xeus-sqlite

Mariana Meireles

Mar 15, 2022

GETTING STARTED

`xeus-sqlite` is a Jupyter kernel for SQLite based on the native implementation of the Jupyter protocol xeus.

CHAPTER

ONE

LICENSING

We use a shared copyright model that enables all contributors to maintain the copyright on their contributions.

This software is licensed under the BSD-3-Clause license. See the LICENSE file for details.

1.1 Installation

1.1.1 With Conda or Mamba

xeus-sqlite has been packaged for the conda package manager.

To ensure that the installation works, it is preferable to install *xeus-sqlite* in a fresh conda/mamba environment. It is also needed to use a miniconda installation because with the full anaconda you may have a conflict with the *zeromq* library which is already installed in the anaconda distribution.

The safest usage is to create an environment named *xeus-sqlite* with your miniconda installation

```
conda create -n xeus-sqlite
conda activate xeus-sqlite # Or `source activate xeus-sqlite` for conda < 4.6
```

```
mamba create -n xeus-sqlite
mamba activate xeus-sqlite
```

Then you can install in this freshly created environment *xeus-sqlite* and its dependencies

```
conda install xeus-sqlite notebook -c conda-forge
```

```
mamba install xeus-sqlite notebook -c conda-forge
```

or, if you prefer to use JupyterLab

```
conda install xeus-sqlite jupyterlab -c conda-forge
```

```
mamba install xeus-sqlite jupyterlab -c conda-forge
```

1.1.2 From Source

You can install `xeus-sqlite` from source with cmake. This requires that you have all the dependencies installed in the same prefix.

```
conda install cmake nlohmann_json xtl cppzmq xeus sqlite sqlitecpp cpp-tabulate=1.3␣
↪xvega xproperty jupyterlab -c conda-forge
```

```
mamba install cmake nlohmann_json xtl cppzmq xeus sqlite sqlitecpp cpp-tabulate=1.3␣
↪xvega xproperty jupyterlab -c conda-forge
```

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
make install
```

On Windows platforms, from the source directory:

```
mkdir build
cd build
cmake -G "NMake Makefiles" -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
nmake
nmake install
```

1.2 Getting started

Launch Jupyter notebook with `jupyter notebook` or Jupyter lab with `jupyter lab` and create a new SQLite notebook by selecting the **xsqlite** kernel. Or launch Jupyter console with *jupyter console –kernel xsqlite*.

To perform operations on `xeus-sqlite` you must have a loaded database. You can do that using the `%LOAD` or `%CREATE` magics. Magics are commands that allow you to perform operations that are not necessarily SQLite code, we give a brief example on how to use the mentioned ones here, for more extensive explanation on magics access the main page.

1.2.1 Creating a new database

If you want to create a new database from scratch you can use:

```
%CREATE path-to-db/yourdatabase.db namedb
```

This will create a `yourdatabase.db` file in the specified location.

1.2.2 Loading a new database

If you already have a database you can load it with:

```
%LOAD path-to-db/yourdatabase.db
```

1.2.3 Usage

Now you can run normal SQLite code and the changes will take place on this database.

```
Warning: the changes made while in the jupyter interface are permanent to your database.
```

To change the database you're working with simply run the `%LOAD` or `%CREATE` magic with a new target.

1.2.4 Notes

It's recommended to use words from the syntax in upper case and arguments in lower case simply to improve readability. The input strings are sanitized and it doesn't matter if the input is written in upper or lower case, **with the exception** of names (like column names, tables names, etc) and the reading options to load a database that should receive the `[r w rw]` arguments in small letters.

1.3 SQLite magics

Magics that allow you to operate on the database.

1.3.1 LOAD

%LOAD <path-to-db/yourdatabase.db> [r | rw]
Loads a database.

Receives two arguments, the path to the database location as a string (it can be either the local or absolute path) and an option to open the database either as read and write "RW" or read only mode "R". If the optional argument is not set it will default to read and write mode.

1.3.2 CREATE

%CREATE <path-to-db/yourdatabase.db> name_of_databse
Creates a database in read and write mode.

Receives two arguments, a string that's the path to where it will create your database database and a string for the name of the database.

1.3.3 DELETE

%DELETE
Deletes the database that's currently loaded in the Jupyter application.

Warning: this will delete the file permanently.

1.3.4 TABLE_EXISTS

%TABLE_EXISTS table_name
Checks if a table exists.

1.3.5 GET_INFO

%GET_INFO
Get the following information about the loaded database: * Magic header string * Page size bytes * File format write version * File format read version * Reserved space bytes * Max embedded payload fraction * Min embedded payload fraction * Leaf payload fraction * File change counter * Database size pages * First freelist trunk page * Total freelist trunk pages * Schema cookie * Schema format number * Default page cache size bytes * Largest B tree page number * Database text encoding * User version * Incremental vaccum mode * Application ID * Version valid for * SQLite version

1.3.6 IS_UNENCRYPTED

%IS_UNENCRYPTED
Test if a file contains an unencrypted database.

1.3.7 LOAD_EXTENSION

%LOAD_EXTENSION <extension>
Load a module into the current sqlite database instance.

Receives the name of the shared library containing the extension.

1.3.8 REKEY

%REKEY <key>
Reset the key for the current sqlite database instance. This is the equivalent of the sqlite3_rekey call and should thus be called after the database has been opened with a valid key. To decrypt a database, call this method with an empty string.

Receives one argument which is the key you want to reset.

1.3.9 SET_KEY

%SET_KEY <key>
Set the key for the current sqlite database instance. This is the equivalent of the sqlite3_key call and should thus be called directly after opening the database.

Receives one argument which is the key you want to reset.

1.3.10 BACKUP

%BACKUP <0, 1>
Load the contents of a database file on disk into the "main" database of open database connection, or to save the current contents of the database into a database file on disk.

Receives one argument which is an int that can either be 0 for saving and 1 for loading.

1.4 XVega magics

Magics that allow you to create graph visualizations using XVega an implementation of vega-light to C++.

1.4.1 X_FIELD

%X_FIELD name_of_column
Represents the X axis of the graph. The name of the axis should be the same as the name of the SQLite column (or result of SQLite query).

1.4.1.1 TYPE

%TYPE type_of_field
Sub-attribute of **X_FIELD**.

Bellow there's list of the types supported by *xeus-sqlite*. If you want to learn more about types please refer to vega lite type official documentation.

- QUANTITATIVE
- NOMINAL
- ORDINAL
- TEMPORAL

1.4.1.2 BIN

%BIN type_of_field
Sub-attribute of **X_FIELD**.

Binning discretizes numeric values into a set of bins. If bin is true, default binning parameters are used.

To customize binning parameters, you can set bin to a bin definition object, which can have the following properties:

If you want to learn more about bin please refer to vega lite bin official documentation.

ANCHOR

%ANCHOR bin_position
Sub-attribute of **BIN**.

A value in the binned domain at which to anchor the bins, shifting the bin boundaries if necessary to ensure that a boundary aligns with the anchor value.

BASE

%BASE number
Sub-attribute of **BIN**.

The number base to use for automatic bin determination (default is base 10).

BINNED

%BASE boolean
Sub-attribute of **BIN**.

MAXBINS

%MAXBINS number
Sub-attribute of **BIN**.

Maximum number of bins.

MINSTEP

%MINSTEP number
Sub-attribute of **BIN**.

A minimum allowable step size (particularly useful for integer values).

NICE

%NICE bool
Sub-attribute of **BIN**.

If true, attempts to make the bin boundaries use human-friendly boundaries, such as multiples of ten.

STEP

%STEP number
Sub-attribute of **BIN**.

An exact step size to use between bins.

1.4.1.3 AGGREGATE

%AGGREGATE type_of_aggregation
Sub-attribute of **X_FIELD**.

The aggregate property of a field definition can be used to compute aggregate summary statistics (e.g., median, min, max) over groups of data.

Bellow there's list of the aggregations supported by *xeus-sqlite*. If you want to learn more about aggregations please refer to vega lite aggregate official documentation.

- COUNT
- VALID
- MISSING
- DISTINCT
- SUM
- PRODUCT
- MEAN
- AVERAGE
- VARIANCE
- VARIANCEP
- STDEV
- STEDEVP
- STEDERR
- MEDIAN
- Q1
- Q3
- CI0
- CI1
- MIN
- MAX
- ARGMIN
- ARGMAX

1.4.1.4 TIME_UNIT

%TIME_UNIT time
Sub-attribute of **X_FIELD**.

Time unit is used to discretize time.

Bellow there's list of the time units supported by *xeus-sqlite*. If you want to learn more about time units please refer to vega lite time unit official documentation.

- YEAR
- QUARTER
- MONTH
- DAY
- DATE
- HOURS
- MINUTES
- SECONDS
- MILISECONDS

1.4.2 Y_FIELD

%Y_FIELD name_of_column
Represents the Y axis of the graph. The name of the axis should be the same as the name of the SQLite column (or result of SQLite query).

1.4.2.1 TYPE

%TYPE type_of_field
Sub-attribute of **Y_FIELD**.

Bellow there's list of the types supported by *xeus-sqlite*. If you want to learn more about types please refer to vega lite type official documentation.

- QUANTITATIVE
- NOMINAL
- ORDINAL
- TEMPORAL

1.4.2.2 BIN

%BIN type_of_field
Sub-attribute of **Y_FIELD**.

Binning discretizes numeric values into a set of bins. If bin is true, default binning parameters are used.

To customize binning parameters, you can set bin to a bin definition object, which can have the following properties:

If you want to learn more about bin please refer to vega lite bin official documentation.

ANCHOR

%ANCHOR bin_position
Sub-attribute of **BIN**.

A value in the binned domain at which to anchor the bins, shifting the bin boundaries if necessary to ensure that a boundary aligns with the anchor value.

BASE

%BASE number
Sub-attribute of **BIN**.

The number base to use for automatic bin determination (default is base 10).

BINNED

%BASE boolean
Sub-attribute of **BIN**.

MAXBINS

%MAXBINS number
Sub-attribute of **BIN**.

Maximum number of bins.

MINSTEP

%MINSTEP number
Sub-attribute of **BIN**.

A minimum allowable step size (particularly useful for integer values).

NICE

%NICE bool
Sub-attribute of **BIN**.

If true, attempts to make the bin boundaries use human-friendly boundaries, such as multiples of ten.

STEP

%STEP number
Sub-attribute of **BIN**.

An exact step size to use between bins.

1.4.2.3 AGGREGATE

%AGGREGATE type_of_aggregation
Sub-attribute of **Y_FIELD**.

The aggregate property of a field definition can be used to compute aggregate summary statistics (e.g., median, min, max) over groups of data.

Bellow there's list of the aggregations supported by *xeus-sqlite*. If you want to learn more about aggregations please refer to vega lite aggregate official documentation.

- COUNT
- VALID
- MISSING
- DISTINCT
- SUM
- PRODUCT
- MEAN
- AVERAGE
- VARIANCE
- VARIANCEP
- STDEV
- STEDEVP
- STEDERR
- MEDIAN
- Q1
- Q3
- CI0
- CI1
- MIN
- MAX

- ARGMIN
- ARGMAX

1.4.2.4 TIME_UNIT

%TIME_UNIT time
Sub-attribute of **Y_FIELD**.

Time unit is used to discretize time.

Bellow there's list of the time units supported by *xeus-sqlite*. If you want to learn more about time units please refer to vega lite time unit official documentation.

- YEAR
- QUARTER
- MONTH
- DAY
- DATE
- HOURS
- MINUTES
- SECONDS
- MILISECONDS

1.4.3 WIDTH

%WIDTH number
Width of the graph in pixels.

1.4.4 HEIGHT

%HEIGHT number
Height of the graph in pixels.

1.4.5 MARK

%MARK mark
Marcs can be one of the following:

- ARC
- AREA
- BAR
- CIRCLE
- LINE
- POINT

- RECT
- RULE
- SQUARE
- TICK
- TRAIL

1.4.5.1 COLOR

%COLOR color
Sub-attribute of **MARK**.

Sets the color of a mark. The color can be one of the valid CSS color string.

1.4.6 GRID

%HEIGHT boolean
Enable or disable grid view on graph.