Installing the SQL language server extension

JupyterLab leverages the VSCode's Language Server Protocol enabling JupyterLabe to leverage many of the Language Servers developed for VSCode. This project is the JupyterLab extension which enables this.

GitHub - krassowski/jupyterlab-lsp: Coding assistance for JupyterLab (code navigation + hover suggestions + linters + autocompletion + rename) using Language Server Protocol

The jupyter-lsp supports many more languages

Language servers — Language Server Protocol integration for Jupyter[Lab]

One of which is sql-language-server

GitHub - joe-re/sql-language-server: SQL Language Server

To install the sql-language-server all you have to do is npm install it. The jupyter-lsp will automatically discover it.

```
npm i -g sql-language-server
```

The sql-language-server supports syntax highlighting and code-completion.

The sql-language-server supports table/column completion when provided a connection string. We could potentially support Trino and Spark connections.

The jupyter-lsp is aware of the sql-language-server extension. The spec configuration file is here.
https://github.com/krassowski/jupyterlab-lsp/blob/master/python_packages/jupyter_lsp/jupyter_lsp/specs/sql_language_server.py

In this file you'll notice that it supports certain languages and mime types

```
languages = [
       "sql",
   ]
   args = ["up", "--method", "stdio"]
   spec = dict(
       display_name=key,
       mime_types=[
           "application/sql",
           "text/sql",
           "text/x-sql",
           "text/x-mysql",
           "text/x-mariadb",
           "text/x-pgsql",
       ],
```

However it does not support %%sparksql. It's possible to manually register LSP servers. I tried to do that for the %%sparksql magic.

I locally modified the spec file but still things were not working. I realized that there is another place where special handling for %%sql occurs and that is in the transclusions (on the client side, in the JavaScript executed by the browser).

https://github.com/krassowski/jupyterlab-lsp/blob/master/packages/jupyterlab-lsp/src/transclusions/ipython-sql/extractors.ts

```
new RegExpForeignCodeExtractor({
     language: 'sql',
     pattern: `^%%sql(?: (?:${SQL_URL_PATTERN}|${COMMAND_PATTERN}|(?:\\w+ << )|(?:\\w+@\\w+)))?\n?((?:.+\n)?(?:[
     foreign_capture_groups: [1],
     is_standalone: true,
     file_extension: 'sql'
   }),
```

I found where this files was locally,

/home/jovyan/.cache/yarn/v6/.tmp/da5f1d9e9c52cd5cd9cdb4bfd3a61f2f/src/transclusions/ipython-sql/extractors.ts
/home/jovyan/.cache/yarn/v6/npm-@krassowski-jupyterlab-lsp-2.1.2-6b0306d187f94f91b4f28c284e4aec6f900b2fea/node_modules/@krassowski/jupyterlab-lsp/src/transclusions/ipython-sql/extractors.ts

But still no luck. I then did a search in my browser's developer window and was able to locate the files. It's a compiled `.js` file. Which I then grep in my environment and replaced %%sql with %%sparksql.

```
$ find / -name "*da612.js"
/opt/conda/share/jupyter/labextensions/@krassowski/jupyterlab-lsp/static/364.2db93b0b5682d99da612.js
/opt/conda/lib/python3.8/site-packages/jupyterlab_lsp/labextensions/@krassowski/jupyterlab-lsp/static/364.2db93b0
```

This eventually worked but means changes to jupyter-lsp. I then realized that I could simply do IPython magic alias.

Built-in magic commands — IPython 8.14.0 documentation

```
%alias_magic --cell sql sparksql
```

So now cells with %%sql are executed by the sparksql-magic extension and thus executed by spark.

However code completion and syntax highlighting is handled by the sql-language-server LSP extension.

Custom JupyterLab SQL Formatter