Documentation

search

Search

- *rocket_launch*

  Get started

  - Installation
    add
  - Fundamentals
    add
  - First steps
    add
- *code*

  Develop

  - Concepts
    add
  - API reference
    remove
    - PAGE ELEMENTS

    - Write and magic
      add
    - Text elements
      add
    - Data elements
      add
    - Chart elements
      add
    - Input widgets
      add
    - Media elements
      add
    - Layouts and containers
      add
    - Chat elements
      add
    - Status elements
      add
    - Third-party components*open_in_new*
    - APPLICATION LOGIC

    - Navigation and pages
      add
    - Execution flow
      add
    - Caching and state
      add
    - Connections and secrets
      remove
      - SECRETS

star

Tip

This page only contains the `st.connections.SQLConnection` class. For a deeper dive into creating and managing data connections within Streamlit apps, read Connecting to data.

st.connections.SQLConnection

Streamlit Version Version 1.41.0

A connection to a SQL database using a SQLAlchemy Engine.

Initialize this connection object using `st.connection("sql")` or `st.connection("<name>", type="sql")`. Connection parameters for a SQLConnection can be specified using `secrets.toml` and/or `**kwargs`. Possible connection parameters include:

- `url` or keyword arguments for `sqlalchemy.engine.URL.create()`, except `drivername`. Use `dialect` and `driver` instead of `drivername`.
- Keyword arguments for `sqlalchemy.create_engine()`, including custom `connect()` arguments used by your specific `dialect` or `driver`.
- `autocommit`. If this is `False` (default), the connection operates in manual commit (transactional) mode. If this is `True`, the connection operates in autocommit (non-transactional) mode.

If `url` exists as a connection parameter, Streamlit will pass it to `sqlalchemy.engine.make_url()`. Otherwise, Streamlit requires (at a minimum) `dialect`, `username`, and `host`. Streamlit will use `dialect` and `driver` (if defined) to derive `drivername`, then pass the relevant connection parameters to `sqlalchemy.engine.URL.create()`.

In addition to the default keyword arguments for `sqlalchemy.create_engine()`, your dialect may accept additional keyword arguments. For example, if you use `dialect="snowflake"` with Snowflake SQLAlchemy, you can pass a value for `private_key` to use key-pair authentication. If you use `dialect="bigquery"` with Google BigQuery, you can pass a value for `location`.

SQLConnection provides the `.query()` convenience method, which can be used to run simple, read-only queries with both caching and simple error handling/retries. More complex database interactions can be performed by using the `.session` property to receive a regular SQLAlchemy Session.

Important

SQLAlchemy must be installed in your environment to use this connection. You must also install your driver, such as `pyodbc` or `psycopg2`.

Class description[source]

st.connections.SQLConnection(connection_name, **kwargs)

Class description[source]

st.connections.SQLConnection(connection_name, **kwargs)

Methods

connect()	Call `.connect()` on the underlying SQLAlchemy Engine, returning a new connection object.
query(sql, *, show_spinner="Running `sql.query(...)`.", ttl=None, index_col=None, chunksize=None, params=None, **kwargs)	Run a read-only query.
reset()	Reset this connection so that it gets reinitialized the next time it's used.

Attributes

driver	The name of the driver used by the underlying SQLAlchemy Engine.
engine	The underlying SQLAlchemy Engine.
session	Return a SQLAlchemy Session.

Examples

Example 1: Configuration with URL

You can configure your SQL connection using Streamlit's Secrets management. The following example specifies a SQL connection URL.

`.streamlit/secrets.toml`:

```
[connections.sql]
url = "xxx+xxx://xxx:xxx@xxx:xxx/xxx"
```

Your app code:

```
import streamlit as st

conn = st.connection("sql")
df = conn.query("SELECT * FROM pet_owners")
st.dataframe(df)
```

Example 2: Configuration with dialect, host, and username

If you do not specify `url`, you must at least specify `dialect`, `host`, and `username` instead. The following example also includes `password`.

`.streamlit/secrets.toml`:

```
[connections.sql]
dialect = "xxx"
host = "xxx"
```

```
username = "xxx"
password = "xxx"
```

Your app code:

```
import streamlit as st

conn = st.connection("sql")
df = conn.query("SELECT * FROM pet_owners")
st.dataframe(df)
```

Example 3: Configuration with keyword arguments

You can configure your SQL connection with keyword arguments (with or without `secrets.toml`). For example, if you use Microsoft Entra ID with a Microsoft Azure SQL server, you can quickly set up a local connection for development using interactive authentication.

This example requires the Microsoft ODBC Driver for SQL Server for *Windows* in addition to the `sqlalchemy` and `pyodbc` packages for Python.

```
import streamlit as st

conn = st.connection(
    "sql",
    dialect="mssql",
    driver="pyodbc",
    host="xxx.database.windows.net",
    database="xxx",
    username="xxx",
    query={
        "driver": "ODBC Driver 18 for SQL Server",
        "authentication": "ActiveDirectoryInteractive",
        "encrypt": "yes",
    },
)

df = conn.query("SELECT * FROM pet_owners")
st.dataframe(df)
```

SQLConnection.connect

Streamlit Version Version 1.41.0

Call `.connect()` on the underlying SQLAlchemy Engine, returning a new connection object.

Calling this method is equivalent to calling `self._instance.connect()`.

NOTE: This method should not be confused with the internal `_connect` method used to implement a Streamlit Connection.

Function signature[source]

SQLConnection.connect()

Returns

(sqlalchemy.engine.Connection) A new SQLAlchemy connection object.

SQLConnection.query

Streamlit Version Version 1.41.0

Run a read-only query.

This method implements query result caching and simple error handling/retries. The caching behavior is identical to that of using `@st.cache_data`.

Note

Queries that are run without a specified ttl are cached indefinitely.

All keyword arguments passed to this function are passed down to `pandas.read_sql`, except `ttl`.

Function signature[source]

SQLConnection.query(sql, *, show_spinner="Running `sql.query(...)`.", ttl=None, index_col=None, chunksize=None, params=None, **kwargs)

Parameters	
sql (str)	The read-only SQL query to execute.
show_spinner (boolean or string)	Enable the spinner. The default is to show a spinner when there is a "cache miss" and the cached resource is being created. If a string, the value of the show_spinner param will be used for the spinner text.
ttl (float, int, timedelta or None)	The maximum number of seconds to keep results in the cache, or None if cached results should not expire. The default is None.
index_col (str, list of str, or None)	Column(s) to set as index(MultiIndex). Default is None.
chunksize (int or None)	If specified, return an iterator where chunksize is the number of rows to include in each chunk. Default is None.
params (list, tuple, dict or None)	List of parameters to pass to the execute method. The syntax used to pass parameters is database driver dependent. Check your database driver documentation for which of the five syntax styles, described in PEP 249 paramstyle, is supported. Default is None.
**kwargs (dict)	Additional keyword arguments are passed to `pandas.read_sql`.
Returns	
(pandas.DataFrame)	The result of running the query, formatted as a pandas DataFrame.

Example

```
import streamlit as st
```

```
conn = st.connection("sql")
df = conn.query(
    "SELECT * FROM pet_owners WHERE owner = :owner",
    ttl=3600,
    params={"owner": "barbara"},
)
st.dataframe(df)
```

SQLConnection.reset

Streamlit Version Version 1.41.0

Reset this connection so that it gets reinitialized the next time it's used.

This method can be useful when a connection has become stale, an auth token has expired, or in similar scenarios where a broken connection might be fixed by reinitializing it. Note that some connection methods may already use `reset()` in their error handling code.

Function signature[source]

SQLConnection.reset()

Returns

(None) No description

Example

```
import streamlit as st

conn = st.connection("my_conn")

# Reset the connection before using it if it isn't healthy
# Note: is_healthy() isn't a real method and is just shown for example here.
if not conn.is_healthy():
    conn.reset()

# Do stuff with conn...
```

SQLConnection.driver

Streamlit Version Version 1.41.0

The name of the driver used by the underlying SQLAlchemy Engine.

This is equivalent to accessing `self._instance.driver`.

Function signature[source]

SQLConnection.driver

Returns

(str) The name of the driver. For example, `"pyodbc"` or `"psycopg2"`.

SQLConnection.engine

Streamlit Version Version 1.41.0

The underlying SQLAlchemy Engine.

This is equivalent to accessing `self._instance`.

Function signature[source]

SQLConnection.engine

Returns

(sqlalchemy.engine.base.Engine) The underlying SQLAlchemy Engine.

SQLConnection.session

Streamlit Version Version 1.41.0

Return a SQLAlchemy Session.

Users of this connection should use the contextmanager pattern for writes, transactions, and anything more complex than simple read queries.

See the usage example below, which assumes we have a table `numbers` with a single integer column `val`. The SQLAlchemy docs also contain much more information on the usage of sessions.

Function signature[source]

SQLConnection.session

Returns

(sqlalchemy.orm.Session) A SQLAlchemy Session.

Example

```
import streamlit as st
conn = st.connection("sql")
n = st.slider("Pick a number")
if st.button("Add the number!"):
    with conn.session as session:
        session.execute("INSERT INTO numbers (val) VALUES (:n);", {"n": n})
        session.commit()
```

←Previous: SnowflakeConnectionNext: BaseConnection→
forum

Still have questions?

Our forums are full of helpful information and Streamlit experts.

forum Ask AI