**Live Table & Writeback Extension**

The LiveTable extension allows a table of data from a SQL database to be embedded in a Qlik Sense app. As selections are made in the Qlik Sense app the table is refreshed automatically from the SQL database. You can configure the extension to allow editing of individual records of the data, and your changes will be written back to the SQL table.

To filter the records listed in the table by selections in the Qlik Sense app, you must add the relevant fields to the list of dimensions in the extension configuration.

As extensions are executed in the context of the user’s browser, a server-side web service is required to execute queries against the database. The extension communicates with the server-side web service using JSON. You could create your own server side web service to process the information sent via JSON, however this extension comes with a Python based web service already developed. The extension therefore has two components to install:

1. The client-side Javascript based LiveTable extension
2. A server-side Python based REST API to query the database(s)

**Installing the Extension object**

1. For Qlik Sense Enterprise, import the LiveTable-lms.zip file
2. For Qlik Sense Desktop, unzip the LiveTable-lms.zip to [my documents]\Qlik\Sense\Extensions

**Installing the Python REST API**

Note that these instructions cover setting up a demonstration environment only. To ‘productionize’ the REST API you should use a scalable web environment. Various options are available depending on where you run the API component; ie. Linux or Windows. You can google for info using “scalable python flask on [windows / linux]”. To run Python apps as a service on windows, you can use [NSSM](https://nssm.cc/).

Installing the REST API on your Qlik Sense Enterprise server for demo purposes (these instructions were adapted from <https://www.youtube.com/watch?v=98JY6MvumVs>):

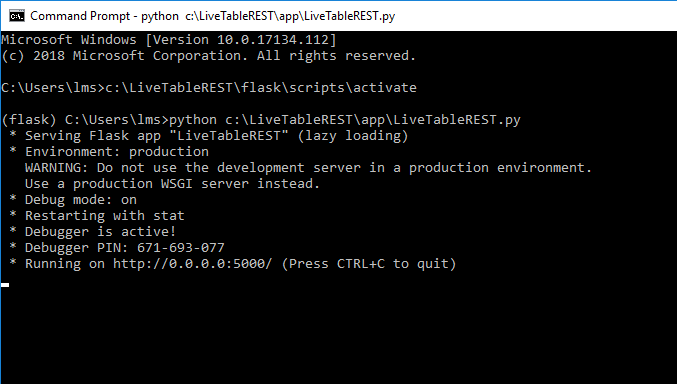
1. Install the latest version of Python3 on your server. I recommend installing into the root folder, for example C:\Python36.
2. Add the path to your Python Scripts folder to your PATH variable (eg. C:\Python36\Scripts)
3. Open a command prompt on your server
   1. Run “pip install virtualenv”
   2. Run “cd \” to change to the root directory
   3. Run “mkdir LiveTableREST” to create a directory for the project
   4. Run “cd LiveTableREST”
   5. Run “virtualenv flask” to create a virtual copy of your Python environment in a folder called flask
   6. Run “flask\scripts\pip install flask” to install the flask library for python
   7. Run “flask\scripts\pip install pysqlite” to install the SQLite library for python
   8. Create a subfolder named app under the LiveTableREST folder. Paste the LiveTableREST.py file into this folder.
   9. Create a subfolder named testdata under the LiveTableREST folder. Paste the DemoData.sqlite file into this folder.

To activate your Python virtual environment and then run the REST API web service, at a command prompt run the following two commands:

C:\LiveTableREST\flask\Scripts\activate

Python C:\LiveTableREST\app\LiveTableREST.py

NOTE: The above commands will run the REST API under a flask development web server. As mentioned above a scalable web server should be added to make this a “productionized” solution.



The flask development web server is exposed as a HTTP REST endpoint. In a production environment you would run the flask REST application as a back-end service, not exposed to your users. You would then run a production web server as a front-end and proxy calls to the API across to the URL and port the flask REST application could be reached on. In that scenario you could add a certificate to the production web server to encrypt all traffic over HTTPS.

If you wish to flask development web server as a HTTPS end point for demo purposes, then you can follow the [instructions here](https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https) to make the flask app run under HTTPS. This involves installing the Python library pyopenssl (eg. run “flask\scripts\pip install pyopenssl”) and changing the last line of the REST API code to read

myApp.run(host= '0.0.0.0', port=5000, debug=True, ssl\_context='adhoc')

**Testing The API**

You can test your REST API using the SQLite database that comes with the extension. This is the file DemoData.sqlite. No database engine is needed and python support for sqlite should have been installed using the steps above (flask\scripts\pip install pysqlite).

1. Open the file LiveTableREST.py and look for the function createConn(connection)
2. Find the line

conn = sqlite3.connect('C:\\liveTableREST\\testdata\\DemoData.sqlite') and ensure that the path is correct for your installation.

1. Run the REST API using the instructions above.
2. Navigate to localhost:5000 using your browser. You should see a JSON message stating that no data was received by the REST API. This is because no parameters were submitted to build a query.
3. Navigate to localhost:5000/test using your browser. This API endpoint has the various parameters hard coded to the SQLite database and the Customers table. You should see a list of records returned in JSON format. This indicates that the REST API is working.
4. Now you should test the extension component in a Qlik Sense app. Import the Qlik Sense demo app that comes with the extension. Ensure that the extension is also imported.
5. Open the app to the Customers (Writeback) sheet. The extension is configured to pull the data from the REST API at localhost:5000 using the SQLite database. The extension should display if you are using a browser on your actual Qlik Sense server. However localhost is not visible to the browser if you access Qlik Sense from another PC. In that case you will need to edit the API endpoint to use the server name or IP address that you access you Qlik Sense server from. (Ensure you add the port name to the URL as well). If you install the REST API on another server you will need to use that server’s name or URL.

**Accessing Your Data**

To access your databases from the REST API you will need to edit the LiveTableREST.py file to add connection details. Ensure that the necessary Python libraries are installed for your database. You will need to google how to install these, but the same approach can be used as for the SQLite libraries. You just need to provide the correct package name when running pip install.

1. Open the file LiveTableREST.py
2. Add an import statement for the Python library you installed at the top of the page. Look for where the SQLite library is imported and copy the same syntax. (Note that a commented example is given for a MySQL database).
3. Look for the function createConn(connection) Note from the code how a connection to the SQLite demo database is created when the parameter ‘SQLITE’ is passed by the extension. Duplicate this section of code for your particular database. (Note that a commented example is given for a MySQL database). You can use any text as the parameter to reference your database connection - as long as you use the same text in the extension configuration
4. Add the extension to a Qlik Sense app. It actually does not matter what data is in the app. The data must match your database only if you want to filter the live table using fields in the Qlik Sense app.
5. Configure the extension as follows:
   1. **URL for REST API** must be the URL (and port) to your REST API
   2. **Connection** must be the text parameter in your REST API that references your database
   3. **Name of table** must be the table you want to pull data from
   4. **List of field names** can be left blank. If you only want to pull in certain fields from the table then enter the column names here with each separated by a comma.

**Adding the extension to a Qlik Sense app**

Once you have the REST API configured to access your data you can begin to use the extension in your Qlik Sense apps. Drag the extension object onto your sheet and configure the following options:

* Dimensions
  + Add the list of fields / dimensions you want to use to filter the list of records displayed
* Measures – This is not used
* Sorting - This is not used
* Appearance / Source Data – These details are used to query the REST API to display data
  + URL for REST API – This is the URL and port where the REST API is running. If using the Python REST API code then you can test this URL independently with your browser.
  + Connection – You can embed multiple connections in the REST API and reference each with a particular string. Enter the string which matches the connection details you want to leverage in the REST API code.
  + Name of table – This is the name of the table in the SQL database you want to query.
  + List of field names – If you don’t want to display all fields from the SQL table in Qlik Sense, then enter a comma separated list of field names here.
  + Max Rows – So as not to return a million rows over the REST API, you need to enter a logical limit to the number of rows that will be returned. (The REST API is currently not designed to page through large numbers data).
* Appearance / Functionality – This is where additional functionality such as writeback can be configured.
  + Allow Edit – Check this to allow the user to edit the data via the extension & REST API.
  + Key field – You must have a single key field in order to successfully edit data. Enter te key field name here.
  + List of edit able fields – Enter the list of fields you want to allow editing as a comma separated list here. If left blank, all fields other than the key field will be editable. If you supply a list of fields, the Key field must be included.
* Appearance / Debug – These options will help with debugging the extension and REST API
  + Display JSON - If you want to see the JSON sent back and forth then tick this. The JSON will be displayed in the browser console (under developer tools). This can be useful if you want to develop your own REST API.
  + Display SQL – If you are running the REST API in a visible console window and want to see the resulting SQL displaed, then tick this checkbox.