**Query Metadata Parser Documentation**

**Problem Statement**

The Query Metadata Parser solves many common and difficult SQL, including the following:

1. Checks SQL syntax offline so that you can validate syntax without connecting to a database.
2. It does an in-depth analysis of SQL Query passed as an input and come up with SQL parse tree.
3. It helps in preventing SQL injection by giving information of how the predicates are used in the query.
4. It helps in identifying the how the table objects are used in the extraction of data from Datawarehouse.

**Solution**

The Query metadata parser is used to deconstruct any sql query into individual components, where each component refers to some section of query construct. This help the analyst to get the catalogue ready with the information of all the entity, attributes which are being used frequently by users. The QMP has exposed its method and have UI where user can directly upload the query and get the parsed metadata delivered on the screen itself and also it has functionality to download the deconstructed query. The various methods which are exposed for users are:

1. get\_query\_type

This method will give the information any the query type which could be DML or DDL or TCL statements

1. get\_schema\_list

This will list out all the schemas where the objects are residing in database from the input query

1. get\_table\_list

This will list out all the tables involved in the sql query passed to the engine

1. get\_column\_list

This will list out all the columns involved in the query along with its parent tables

1. get\_filter\_condition\_list

This will list out all the predicates used in the query.

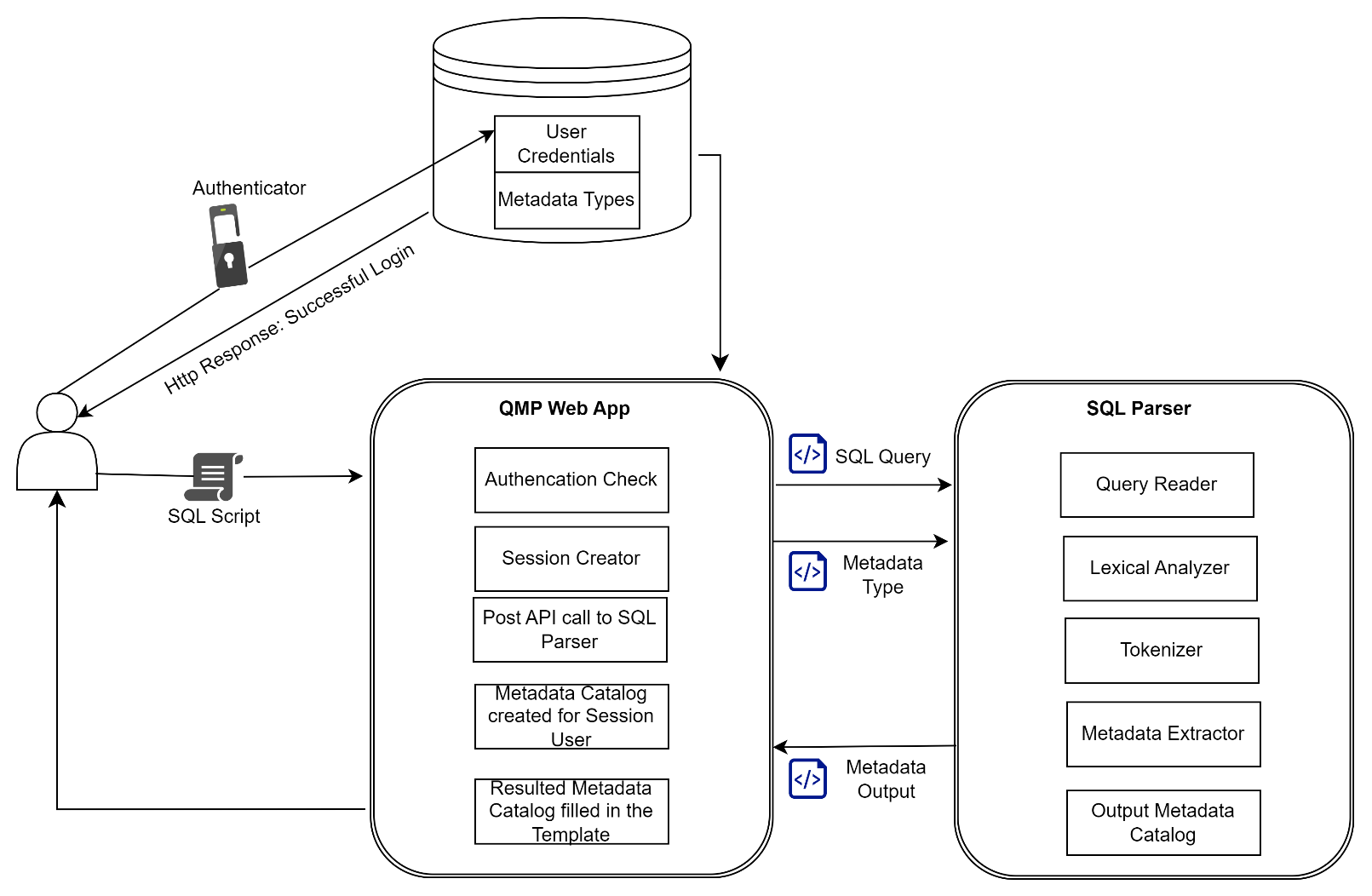
1. get\_joins\_used

This will list of “kind of joins” used to join the various tables used in the sql query passed by user to system

1. get\_all\_metadata

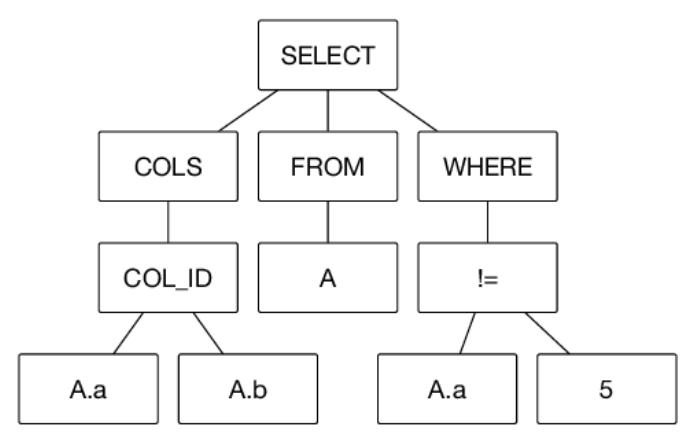
This will list out all the metadata which include tables, columns, predeicates, joins, schemas etc.

**Architecture Diagram**



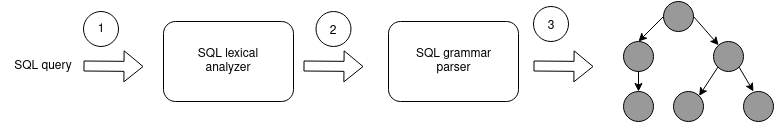
**Parser**

The parser deconstructs the SQL query based on the SQL syntax, identifies each of the query components, and then creates an Abstract Syntax Tree in a hierarchical format.



**Lexical Analysis/Tokenization**

The query parser breaks the entire sql statement to various tokens, each token refers to one or other part of query. It forms a list of tokens, which will there after used to do the lexical analysis.



**Usage of Query Metadata Parser**

1. Help in identifying the mostly used tables in the data warehouse.
2. Help in knowing how each table are being queried and what kind of predicates are used in general to extract the data
3. Help in knowing how various tables are joined with each other.
4. Help in identifying various query pattern.

Having all above metrics available below question can be answered:

1. Popular tales and the attributes used in reporting.
2. Data set schemas metadata for perf optimization.
3. Joins information which will help in extablish the need for indexing and query tunning

**Prerequisites**

Tools/Technologies used

1. Python
2. SQLite Database
3. Jinja Framework
4. Bootstrap Framework
5. HTML, CSS

Required Python packages

1. Flask
2. Sqlite3
3. Sqlparse

Project Structure

---------QueryMetadataParser  
 ├── \_\_init\_\_.py # setup your app  
 ├── query\_parser.py # Code for SQL Query Parsing  
 ├── create\_sqlite\_db.py # Database Creation   
 ├── app.py # Flask Code

├── query\_parser.py # Query Parser Code  
 ├── db.sqlite # your database  
 └── templates  
 ├── display\_all\_metadata.html # display all type of metadata  
 ├── display\_metadata.html # display selected metadata  
 ├── index.html # show the login form  
 ├── successful\_file\_upload.html # SQL script upload to directory   
 └── successful\_login.html # page after login

└── type\_of\_metadata.html # select type of metatdata

└── upload\_sql\_script.html # form to upload sql script

└── style.css # css

└── UploadedSQLScripts

└── SQL\_Script\_1.html # sql scripts uploaded in this directory

└── SQL\_Script\_2.html # sql scripts uploaded in this directory

**Technical Implementation**

1. Prepare your Environment
2. Create a virtual environment in python

virtualenv venv sql\_parser

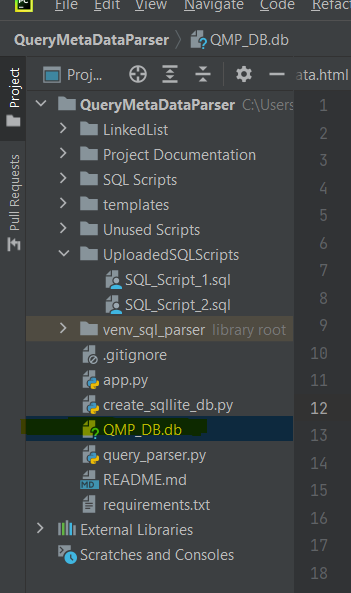
1. Install the python packages in the virtual environment described in the pre-requisite section.

flask  
sql-parser

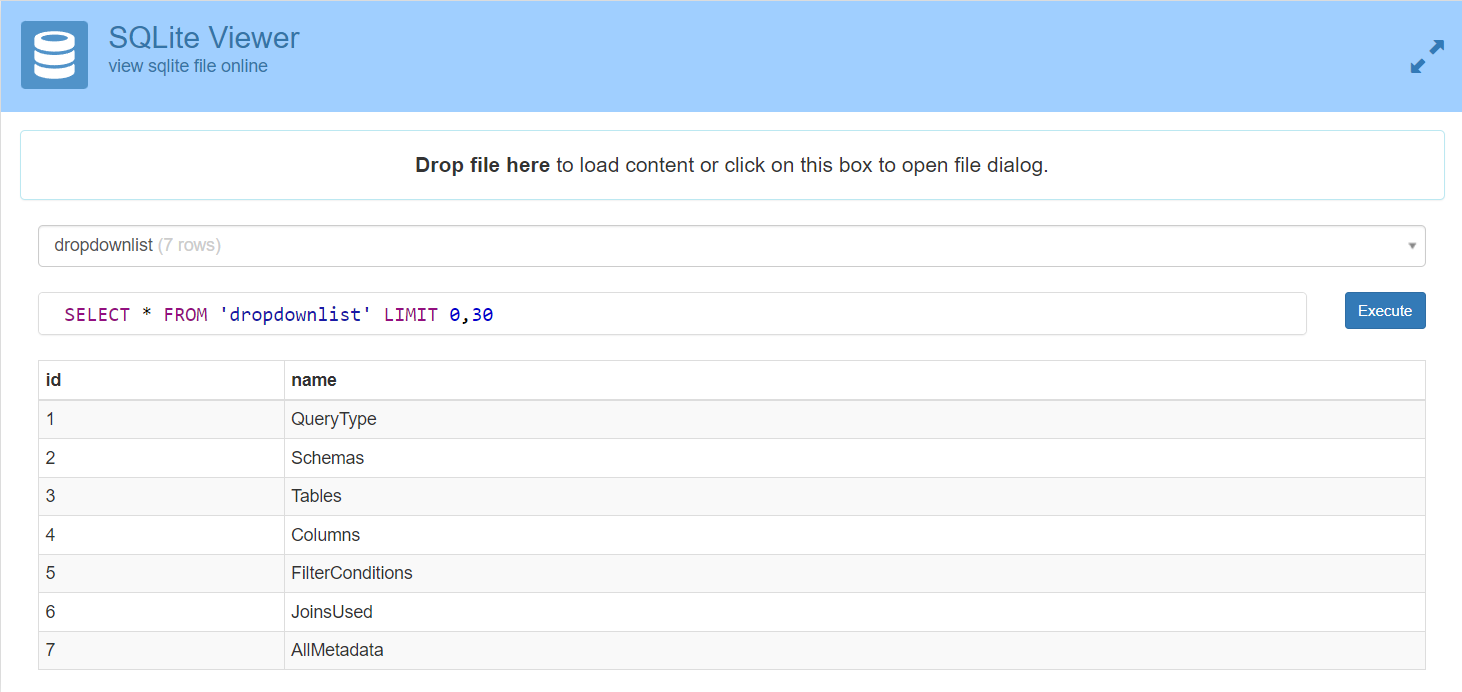
sqlite3

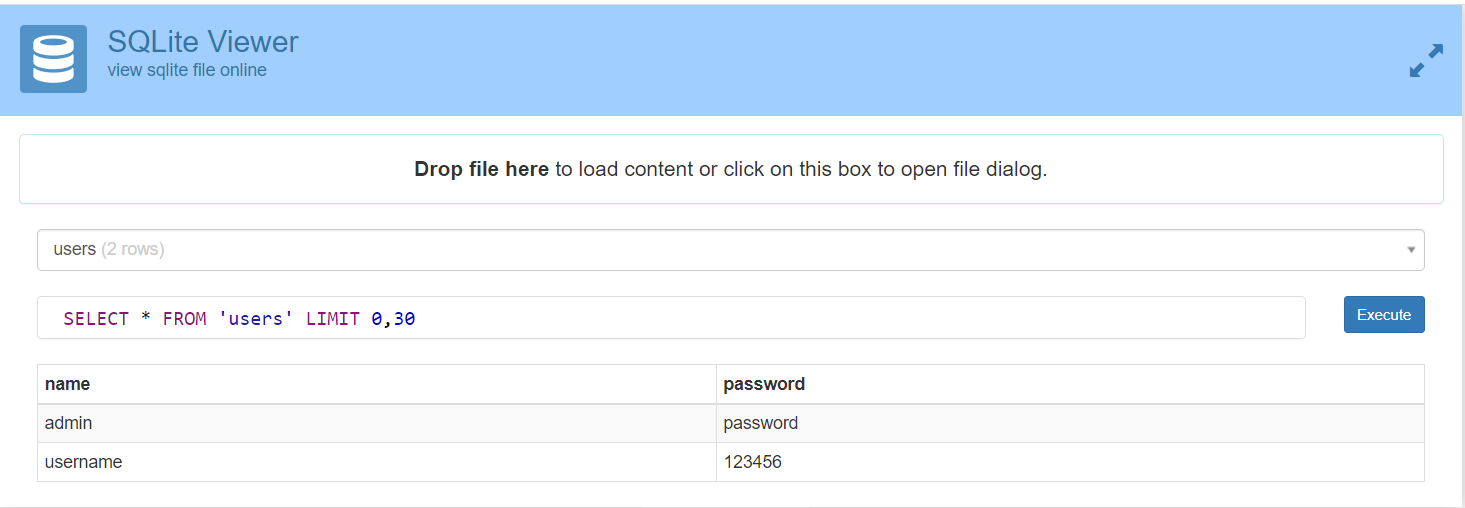
To install packages pip install package\_name or you can create a new file requirements.txt (this file will contains one package name each row) and install all packages once using : pip install -r requirements.txt

1. Create project structure as mentioned in the pre-requisite section.
2. Create Database
3. Create Sqlite Database – QMP\_DB.db



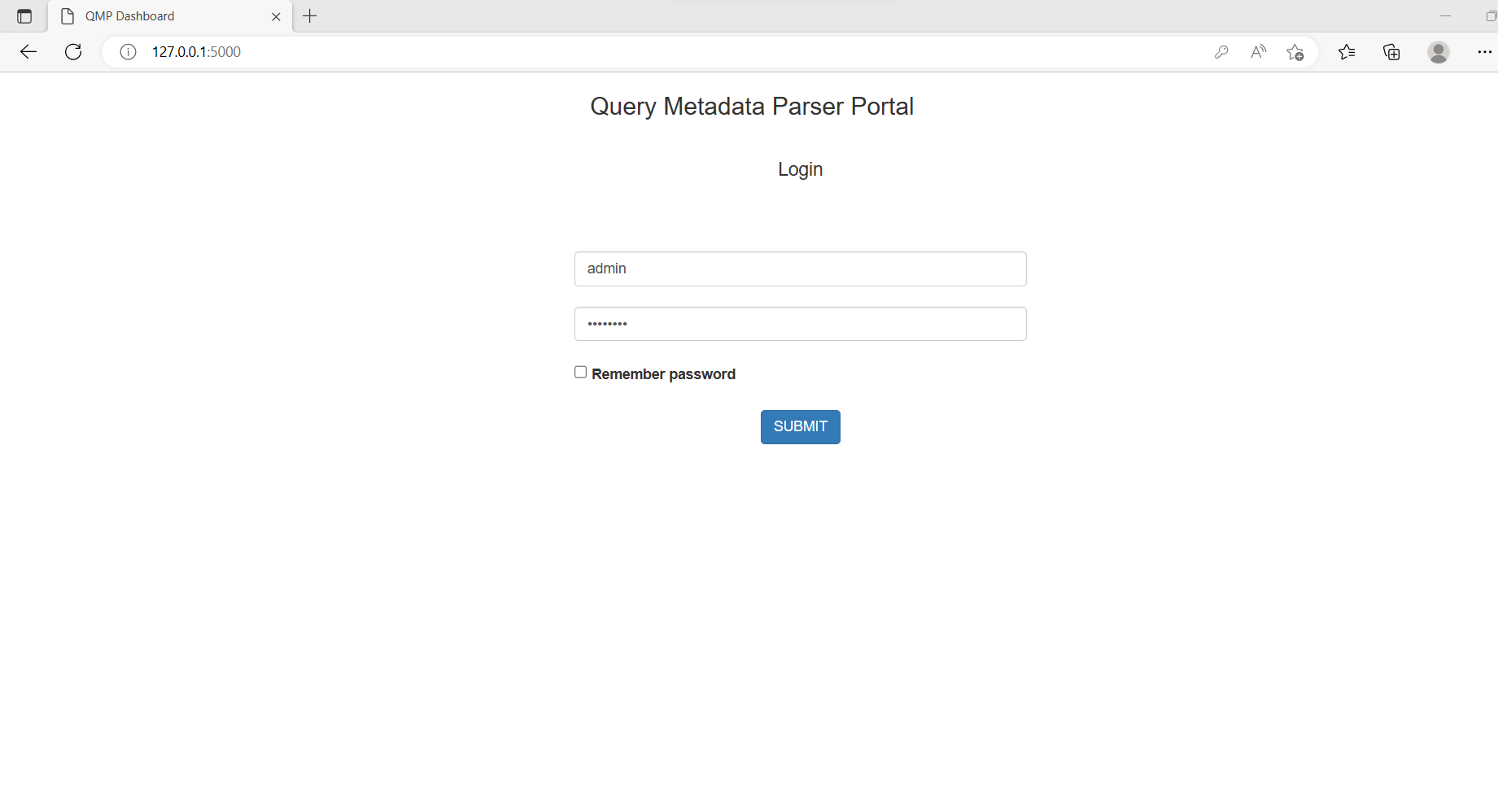
1. Create Tables dropdownlist and users in the database QMP\_DB.db



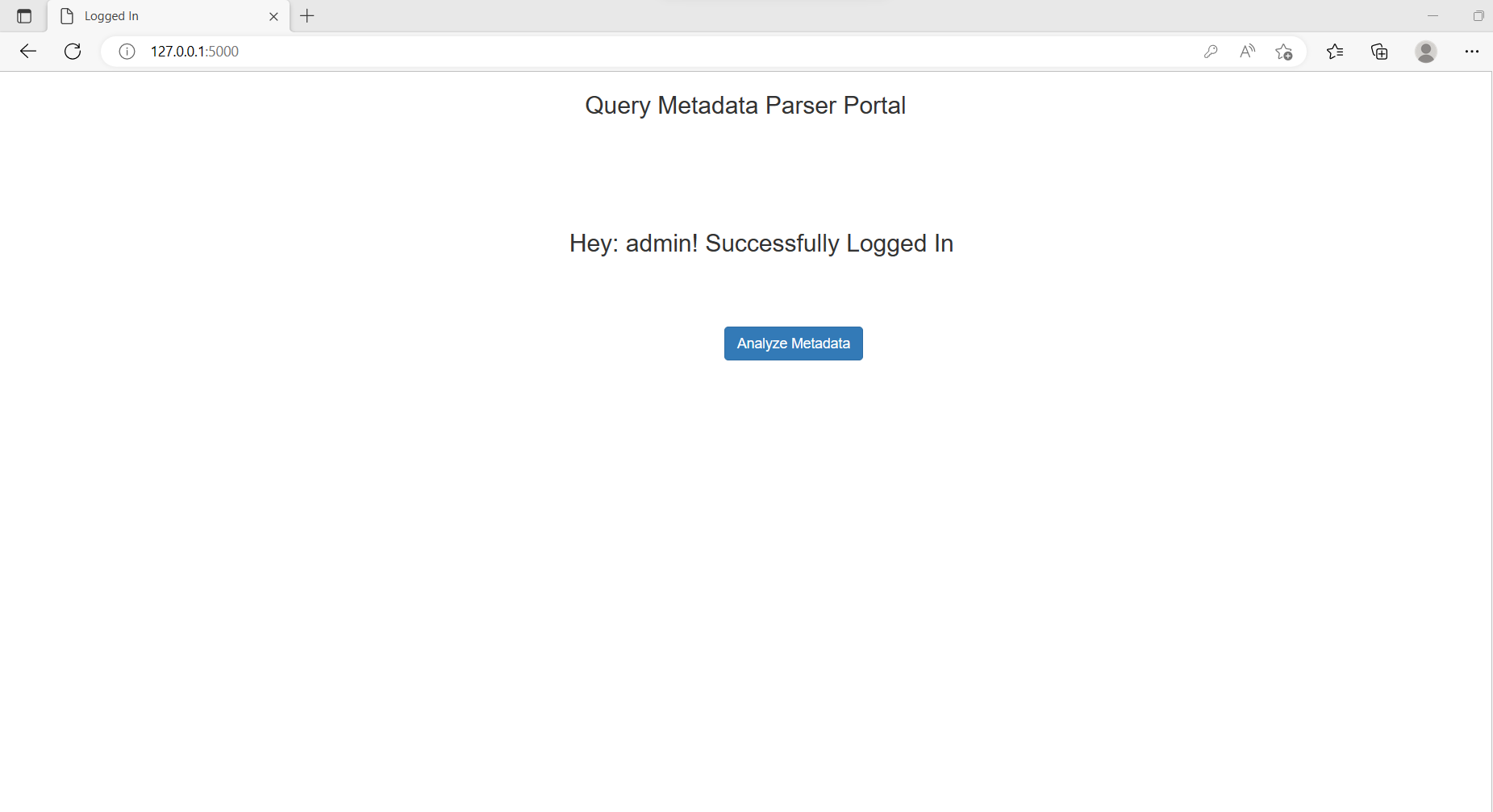


1. Create Web Pages

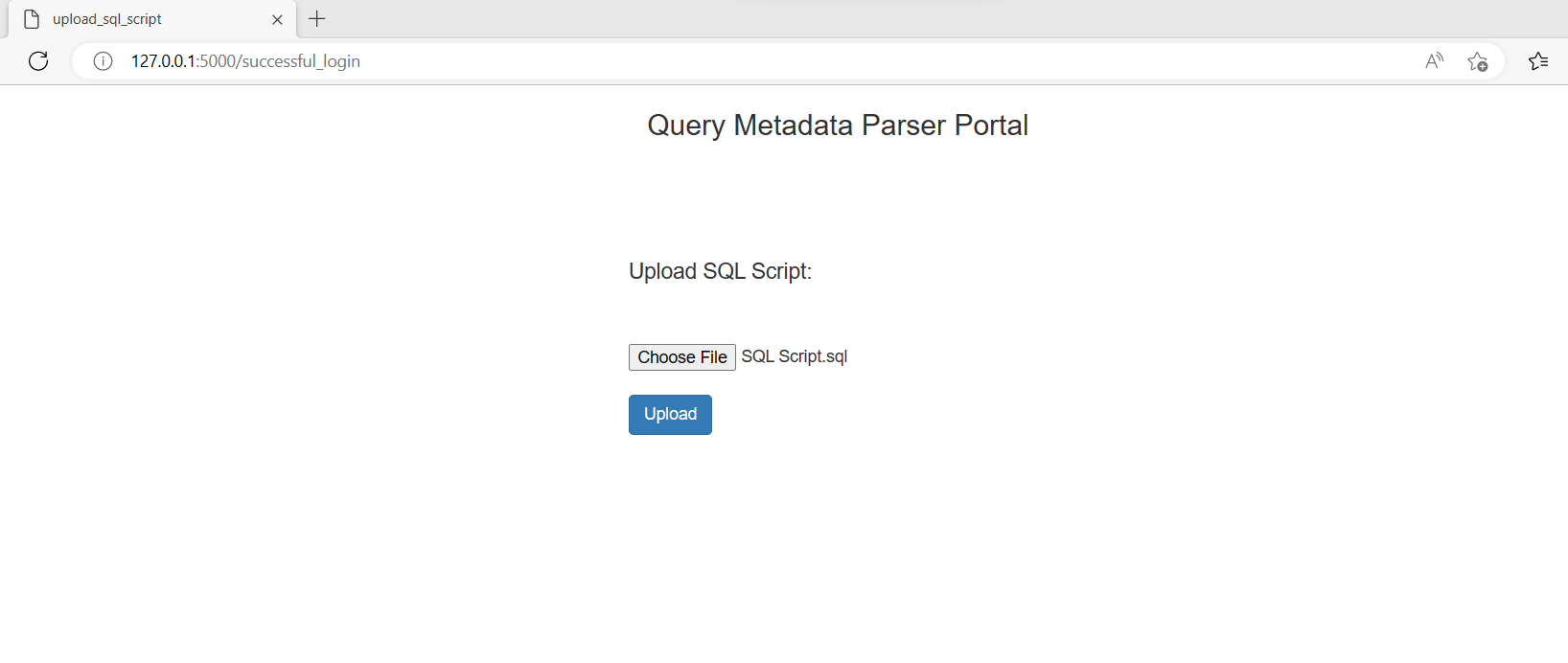
* Enter user details and login to the Query Metadata Parser portal.
* The application will validate the user details with the credentials stored in the Sqlite database and if it matches it will redirect to the next page.

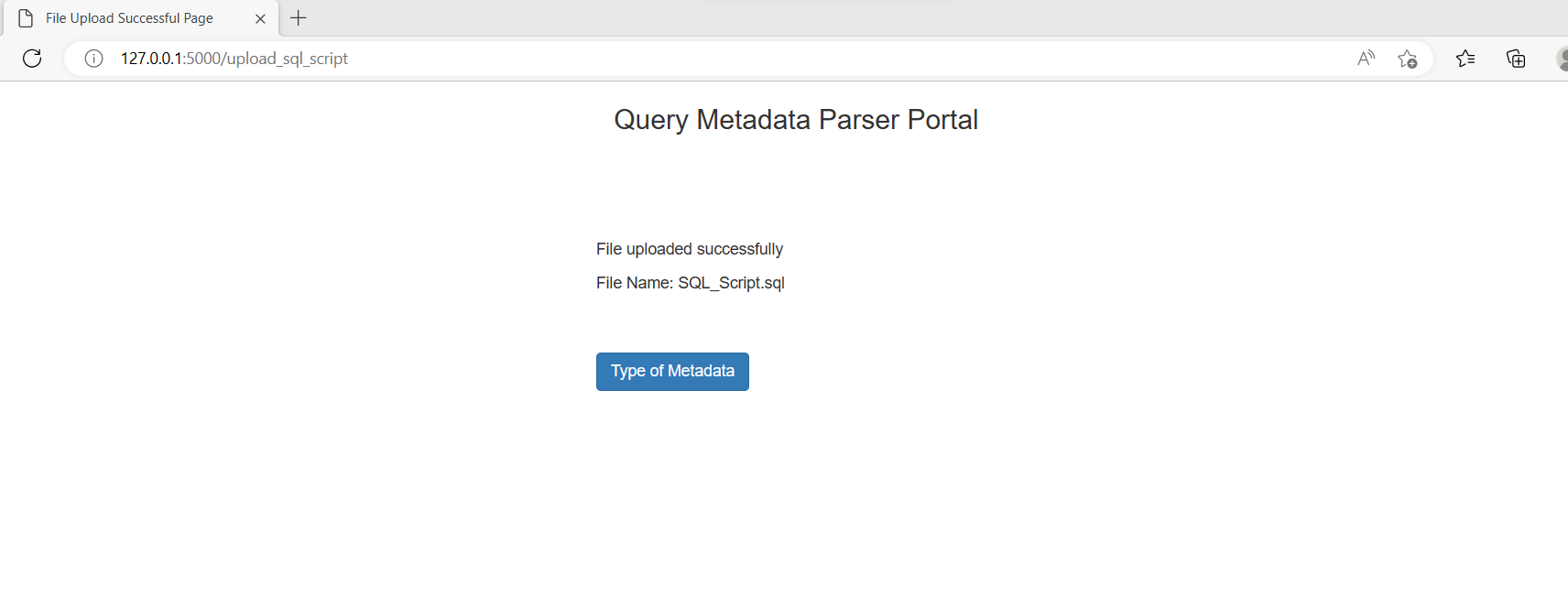


* Click on Analyze Metadata button post successful login.

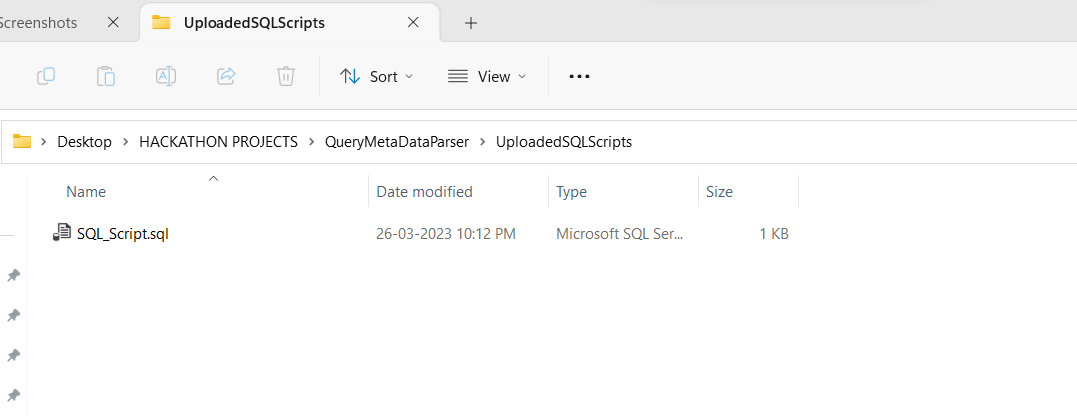


* Choose a SQL Script from your local device and upload it to the portal.

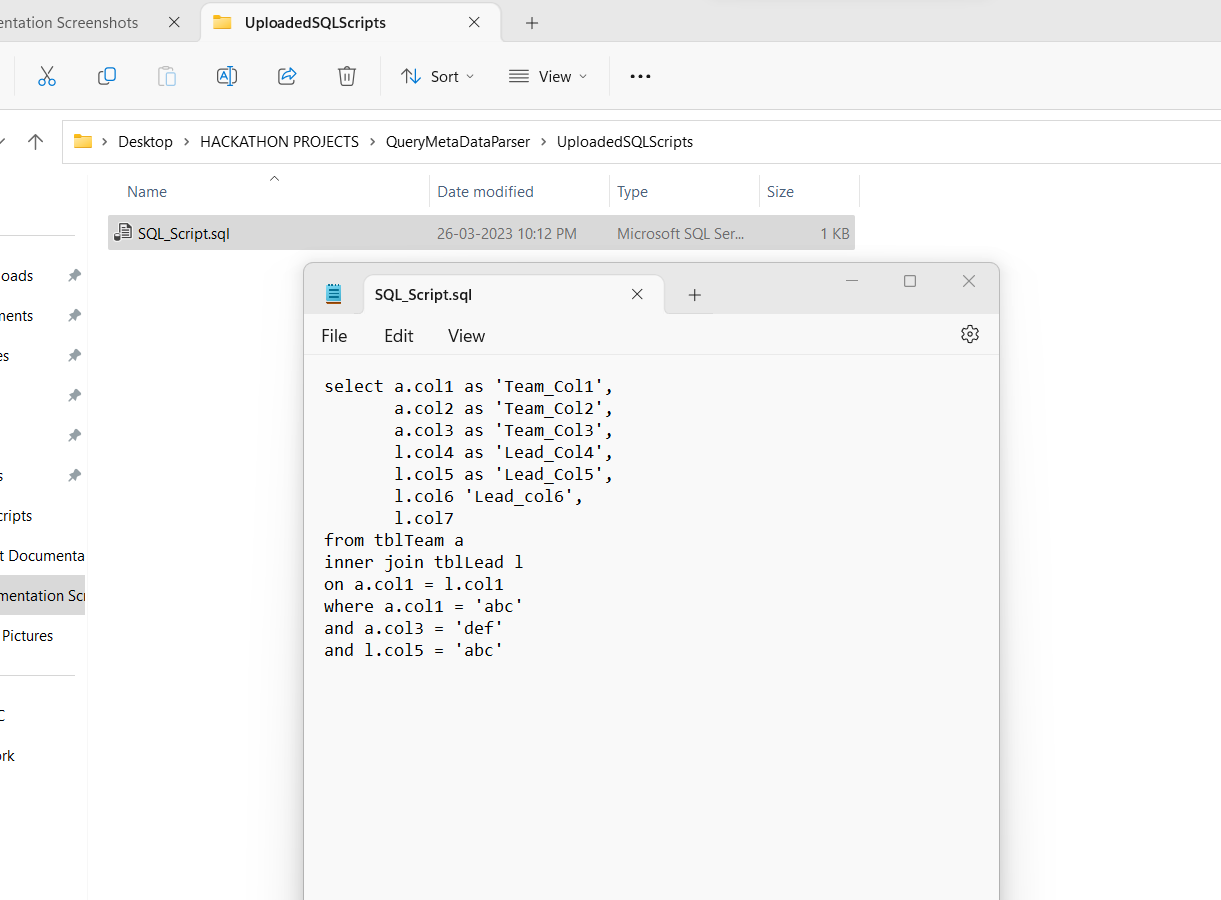




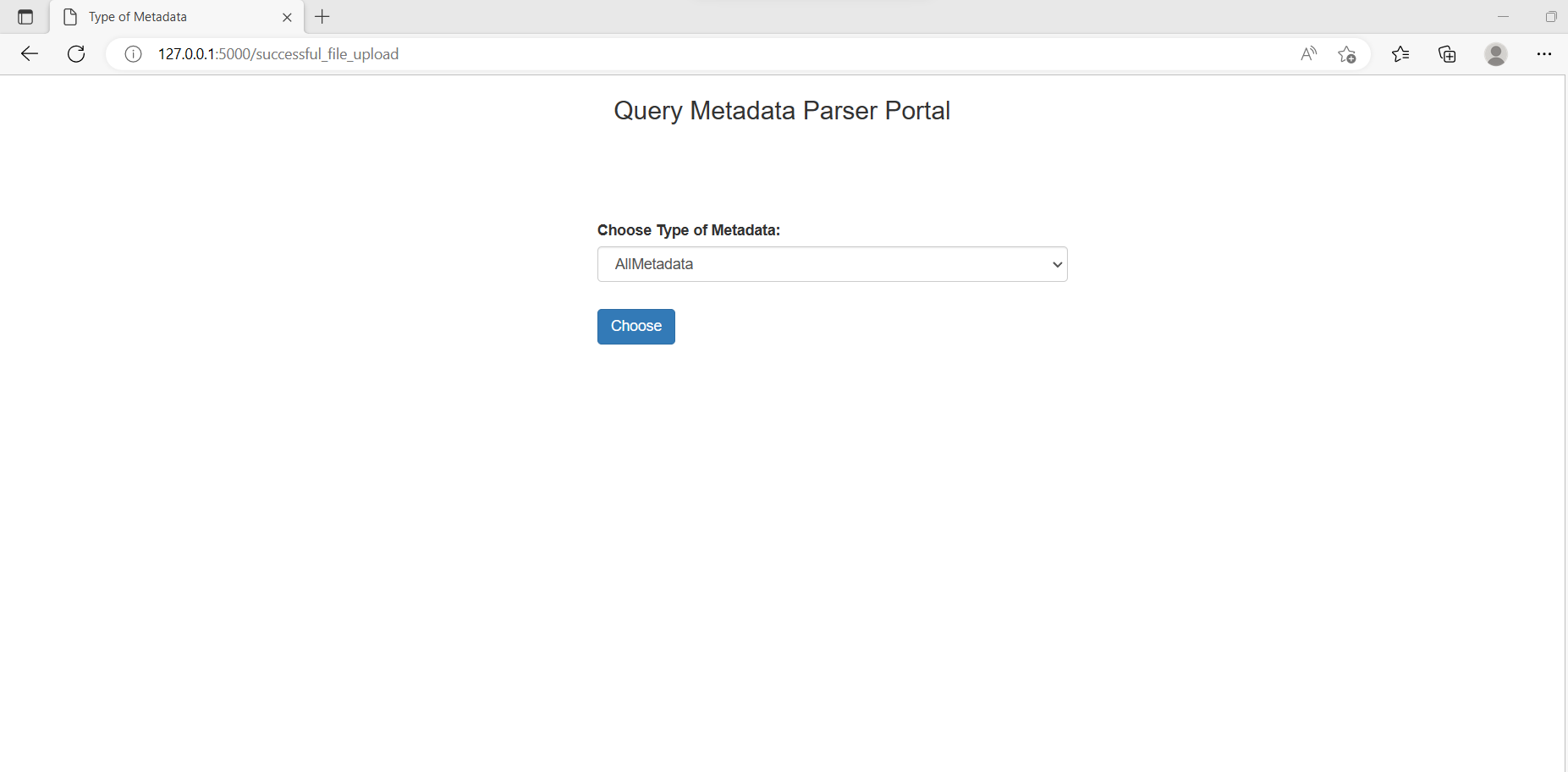
* SQL script uploaded to the Folder “UploadedSQLScripts”

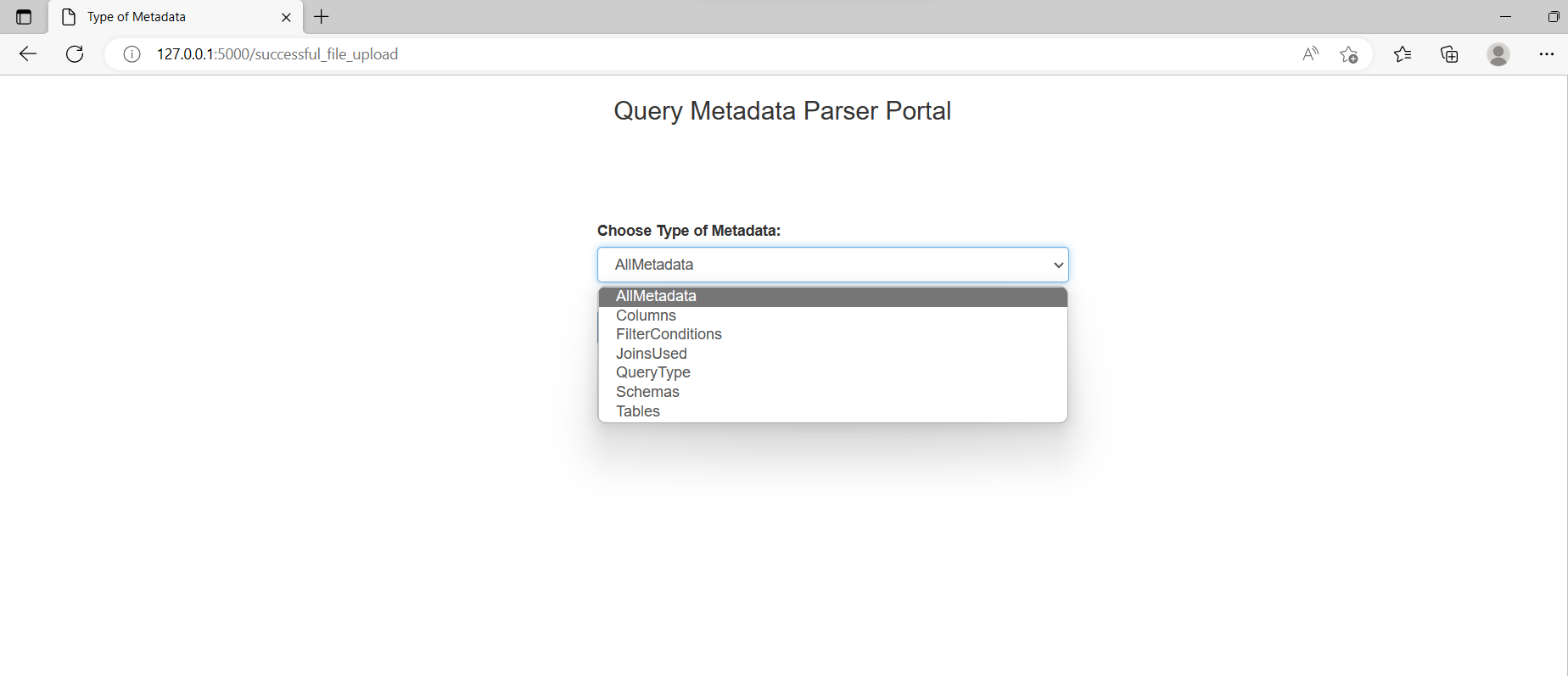


* SQL query present in the user input script.



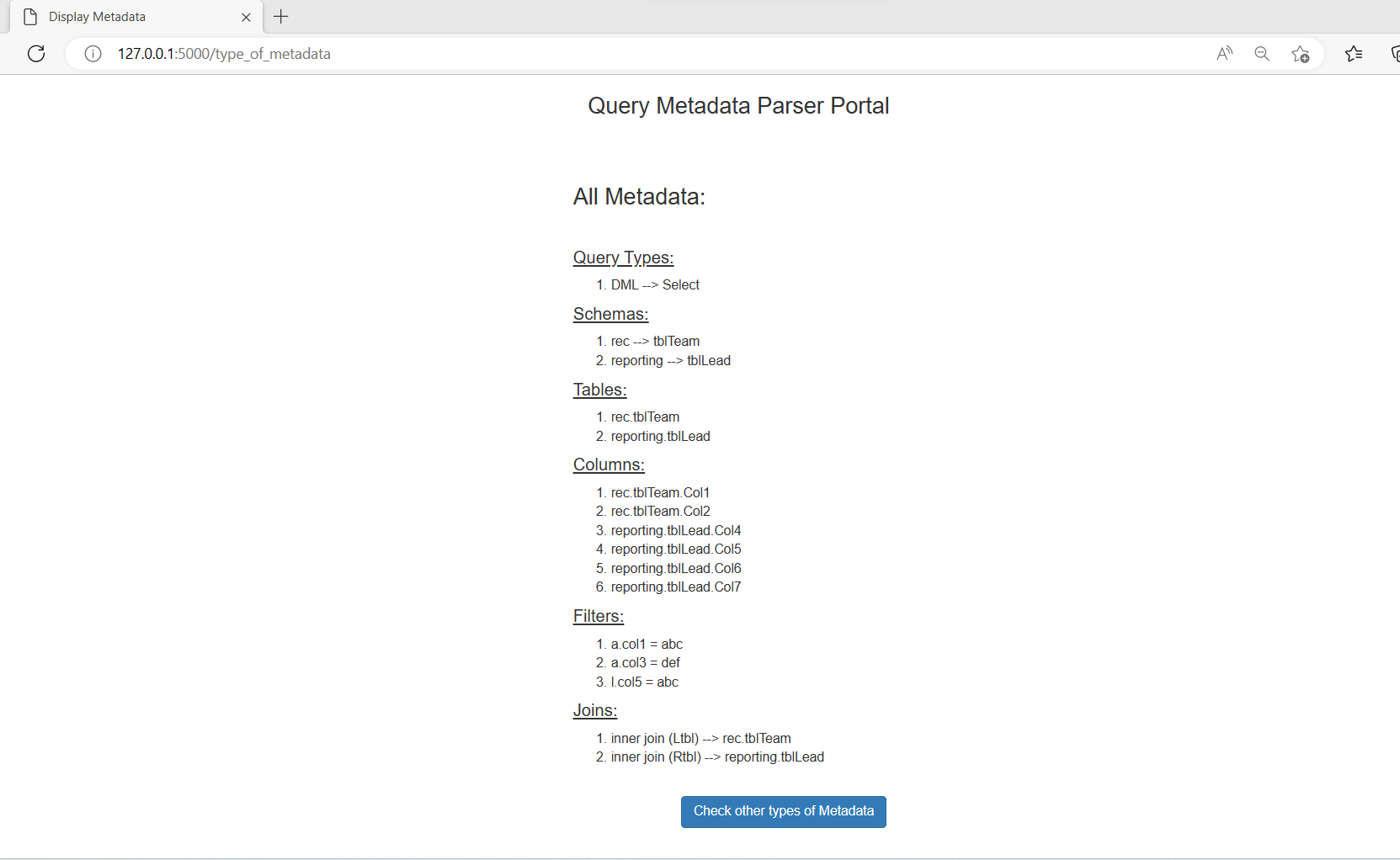
* Choose the type of Metadata that you want to display.



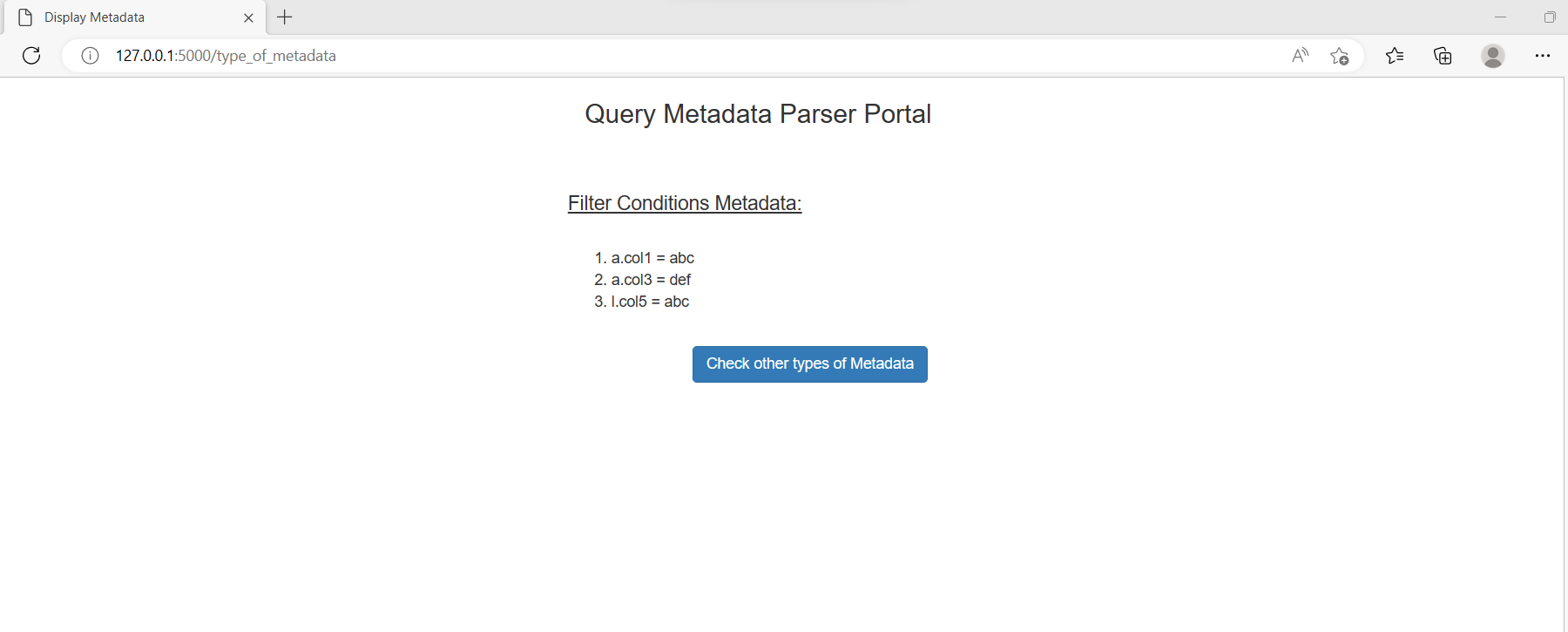


* Based on the Metadata type selection, Metadata information would be displayed. Check other types of metadata by clicking on the button “Check other types of Metadata”.

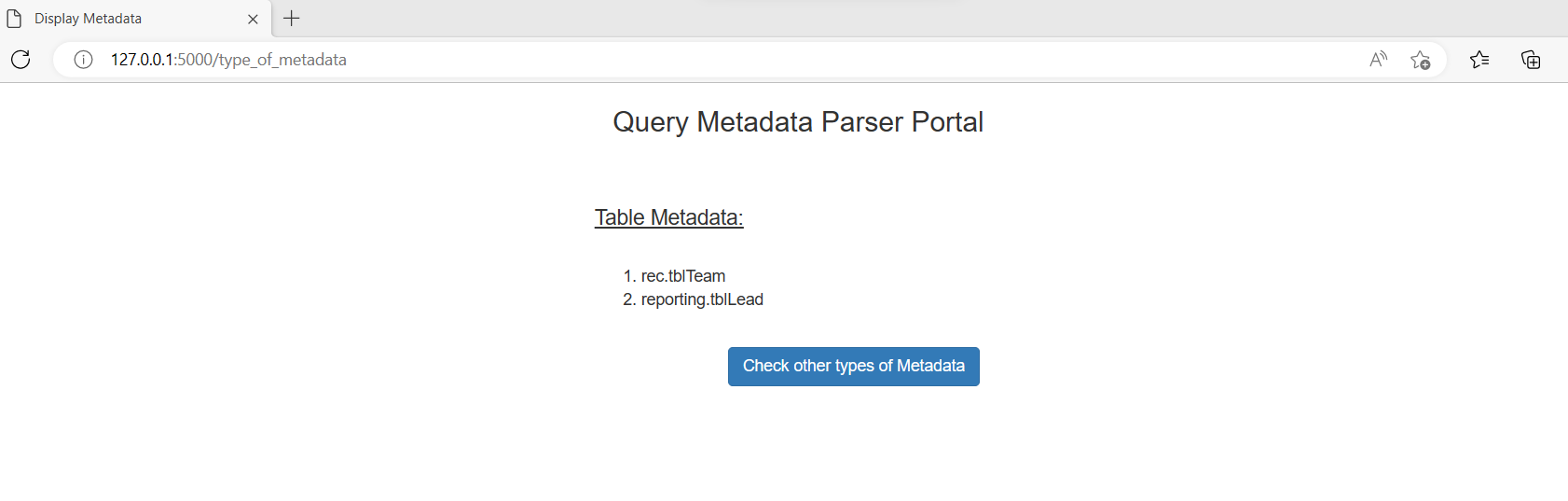
Output for All Metadata



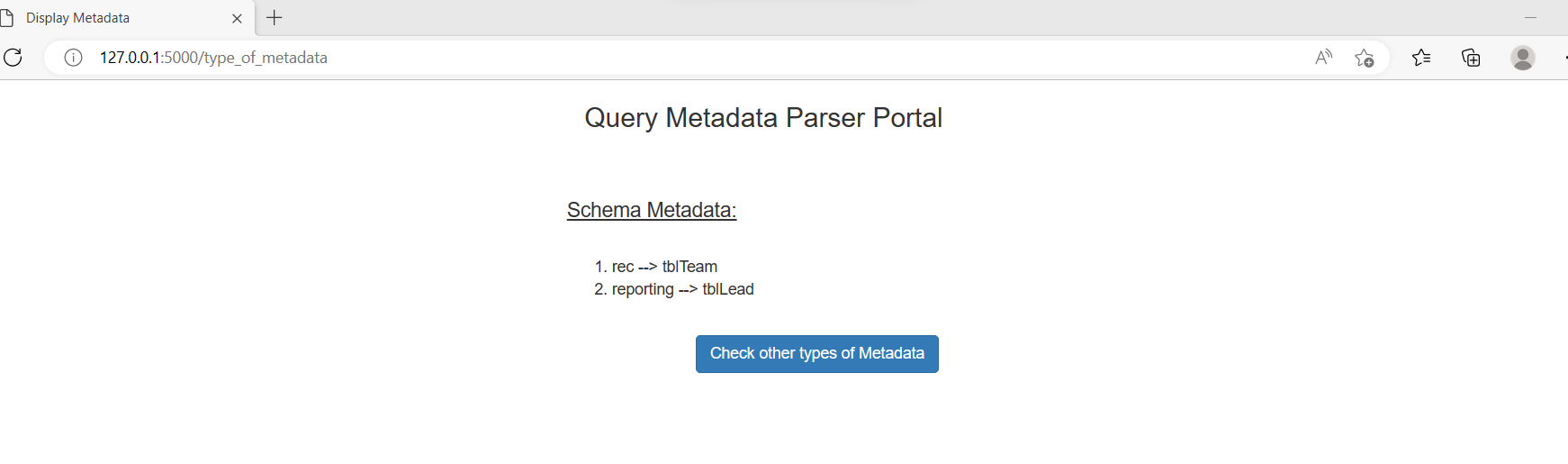
Output for Filter Conditions Metadata



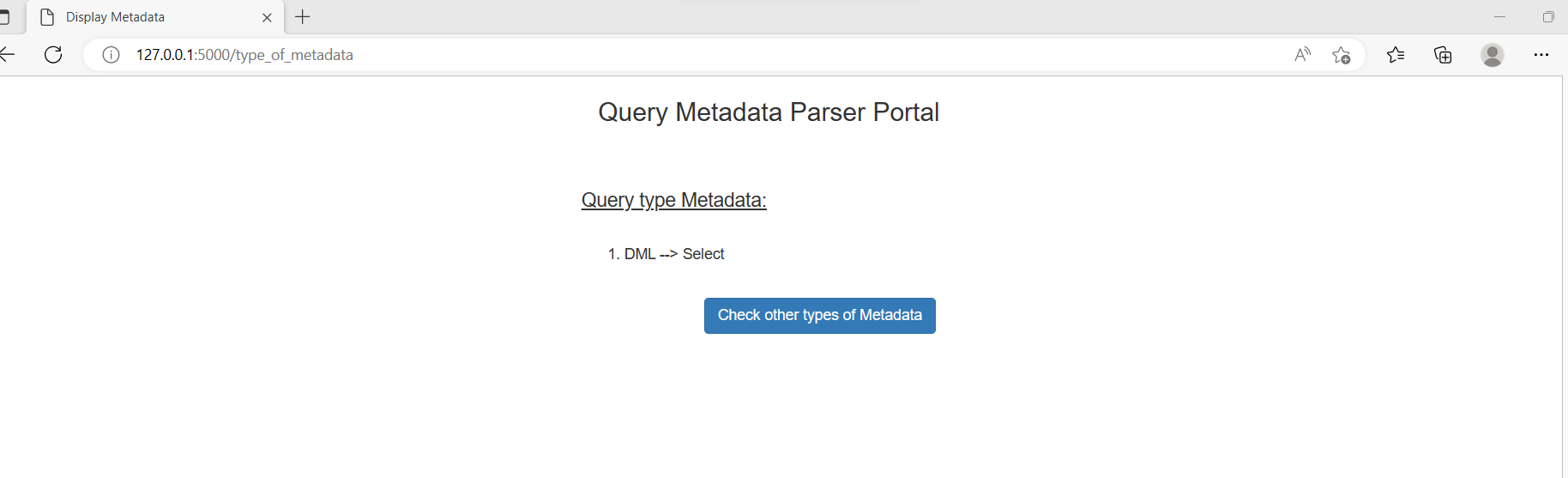
Output for Table Metadata



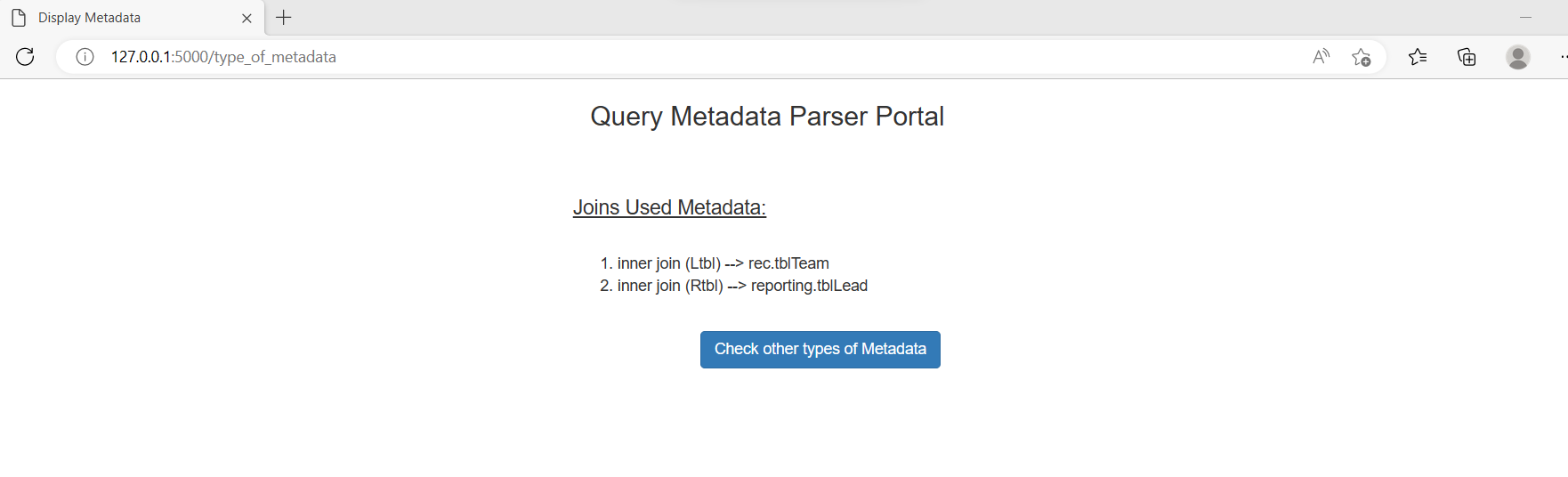
Output for Schema Metadata



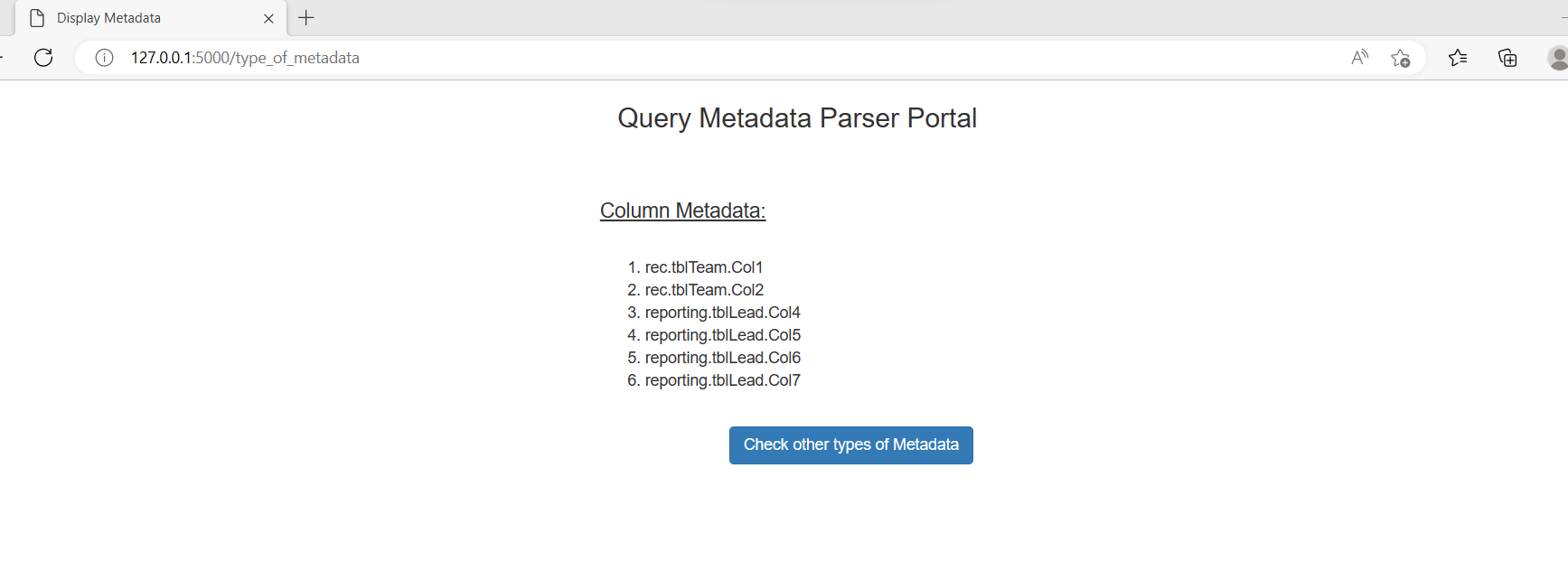
Output for Query Type Metadata



Output for Joins Used Metadata



Output for Columns Metadata



1. Connect Web Pages using the Flask Framework

* Connect to SQLite database.
* Create routing methods to route/redirect web pages based on the GET/POST.

1. Create sql\_parser python script

* Create the sql parser python script which will parse the sql query provided by the user and send the metadata extract back to the webapp.

**Conclusion**

The Query Metadata Parser help in deconstruct the SQL passed as an input to the lowest granularity which will help analyst to take decision pertaining to DB tunning. The above parser can be extended further to connected to the GIT repo and read all the sql and create Metadata catalogue from it. It can also be extended to have metadata catalogue versioned for any changes in the query.

**GitHub link**

* <https://github.com/tariniteam/QueryMetaDataParser>

**Contributors**

1. Harsha Navalkar ( <https://www.linkedin.com/in/harsha-navalkar-00085515b/> )
2. Vikram Mahapatra ( <https://www.linkedin.com/in/vikrammahapatra> )