Custom JupyterLab SQL Formatter

This page explains how to implement a custom formatter https://jupyterlab-code-formatter.readthedocs.io/en/latest/how-to-use.html#custom-formatter

Here's an example of a SQL formatter leveraging the sqlparse python package
sqlparse – Parse SQL statements — python-sqlparse 0.4.5.dev0 documentation

First install the package, make sure to install it as user

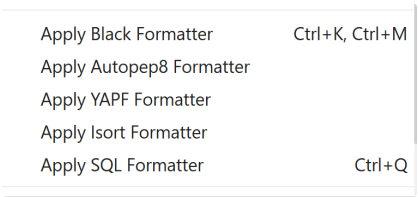
```
# Install the universal jupyterlab code formatter extension
pip install --user jupyterlab_code_formatter

# The documentation says you don't have to do this for JupyterLab > 3
# but I found I still needed to do this (JupyterHub issue?)
jupyter serverextension enable --py jupyterlab_code_formatter

# Our SQL formatter implementation leverages the sqlparse library so lets install it
pip install --user sqlparse
```

Here we implement a SqlFormatter class and register it with the universal code formatter extension. We do this by adding it directly into the configuration file `~/.jupyter/jupyter_notebook_config.py`

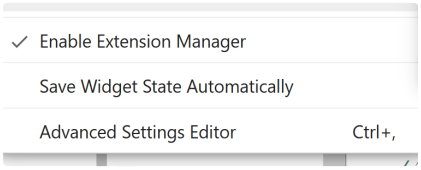
```
from jupyterlab_code_formatter.formatters import (
    BaseFormatter,
    SERVER_FORMATTERS,
)
import re

MAGIC_COMMAND_RE = re.compile(r"^%", flags=re.M)
COMMENTED_MAGIC_COMMAND_RE = re.compile(r"^-- #%#", flags=re.M)


class SqlFormatter(BaseFormatter):
    # Documentation of how to implement a custom jupyterlab formatter can be found
    # here https://jupyterlab-code-formatter.readthedocs.io/en/latest/how-to-use.html#custom-formatter
    # Files with the SERVER_FORMATTERS extensions in this case .sql will be formatted using this formatter
    # cells with the %%sql magic will also be formatted with this Class.
    #
    # Using the sqlparse python package to format sparksql magic cells
    # https://sqlparse.readthedocs.io/en/latest/api/#formatting
    # make sure to install it as user
    # pip install --user sqlparse

    label = "Apply SQL Formatter"

    @property
    def importable(self) -> bool:
```

```python
        try:
            import sqlparse
        except ImportError as error:
            # Output expected ImportErrors.
            print(error.__class__.__name__ + ": " + error.message)
            return False
        return True

    def format_code(self, code: str, notebook: bool, **options) -> str:
        import sqlparse

        code = re.sub(MAGIC_COMMAND_RE, "-- #%#", code)

        statements = sqlparse.split(code)
        formatted = []
        for s in statements:
            print(s)
            formatted.append(
                sqlparse.format(
                    s,
                    reindent_aligned=True,
                    keyword_case="upper",
                    use_space_around_operators=True,
                )
            )

        code =  "\n\n".join(formatted)
        code = re.sub(COMMENTED_MAGIC_COMMAND_RE, "%", code)
        return code

SERVER_FORMATTERS["sql"] = SqlFormatter()
```

Once this formatter is registered it will show up in the Edit menu of the JupyterLab.



To make it easier to invoke the formatting of SQL code you can register keyboard shortcuts in the advanced JupyterLab settings




```json
{
  "shortcuts": [
        {
            "command": "jupyterlab_code_formatter:black",
            "keys": [
                "Ctrl K",
```

```
                "Ctrl M"
            ],
            "selector": ".jp-Notebook.jp-mod-editMode"
        },
        {
            "command": "jupyterlab_code_formatter:sql",
            "keys": [
                "Ctrl Q"
            ],
            "selector": ".jp-Notebook.jp-mod-editMode"
        },
        {
            "command": "completer:invoke-notebook",
            "keys": [
                "Tab",
                "Ctrl G"
            ],
            "selector": ".jp-Notebook.jp-mod-editMode .jp-mod-completer-enabled"
        },
    ]}
```

Notice the `Ctrl Q` invokes the `sql` formatter which we registered in the SERVER_FORMATTERS.

We can invoke this formatter from the `.sql` file editor or from a JupyterLab code cell containing SQL for the `%%sparksql` or `%%sql` magic.