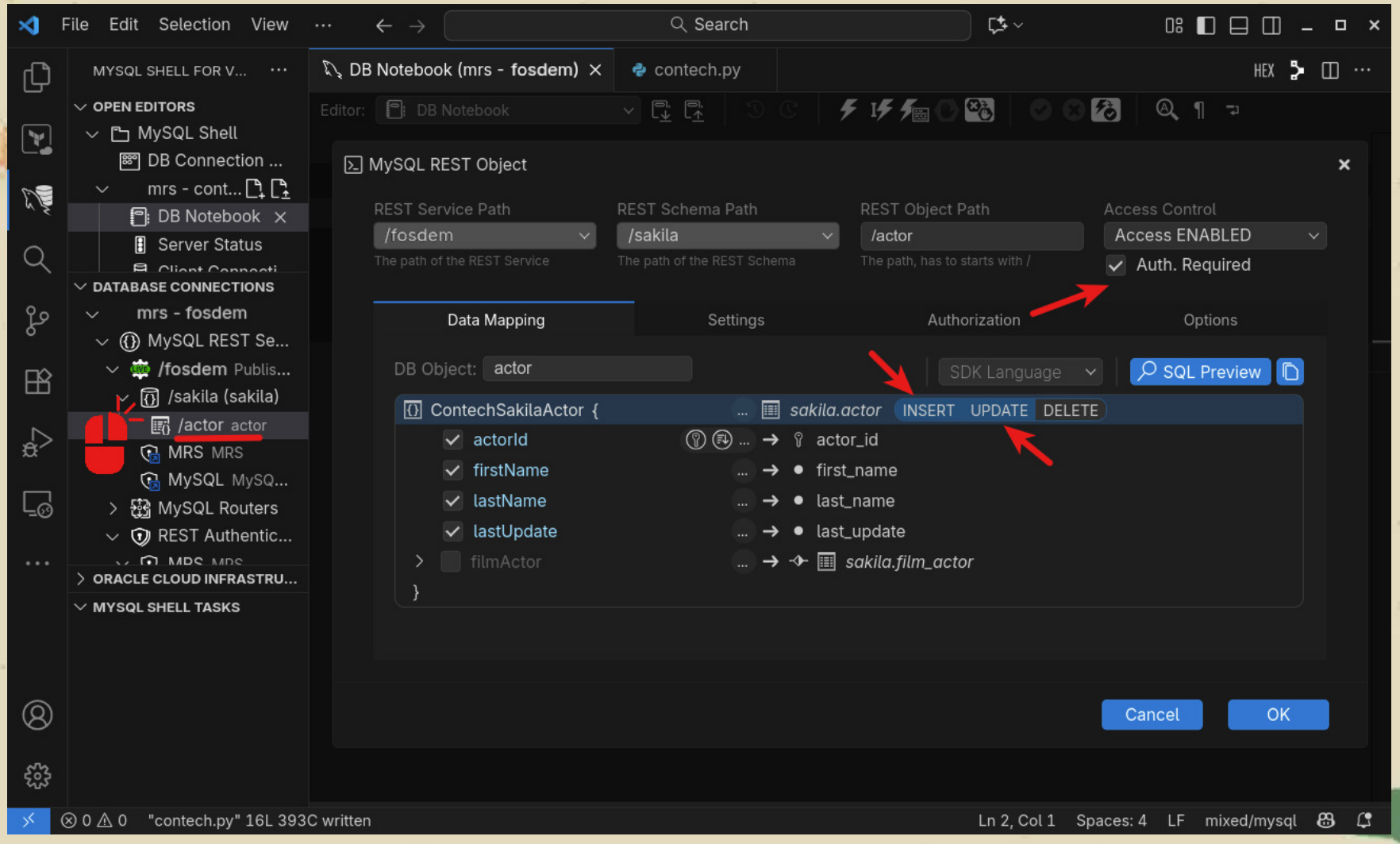
We will start by editing the REST Object `/actor` *we created before.*



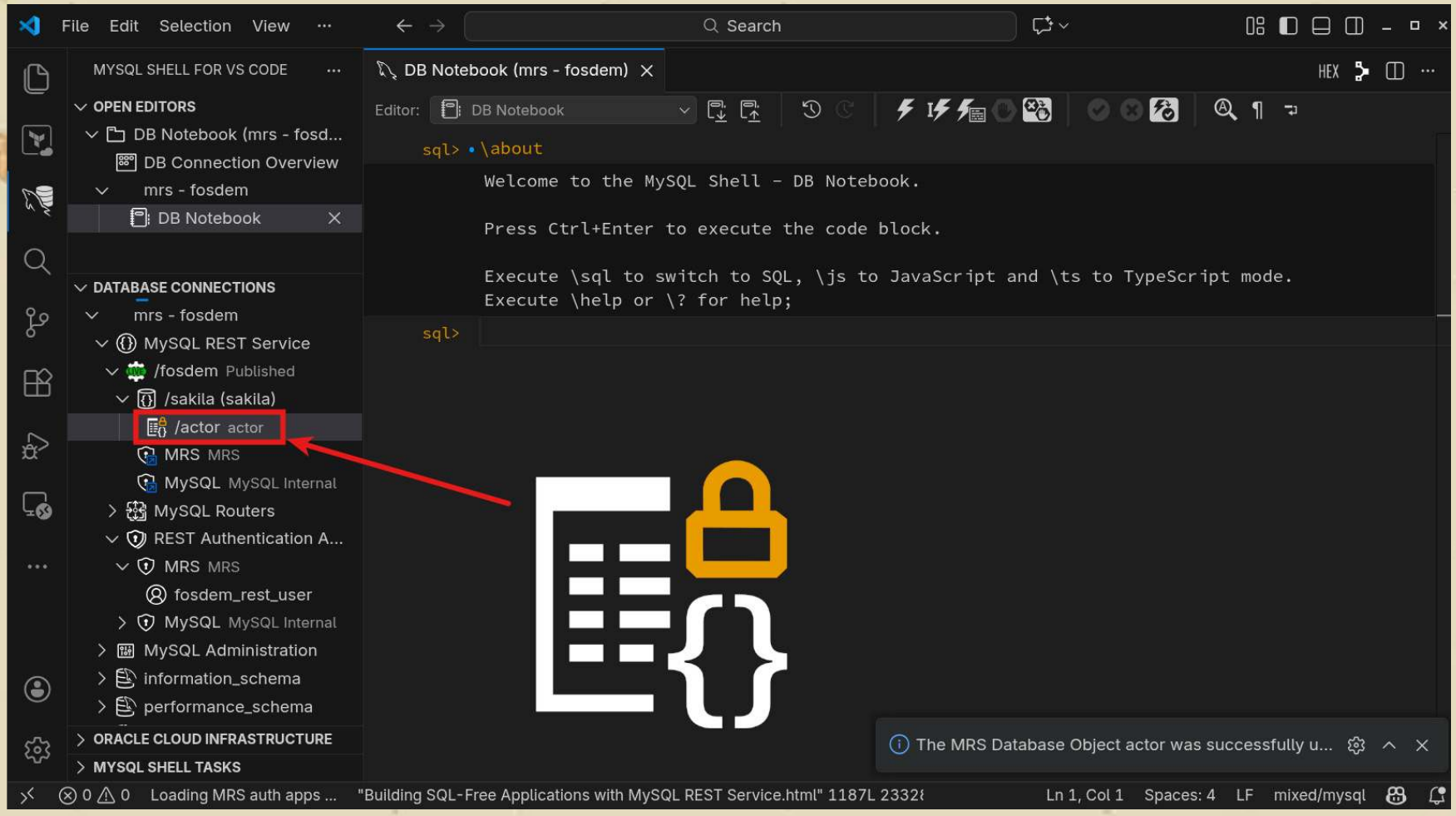
Modify our RESTful Endpoint



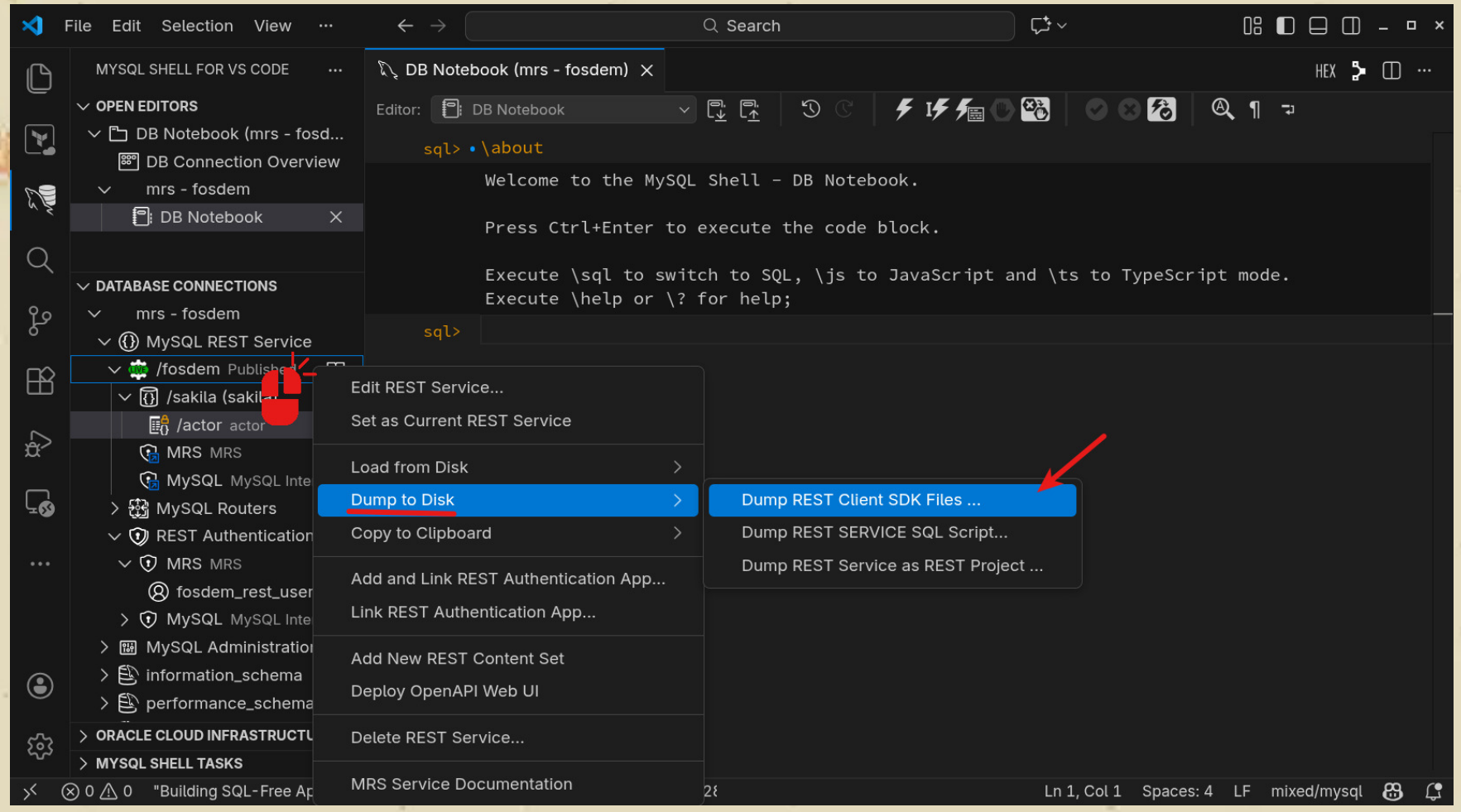
Modify our RESTful Endpoint



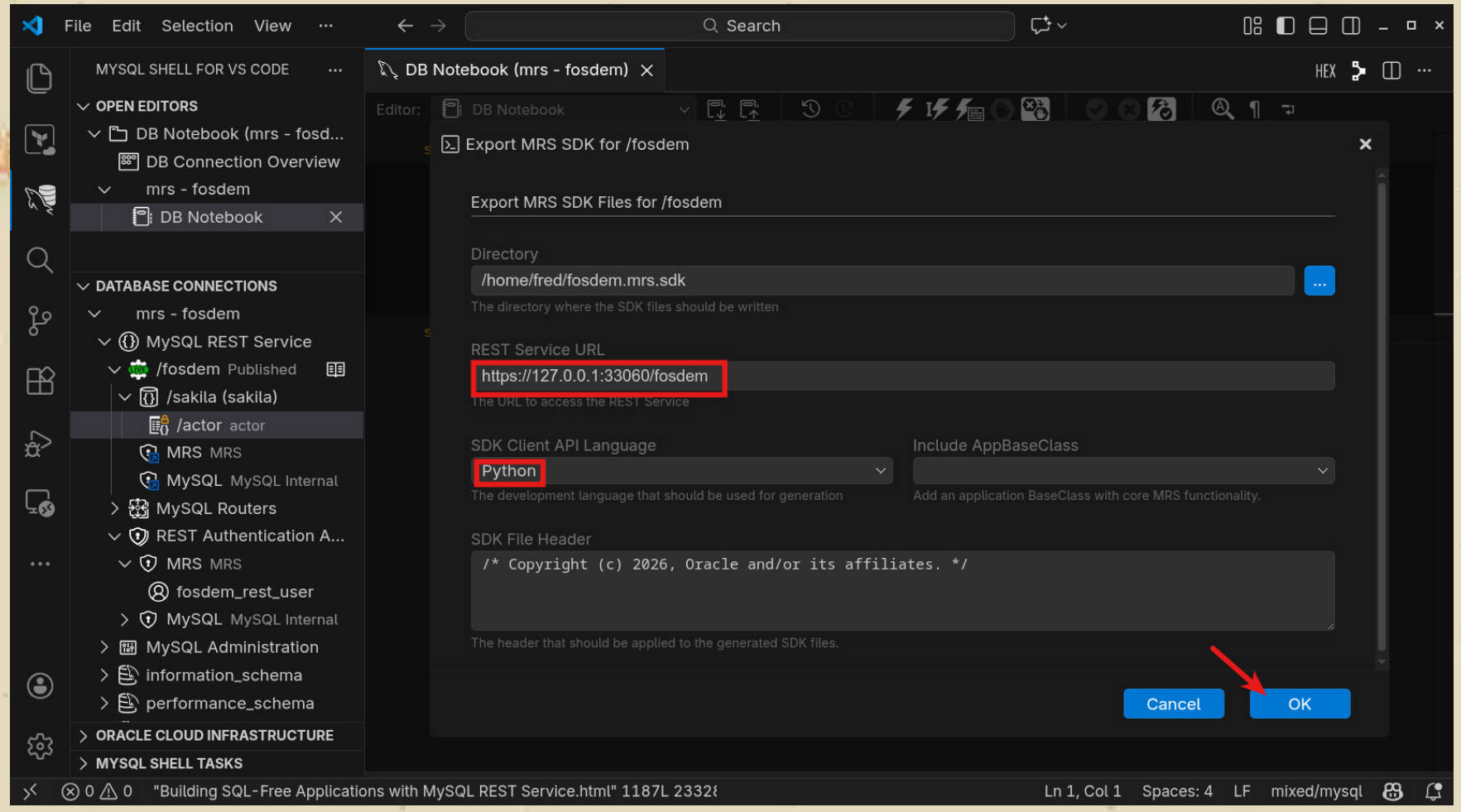
Modify our RESTful Endpoint



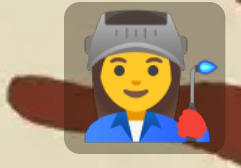
Get the SDK

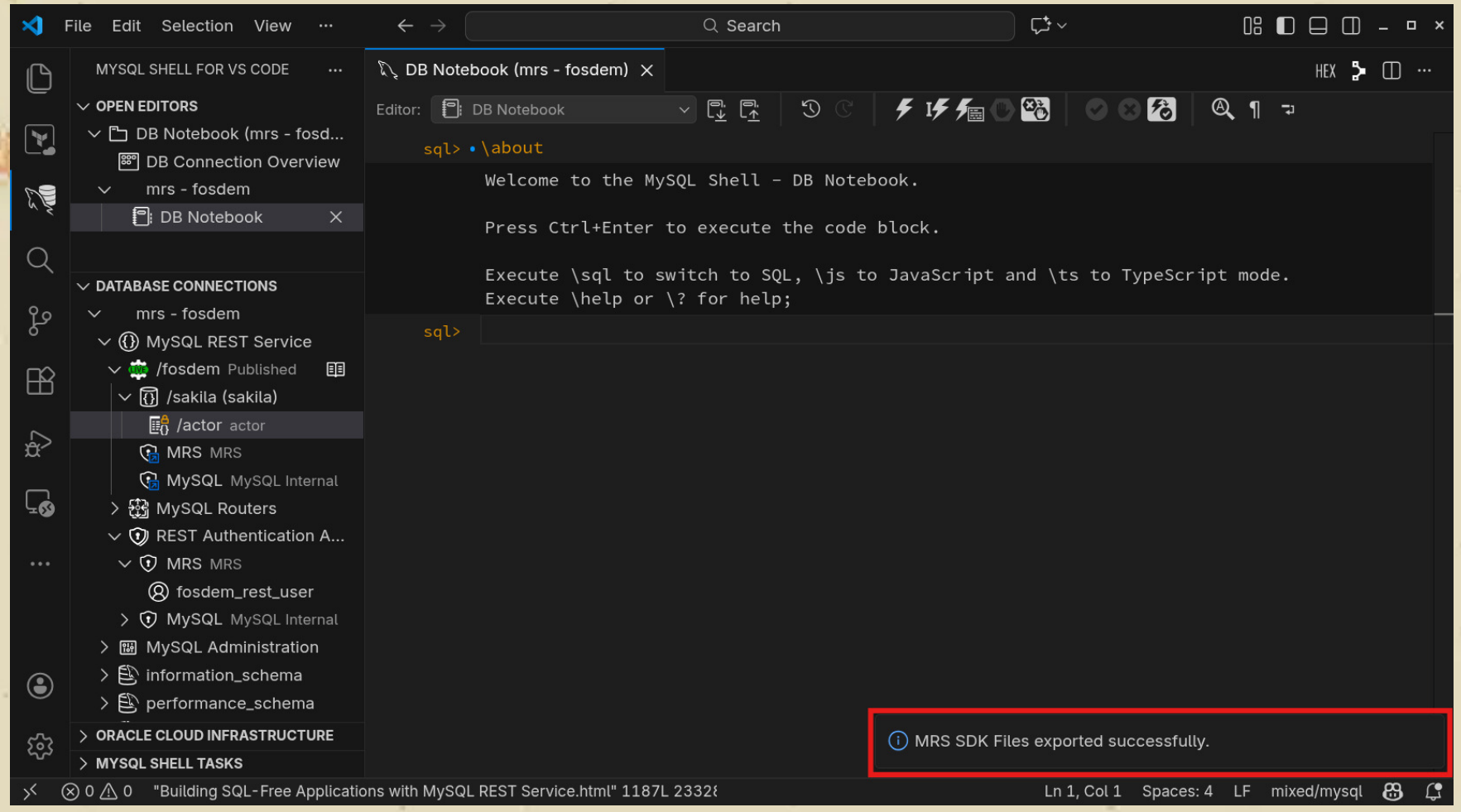


Get the SDK



Get the SDK





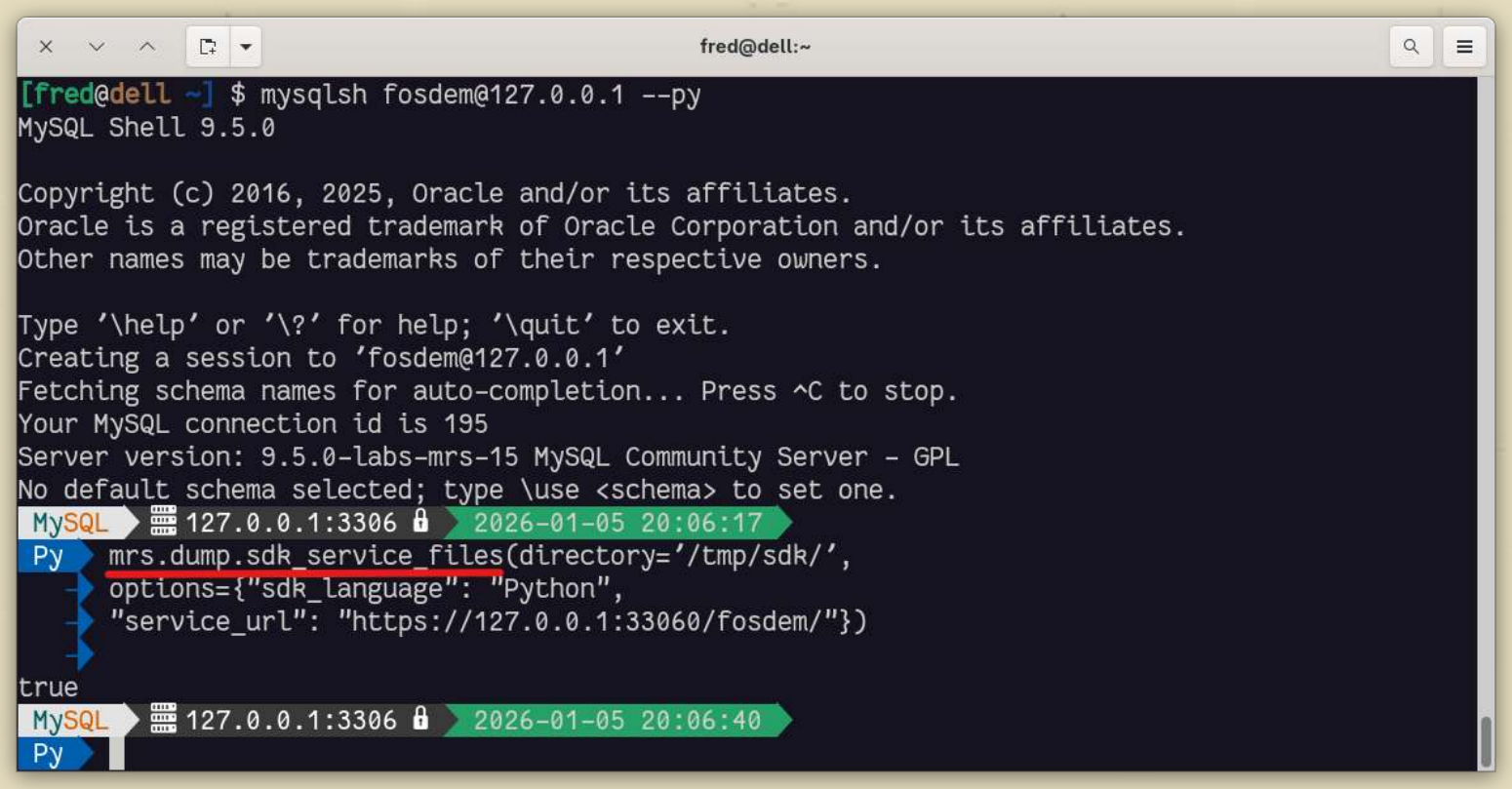
Get the SDK

We can see that the SDK has been saved in our file system:

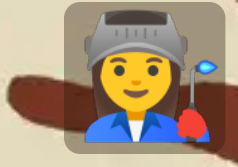
```
$ du -sh fosdem.mrs.sdk
128K    fosdem.mrs.sdk

$ tree fosdem.mrs.sdk/
fosdem.mrs.sdk/
├── fosdem.py
├── __init__.py
├── mrs_base_classes.py
└── mrs.config.json

1 directory, 4 files
```

Get the SDK - cmd line

It's also possible to export the SDK using MySQL Shell command Line without using MySQL Shell for VS Code like this;



Using the SDK



We create a directory for our Python code and copy the SDK there:

```
[~] $ mkdir fosdem
[~] $ mv fosdem.mrs.sdk fosdem/sdk
[~] $ cd fosdem/
[~/fosdem] $ touch fosdem.py
```

We can edit our code in the terminal (using Vi) or using VS Code.

Using the SDK


```python
from  sdk.fosdem import *

my_service = Fosdem(verify_tls_cert=False)
async def main():
    await my_service.authenticate(
        username = "fosdem_rest_user",
        password = "Fosd3m2026!",
        app = "MRS",
    )
    records = await my_service.sakila.actor.find(take=10)
    for record in records:
        print(record.first_name + " " + record.last_name)

asyncio.run(main())
```

Using the SDK - running it

```
$ python3.13 fosdem.py
PENELOPE GUINESS
NICK WAHLBERG
ED CHASE
JENNIFER DAVIS
JOHNNY LOLLOBRIGIDA
BETTE NICHOLSON
GRACE MOSTEL
MATTHEW JOHANSSON
JOE SWANK
CHRISTIAN GABLE
```

Using the SDK - updates

We can also try to update a record like this:

fosdem_update.py

```
from  sdk.fosdem import *

my_service = Fosdem(verify_tls_cert=False)

async def main():
    await my_service.authenticate(
        username = "fosdem_rest_user", password = "Fosd3m2026!", app = "MRS",
    )
    records = await my_service.sakila.actor.find(where={"actor_id": {"$eq": 4}})
    for record in records:
        print(record.first_name + " " + record.last_name)
        record.first_name = "ANDRA"
        await record.update();

asyncio.run(main())
```

Using the SDK - updates (2)

```
$ python3.13 fosdem_update.py
JENNIFER DAVIS
```

Using the SDK - updates (2)

```
$ python3.13 fosdem_update.py
JENNIFER DAVIS
```

```
$ python3.13 fosdem.py
PENELOPE GUINESS
NICK WAHLBERG
ED CHASE
ANDRA DAVIS
JOHNNY LOLLOBRIGIDA
BETTE NICHOLSON
GRACE MOSTEL
MATTHEW JOHANSSON
JOE SWANK
CHRISTIAN GABLE
```

Playing with other database objects

We can also use the SDK to play with other database objects like Views, Stored Procedures and Functions.

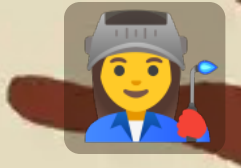
The full documentation is available at:

https://dev.mysql.com/doc/dev/mysql-rest-service/latest/sdk.html

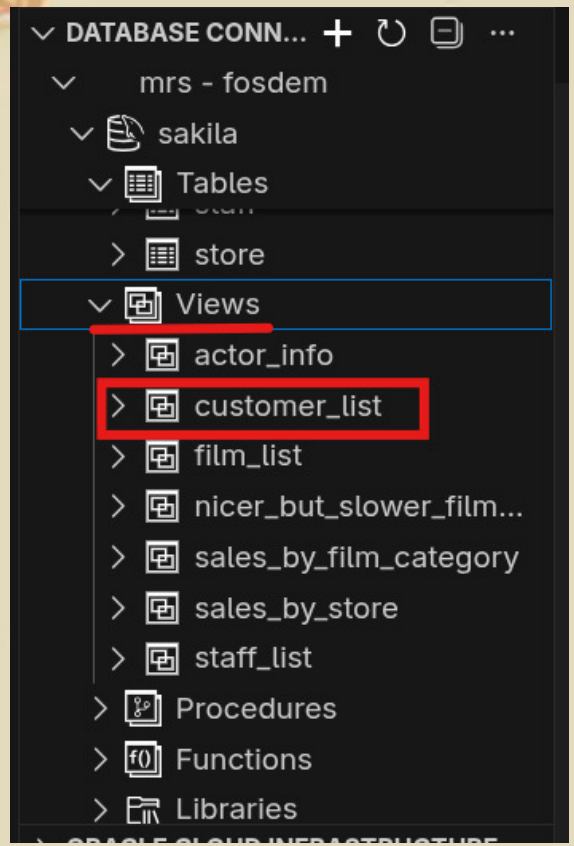
Using Views

Let's use one of the Views available in the Sakila database: `customer_list`

Using Views

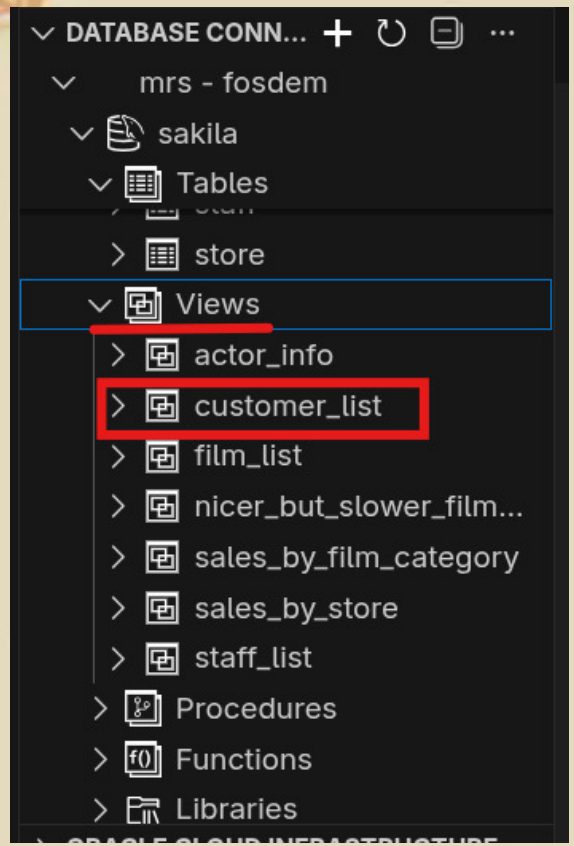


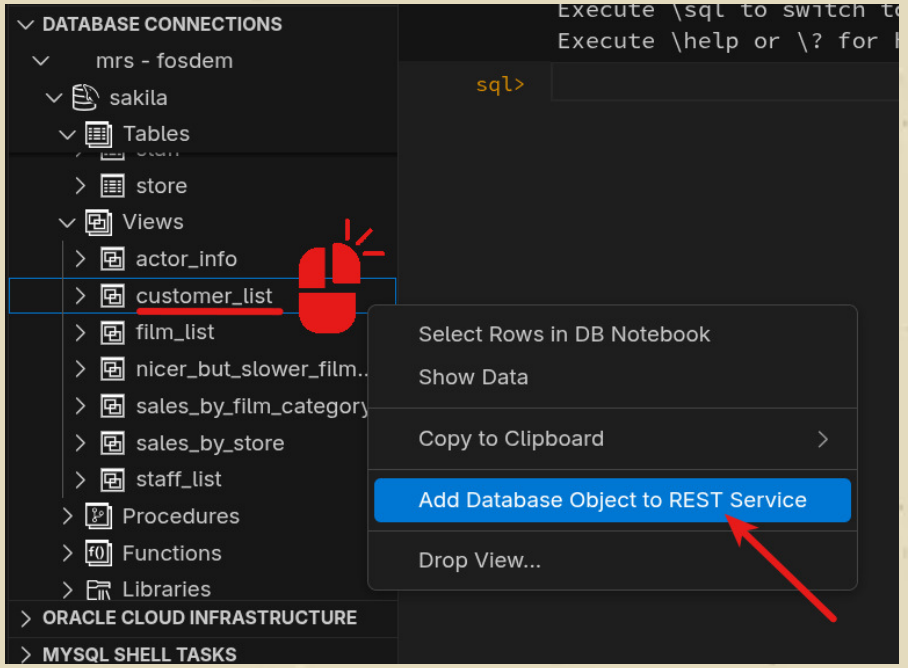
Let's use one of the Views available in the Sakila database: `customer_list`



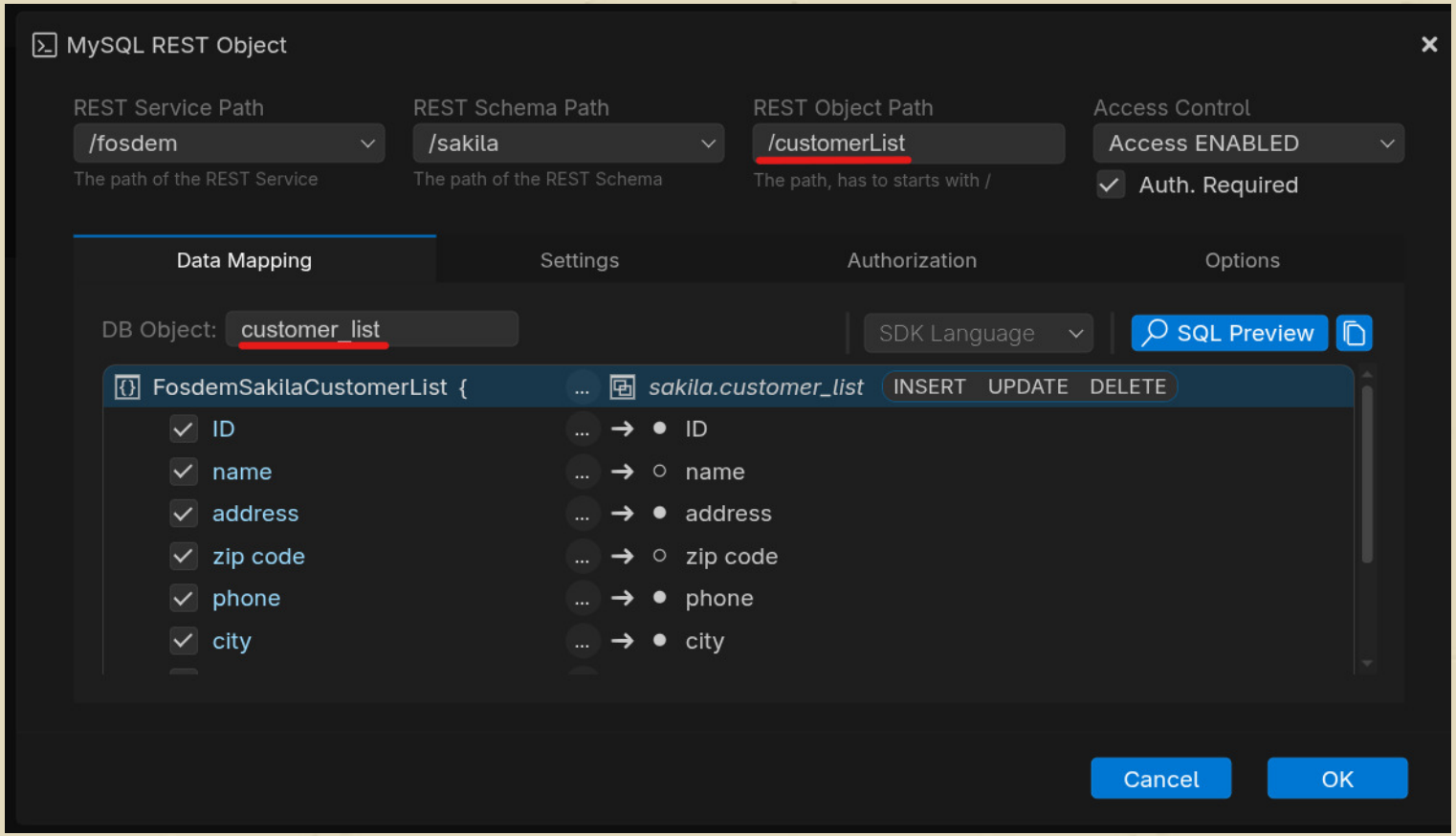
Using Views

Let's use one of the Views available in the Sakila database: `customer_list`





Using Views (2)

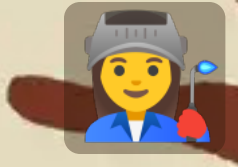


Using Views (3)

Create a Python program to use the view and display the name and the county of customers.

Don't forget to export again the SDK !!

Using Views (3)



Create a Python program to use the view and display the name and the county of customers.

Don't forget to export again the SDK !!

```
$ python fosdem_view.py
VERA MCCOY, Afghanistan
MARIO CHEATHAM, Algeria
JUDY GRAY, Algeria
JUNE CARROLL, Algeria
ANTHONY SCHWAB, American Samoa
CLAUDE HERZOG, Angola
MARTIN BALES, Angola
...
```

Using Views (4)

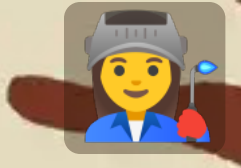
fosdem_view.py

```
from  sdk.fosdem import *
my_service = Fosdem(verify_tls_cert=False)
async def main():
    await my_service.authenticate(
        username = "fosdem_rest_user",
        password = "Fosd3m2026!",
        app = "MRS",
    )
    records = await my_service.sakila.customer_list.find()
    for record in records:
        print( record.name + ", " + record.country)

asyncio.run(main())
```

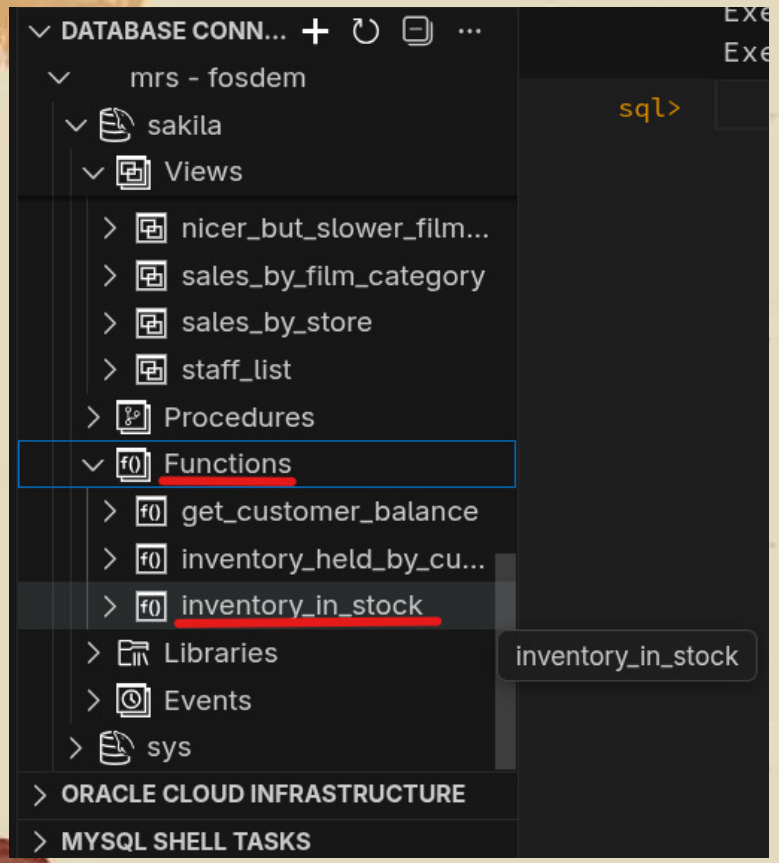
Using Routines

Now we will use a function from the Sakila database: `inventory_in_stock`

Using Routines

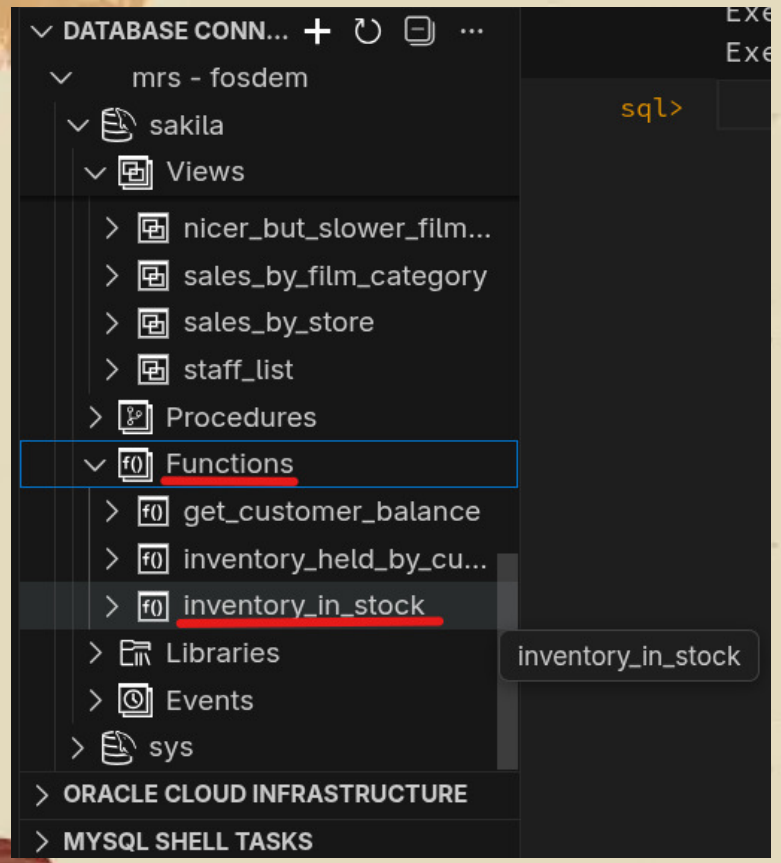


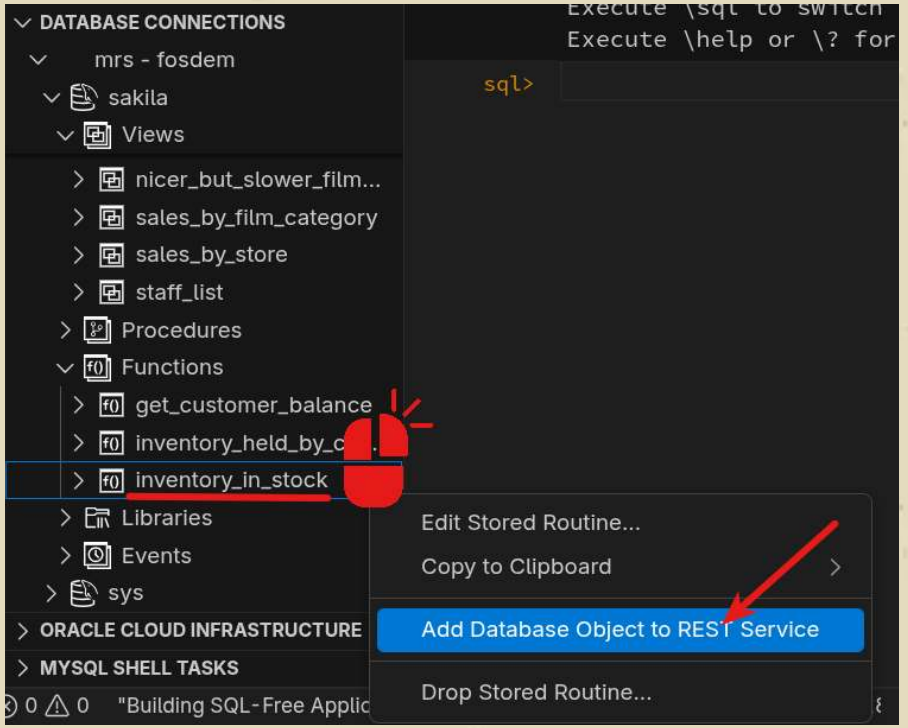
Now we will use a function from the Sakila database: `inventory_in_stock`



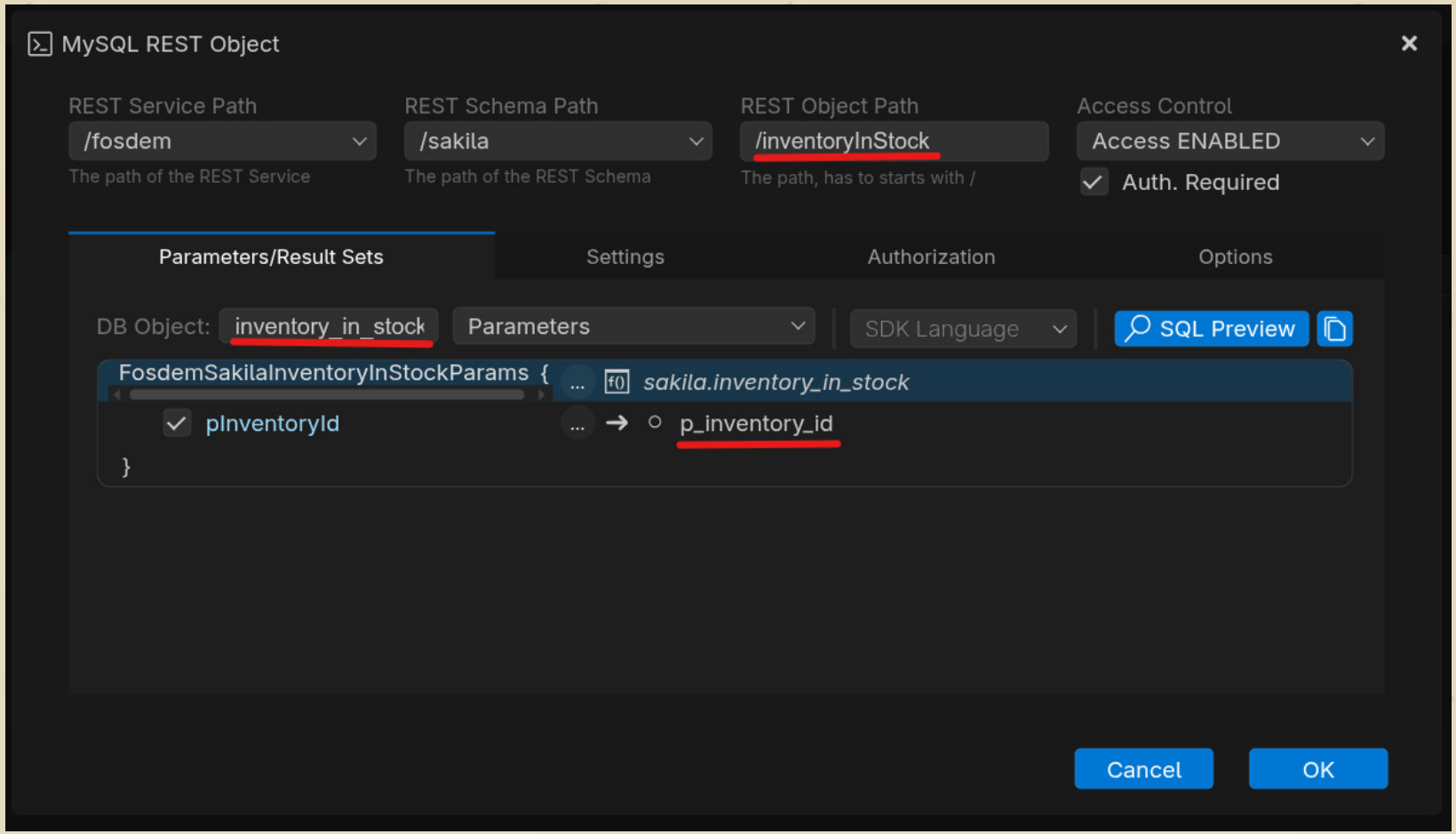
Using Routines

Now we will use a function from the Sakila database: `inventory_in_stock`

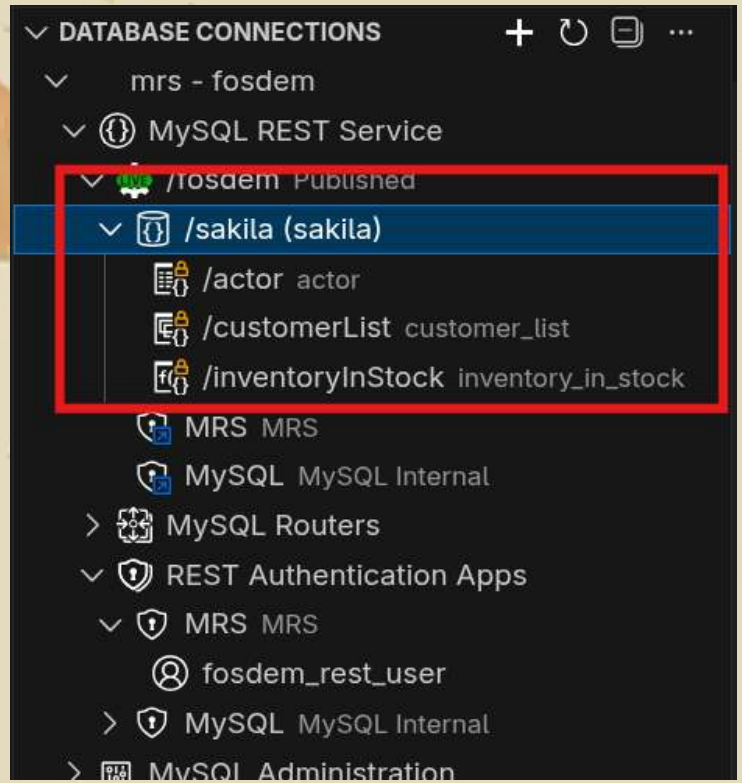




Using Routines (2)

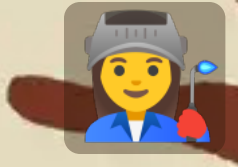


Using the SDK



Don't forget to export the SDK at every changes in your service!

Using Routines (3)



Now create a Python program to call the function and display if the movie with a given id is in stock or not.

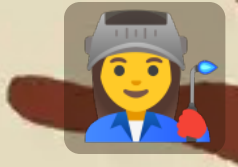
```
$ python fosdem_function.py
Movie id? 1
Movie in stock
Movie id? 6
Movie not in stock
Movie id?
```

Using Routines (4)

fosdem_function.py

```
from  sdk.fosdem import *
my_service = Fosdem(verify_tls_cert=False)
async def main():
    await my_service.authenticate(
        username = "fosdem_rest_user",
        password = "Fosd3m2026!",
        app = "MRS",
    )
    movie_id = input("Movie id? ")
    records = await my_service.sakila.inventory_in_stock.call(p_inventory_id=int(movie_id))
    if records == 0:
        print("Movie not in stock")
    else:
        print("Movie in stock")

while True:
    asyncio.run(main())
```

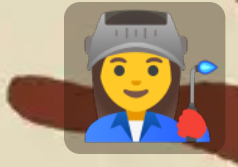
Challenge for the Community Dinner ;)



Run the following query:

```
SQL > update film set language_id=5 where film_id = round(rand()*100);
```

Challenge for the Community Dinner ;)



Run the following query:

```
SQL > update film set language_id=5 where film_id = round(rand()*100);
```

Create an unauthenticated RESTful Endpoint that will return the list of films in a given languageId (5).

Share your ❤️ to MySQL

#mysql #MySQLCommunity

slack

Join our slack channel!

bit.ly/mysql-slack

Questions ?