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

Tip

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

st.connections.SnowflakeConnection

Streamlit Version Version 1.41.0

A connection to Snowflake using the Snowflake Connector for Python.

Initialize this connection object using `st.connection("snowflake")` or `st.connection("<name>", type="snowflake")`. Connection parameters for a SnowflakeConnection can be specified using `secrets.toml` and/or `**kwargs`. Connection parameters are passed to `snowflake.connector.connect()`.

When an app is running in Streamlit in Snowflake, `st.connection("snowflake")` connects automatically using the app owner's role without further configuration. `**kwargs` will be ignored in this case. Use `secrets.toml` and `**kwargs` to configure your connection for local development.

SnowflakeConnection includes several convenience methods. For example, you can directly execute a SQL query with `.query()` or access the underlying Snowflake Connector object with `.raw_connection`.

Tip

snowflake-snowpark-python must be installed in your environment to use this connection. You can install Snowflake extras along with Streamlit:

```
>>> pip install streamlit[snowflake]
```

Important

Account identifiers must be of the form `<orgname>-<account_name>` where `<orgname>` is the name of your Snowflake organization and `<account_name>` is the unique name of your account within your organization. This is dash-separated, not dot-separated like when used in SQL queries. For more information, see Account identifiers.

Class description[source]

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

Methods	
cursor()	Create a new cursor object from this connection.
query(sql, *, ttl=None, show_spinner="Running `snowflake.query(...)`.", params=None, **kwargs)	Run a read-only SQL query.
reset()	Reset this connection so that it gets reinitialized the next time it's used.
session()	Create a new Snowpark session from this connection.
write_pandas(df, table_name, database=None, schema=None, chunk_size=None, **kwargs)	Write a `pandas.DataFrame` to a table in a Snowflake database.
Attributes	
raw_connection	Access the underlying connection object from the Snowflake Connector for Python.

Examples

Example 1: Configuration with Streamlit secrets

You can configure your Snowflake connection using Streamlit's Secrets management. For example, if you have MFA enabled on your account, you can connect using key-pair authentication.

`.streamlit/secrets.toml`:

```
[connections.snowflake]
account = "xxx-xxx"
user = "xxx"
private_key_file = "/xxx/xxx/xxx.p8"
role = "xxx"
warehouse = "xxx"
database = "xxx"
schema = "xxx"
```

Your app code:

```
import streamlit as st
conn = st.connection("snowflake")
df = conn.query("SELECT * FROM my_table")
```

Example 2: Configuration with keyword arguments and external authentication

You can configure your Snowflake connection with keyword arguments (with or without `secrets.toml`). For example, if your Snowflake account supports SSO, you can set up a quick local connection for development using browser-based SSO.

```
import streamlit as st
conn = st.connection(
    "snowflake", account="xxx-xxx", user="xxx", authenticator="externalbrowser"
)
df = conn.query("SELECT * FROM my_table")
```

Example 3: Named connection with Snowflake's connection configuration file

Snowflake's Python Connector supports a connection configuration file, which is well integrated with Streamlit's `SnowflakeConnection`. If you already have one or more connections configured, all you need to do is pass the name of the connection to use.

`~/.snowflake/connections.toml`:

```
[my_connection]
account = "xxx-xxx"
user = "xxx"
password = "xxx"
warehouse = "xxx"
database = "xxx"
schema = "xxx"
```

Your app code:

```
import streamlit as st
conn = st.connection("my_connection", type="snowflake")
df = conn.query("SELECT * FROM my_table")
```

Example 4: Named connection with Streamlit secrets and Snowflake's connection configuration file

If you have a Snowflake configuration file with a connection named `my_connection` as in Example 3, you can pass the connection name through `secrets.toml`.

`.streamlit/secrets.toml`:

```
[connections.snowflake]
connection_name = "my_connection"
```

Your app code:

```
import streamlit as st
conn = st.connection("snowflake")
df = conn.query("SELECT * FROM my_table")
```

Example 5: Default connection with an environment variable

If you have a Snowflake configuration file with a connection named `my_connection` as in Example 3, you can set an environment variable to declare it as the default Snowflake connection.

```
SNOWFLAKE_DEFAULT_CONNECTION_NAME = "my_connection"
```

Your app code:

```
import streamlit as st
conn = st.connection("snowflake")
df = conn.query("SELECT * FROM my_table")
```

Example 6: Default connection in Snowflake's connection configuration file

If you have a Snowflake configuration file that defines your `default` connection, Streamlit will automatically use it if no other connection is declared.

`~/.snowflake/connections.toml`:

```
[default]
account = "xxx-xxx"
user = "xxx"
password = "xxx"
warehouse = "xxx"
database = "xxx"
schema = "xxx"
```

Your app code:

```
import streamlit as st
conn = st.connection("snowflake")
df = conn.query("SELECT * FROM my_table")
```

SnowflakeConnection.cursor

Streamlit Version Version 1.41.0

Create a new cursor object from this connection.

Snowflake Connector cursors implement the Python Database API v2.0 specification (PEP-249). For more information, see the Snowflake Connector for Python documentation.

Function signature[source]

SnowflakeConnection.cursor()

Returns

(snowflake.connector.cursor.SnowflakeCursor) A cursor object for the connection.

Example

The following example uses a cursor to insert multiple rows into a table. The `qmark` parameter style is specified as an optional keyword argument. Alternatively, the parameter style can be declared in your connection configuration file. For more information, see the Snowflake Connector for Python documentation.

```
import streamlit as st

conn = st.connection("snowflake", "paramstyle"="qmark")
rows_to_insert = [("Mary", "dog"), ("John", "cat"), ("Robert", "bird")]
conn.cursor().executemany(
    "INSERT INTO mytable (name, pet) VALUES (?, ?)", rows_to_insert
)
```

SnowflakeConnection.query

Streamlit Version Version 1.41.0

Run a read-only SQL 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.

Function signature[source]

SnowflakeConnection.query(sql, *, ttl=None, show_spinner="Running `snowflake.query(...)`.", params=None, **kwargs)

Parameters	
sql (str)	The read-only SQL query to execute.
ttl (float, int, timedelta or None)	The maximum number of seconds to keep results in the cache. If this is `None` (default), cached results do not expire with time.
show_spinner (boolean or string)	Whether to enable the spinner. When a cached query is executed, no spinner is displayed because the result is immediately available. When a new query is executed, the default is to show a spinner with the message "Running `snowflake.query(...)`." If this is `False`, no spinner displays while executing the query. If this is a string, the string will be used as the message for the spinner.
params (list, tuple, dict or None)	List of parameters to pass to the Snowflake Connector for Python `Cursor.execute()` method. This connector supports binding data to a SQL statement using qmark bindings. For more information and examples, see the Snowflake Connector for Python documentation. This defaults to `None`.
Returns	
(pandas.DataFrame)	The result of running the query, formatted as a pandas DataFrame.

Example

```
import streamlit as st

conn = st.connection("snowflake")
df = conn.query("SELECT * FROM my_table")
st.dataframe(df)
```

SnowflakeConnection.raw_connection

Streamlit Version Version 1.41.0

Access the underlying connection object from the Snowflake Connector for Python.

For information on how to use the Snowflake Connector for Python, see the Snowflake Connector for Python documentation.

Function signature[source]

SnowflakeConnection.raw_connection

Returns

(snowflake.connector.connection.SnowflakeConnection) The connection object.

Example

The following example uses a cursor to submit an asynchronous query, saves the query ID, then periodically checks the query status through the connection before retrieving the results.

```
import streamlit as st
import time

conn = st.connection("snowflake")
cur = conn.cursor()
cur.execute_async("SELECT * FROM my_table")
query_id = cur.sfqid
while True:
    status = conn.raw_connection.get_query_status(query_id)
    if conn.raw_connection.is_still_running(status):
        time.sleep(1)
    else:
        break
cur.get_results_from_sfqid(query_id)
df = cur.fetchall()
```

SnowflakeConnection.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]

SnowflakeConnection.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...
```

SnowflakeConnection.session

Streamlit Version Version 1.41.0

Create a new Snowpark session from this connection.

For information on how to use Snowpark sessions, see the Snowpark developer guide and Snowpark API Reference.

Function signature[source]

SnowflakeConnection.session()

Returns

(snowflake.snowpark.Session) A new Snowpark session for this connection.

Example

The following example creates a new Snowpark session and uses it to run a query.

```
import streamlit as st

conn = st.connection("snowflake")
session = conn.session()
df = session.sql("SELECT * FROM my_table").collect()
```

SnowflakeConnection.write_pandas

Streamlit Version Version 1.41.0

Write a `pandas.DataFrame` to a table in a Snowflake database.

This convenience method is a thin wrapper around `snowflake.connector.pandas_tools.write_pandas()` using the underlying connection. The `conn` parameter is passed automatically. For more information and additional keyword arguments, see the Snowflake Connector for Python documentation.

Function signature[source]

SnowflakeConnection.write_pandas(df, table_name, database=None, schema=None, chunk_size=None, **kwargs)

Parameters

df (pandas.DataFrame)	The `pandas.DataFrame` object containing the data to be copied into the table.
table_name (str)	Name of the table where the data should be copied to.
database (str)	Name of the database containing the table. By default, the function writes to the database that is currently in use in the session. Note If you specify this parameter, you must also specify the schema parameter.
schema (str)	Name of the schema containing the table. By default, the function writes to the table in the schema that is currently in use in the session.
chunk_size (int)	Number of elements to insert at a time. By default, the function inserts all elements in one chunk.
**kwargs (Any)	Additional keyword arguments for `snowflake.connector.pandas_tools.write_pandas()`.

Returns

(tuple[bool, int, int])	A tuple containing three values: 1. A boolean value that is `True` if the write was successful. 2. An integer giving the number of chunks of data that were copied. 3. An integer giving the number of rows that were inserted.

Example

The following example uses the database and schema currently in use in the session and copies the data into a table named "my_table."

```
import streamlit as st
import pandas as pd

df = pd.DataFrame(
    {"Name": ["Mary", "John", "Robert"], "Pet": ["dog", "cat", "bird"]}
)
conn = st.connection("snowflake")
conn.write_pandas(df, "my_table")
```

←Previous: st.connectionNext: SQLConnection→

forum

Still have questions?

Our forums are full of helpful information and Streamlit experts.

© 2025 Snowflake Inc. Cookie policy

forum Ask AI