In [2]:
```
!pip install ipython-sql
```

```
Requirement already satisfied: ipython-sql in c:\users\lenovo\anaconda3\lib\site-packages (0.5.0)
Requirement already satisfied: prettytable in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql) (3.
16.0)
Requirement already satisfied: ipython in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql) (8.27.0
)
Requirement already satisfied: sqlalchemy>=2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql)
(2.0.34)
Requirement already satisfied: sqlparse in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql) (0.5.3
)
Requirement already satisfied: six in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql) (1.16.0)
Requirement already satisfied: ipython-genutils in c:\users\lenovo\anaconda3\lib\site-packages (from ipython-sql
) (0.2.0)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\lenovo\anaconda3\lib\site-packages (from sql
alchemy>=2.0->ipython-sql) (4.11.0)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\lenovo\anaconda3\lib\site-packages (from sqlalchemy>
=2.0->ipython-sql) (3.0.1)
Requirement already satisfied: decorator in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->ipython-s
ql) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->ipython-
sql) (0.19.1)
Requirement already satisfied: matplotlib-inline in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->i
python-sql) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\lenovo\anaconda3\lib\site-packages (fro
m ipython->ipython-sql) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->ipy
thon-sql) (2.15.1)
Requirement already satisfied: stack-data in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->ipython-
sql) (0.2.0)
Requirement already satisfied: traitlets>=5.13.0 in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->i
python-sql) (5.14.3)
Requirement already satisfied: colorama in c:\users\lenovo\anaconda3\lib\site-packages (from ipython->ipython-sq
l) (0.4.6)
Requirement already satisfied: wcwidth in c:\users\lenovo\anaconda3\lib\site-packages (from prettytable->ipython
-sql) (0.2.5)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\lenovo\anaconda3\lib\site-packages (from jedi>=0.
16->ipython->ipython-sql) (0.8.3)
Requirement already satisfied: executing in c:\users\lenovo\anaconda3\lib\site-packages (from stack-data->ipytho
n->ipython-sql) (0.8.3)
Requirement already satisfied: asttokens in c:\users\lenovo\anaconda3\lib\site-packages (from stack-data->ipytho
n->ipython-sql) (2.0.5)
Requirement already satisfied: pure-eval in c:\users\lenovo\anaconda3\lib\site-packages (from stack-data->ipytho
n->ipython-sql) (0.2.2)
```

In [10]:
```
# %load
!pip install ipython-sql
!pip install ipython-sql

import json
import getpass
import hashlib

def import_pandas_safely():
    try:
        return __import__('pandas')
    except ImportError:
        return False


__pandas = import_pandas_safely()


def is_data_frame(v: str):
    obj = eval(v)
    if  isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):
        return True


def dataframe_columns(var):
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return [[df.name, str(df.dtype)]]
    return list(map(lambda col: [col, str(df[col].dtype)], df.columns))


def dtypes_str(frame):
    return str(eval(frame).dtypes)

def dataframe_hash(var):
    # Return a hash including the column names and number of rows
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode('utf-8')).hexdigest()
    return hashlib.sha256(f"{var}-{','.join(df.columns)},{len(df)}".encode('utf-8')).hexdigest()

def get_dataframes():
    if __pandas is None:
        return []
    user = getpass.getuser()
    values = %who_ls
    dataframes = [
        {
            "name": var,
            "type": type(eval(var)).__name__,
            "hash": dataframe_hash(var),
            "cols": dataframe_columns(var),
            "dtypesStr": dtypes_str(var),
        }
        for var in values if is_data_frame(var)
    ]
    result = {"dataframes": dataframes, "user": user}
    return json.dumps(result, ensure_ascii=False)


get_dataframes()
%sql sqlite:///economic_

import json
import getpass
import hashlib

def import_pandas_safely():
    try:
        return __import__('pandas')
    except ImportError:
        return False


__pandas = import_pandas_safely()


def is_data_frame(v: str):
    obj = eval(v)
    if  isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):
        return True


def dataframe_columns(var):
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return [[df.name, str(df.dtype)]]
    return list(map(lambda col: [col, str(df[col].dtype)], df.columns))


def dtypes_str(frame):
    return str(eval(frame).dtypes)

def dataframe_hash(var):
    # Return a hash including the column names and number of rows
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode('utf-8')).hexdigest()
    return hashlib.sha256(f"{var}-{','.join(df.columns)},{len(df)}".encode('utf-8')).hexdigest()

def get_dataframes():
    if __pandas is None:
        return []
    user = getpass.getuser()
    values = %who_ls
    dataframes = [
        {
            "name": var,
            "type": type(eval(var)).__name__,
            "hash": dataframe_hash(var),
            "cols": dataframe_columns(var),
            "dtypesStr": dtypes_str(var),
        }
        for var in values if is_data_frame(var)
    ]
    result = {"dataframes": dataframes, "user": user}
    return json.dumps(result, ensure_ascii=False)


get_dataframes()
```

```
%sql sqlite:///economic_data.db

import json
import getpass
import hashlib

def import_pandas_safely():
    try:
        return __import__('pandas')
    except ImportError:
        return False


__pandas = import_pandas_safely()


def is_data_frame(v: str):
    obj = eval(v)
    if  isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):
        return True


def dataframe_columns(var):
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return [[df.name, str(df.dtype)]]
    return list(map(lambda col: [col, str(df[col].dtype)], df.columns))


def dtypes_str(frame):
    return str(eval(frame).dtypes)

def dataframe_hash(var):
    # Return a hash including the column names and number of rows
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode('utf-8')).hexdigest()
    return hashlib.sha256(f"{var}-{','.join(df.columns)},{len(df)}".encode('utf-8')).hexdigest()

def get_dataframes():
    if __pandas is None:
        return []
    user = getpass.getuser()
    values = %who_ls
    dataframes = [
        {
            "name": var,
            "type": type(eval(var)).__name__,
            "hash": dataframe_hash(var),
            "cols": dataframe_columns(var),
            "dtypesStr": dtypes_str(var),
        }
        for var in values if is_data_frame(var)
    ]
    result = {"dataframes": dataframes, "user": user}
    return json.dumps(result, ensure_ascii=False)


get_dataframes()
sql sqlite:///economic_data.db

import json
import getpass
import hashlib

def import_pandas_safely():
    try:
        return __import__('pandas')
    except ImportError:
        return False


__pandas = import_pandas_safely()


def is_data_frame(v: str):
    obj = eval(v)
    if  isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):
        return True


def dataframe_columns(var):
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return [[df.name, str(df.dtype)]]
    return list(map(lambda col: [col, str(df[col].dtype)], df.columns))


def dtypes_str(frame):
    return str(eval(frame).dtypes)

def dataframe_hash(var):
    # Return a hash including the column names and number of rows
    df = eval(var)
    if isinstance(df, __pandas.core.series.Series):
        return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode('utf-8')).hexdigest()
    return hashlib.sha256(f"{var}-{','.join(df.columns)},{len(df)}".encode('utf-8')).hexdigest()

def get_dataframes():
    if __pandas is None:
        return []
    user = getpass.getuser()
    values = %who_ls
    dataframes = [
        {
            "name": var,
            "type": type(eval(var)).__name__,
            "hash": dataframe_hash(var),
            "cols": dataframe_columns(var),
            "dtypesStr": dtypes_str(var),
        }
        for var in values if is_data_frame(var)
    ]
    result = {"dataframes": dataframes, "user": user}
    return json.dumps(result, ensure_ascii=False)


get_dataframes()
```

In [1]:
```
import pandas as pd
import sqlite3
```

In [5]:
```
conn=sqlite3.connect('jupyter_sql-tutorial.db')
```

In [7]:
```
df.to_sql('people',conn)
```

Out[7]:
```
3
```

In [9]:
```
load_ext sql
```

In [11]:
```
%sql sqlite:///jupyter_sql_tutorial.db
```

In [15]:
```
%sql sqlite:///economic_data.db
```

In [66]:
```
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()
cursor.execute('''
    CREATE TABLE economic (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        country TEXT NOT NULL,
        year INTEGER NOT NULL,
        gdp REAL,
        inflation REAL,
        unemployment_rate REAL
    );
''')
conn.commit()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
print(cursor.fetchall())
```

```
[('economic',), ('sqlite_sequence',)]
```

In [74]:
```
cursor.executemany('''
    INSERT INTO economic (country, year, gdp, inflation, unemployment_rate) VALUES (?, ?, ?, ?, ?)
''', [
    ('USA', 2020, 21.43, 1.2, 6.7),
    ('USA', 2021, 23.0, 4.7, 5.4),
    ('Germany', 2020, 3.8, 0.5, 4.2),
    ('Germany', 2021, 4.2, 3.1, 3.6),
    ('India', 2020, 2.9, 6.6, 7.1),
    ('India', 2021, 3.1, 5.1, 6.3)
])
conn.commit()
```

```
In [80]: query1 = '''
         SELECT country, gdp, inflation
         FROM economic
         WHERE year = 2021
         ORDER BY gdp DESC;
         '''
         cursor.execute(query1)
         print(cursor.fetchall())
```

```
[('USA', 23.0, 4.7), ('USA', 23.0, 4.7), ('USA', 23.0, 4.7), ('Germany', 4.2, 3.1), ('Germany', 4.2, 3.1), ('Ger
many', 4.2, 3.1), ('India', 3.1, 5.1), ('India', 3.1, 5.1), ('India', 3.1, 5.1)]
```

```
In [78]: query2 = '''
         SELECT country, AVG(unemployment_rate) as avg_unemployment
         FROM economic
         GROUP BY country;
         '''
         cursor.execute(query2)
         print(cursor.fetchall())
```

```
[('Germany', 3.9000000000000004), ('India', 6.699999999999999), ('USA', 6.050000000000001)]
```

```
In [88]: # INNER JOIN: economic + regions
         query3 = '''
         SELECT e.country, e.year, e.gdp, r.region
         FROM economic e
         INNER JOIN regions r ON e.country = r.country;
         '''
         cursor.execute(query3)
         print(cursor.fetchall())
```

```
[('USA', 2020, 21.43, 'North America'), ('USA', 2021, 23.0, 'North America'), ('Germany', 2020, 3.8, 'Europe'),
('Germany', 2021, 4.2, 'Europe'), ('India', 2020, 2.9, 'Asia'), ('India', 2021, 3.1, 'Asia'), ('USA', 2020, 21.4
3, 'North America'), ('USA', 2021, 23.0, 'North America'), ('Germany', 2020, 3.8, 'Europe'), ('Germany', 2021, 4
.2, 'Europe'), ('India', 2020, 2.9, 'Asia'), ('India', 2021, 3.1, 'Asia'), ('USA', 2020, 21.43, 'North America')
, ('USA', 2021, 23.0, 'North America'), ('Germany', 2020, 3.8, 'Europe'), ('Germany', 2021, 4.2, 'Europe'), ('In
dia', 2020, 2.9, 'Asia'), ('India', 2021, 3.1, 'Asia')]
```

```
In [84]: # LEFT JOIN
         query4 = '''
         SELECT e.country, r.region
         FROM economic e
         LEFT JOIN regions r ON e.country = r.country;
         '''
         cursor.execute(query4)
         print(cursor.fetchall())
```

```
[('USA', 'North America'), ('USA', 'North America'), ('Germany', 'Europe'), ('Germany', 'Europe'), ('India', 'As
ia'), ('India', 'Asia'), ('USA', 'North America'), ('USA', 'North America'), ('Germany', 'Europe'), ('Germany',
'Europe'), ('India', 'Asia'), ('India', 'Asia'), ('USA', 'North America'), ('USA', 'North America'), ('Germany',
'Europe'), ('Germany', 'Europe'), ('India', 'Asia'), ('India', 'Asia')]
```

```
In [86]: # Simulate RIGHT JOIN (SQLite doesn't support RIGHT JOIN)
         # Swap LEFT JOIN table order
         query5 = '''
         SELECT r.country, r.region, e.gdp
         FROM regions r
         LEFT JOIN economic e ON r.country = e.country;
         '''
         cursor.execute(query5)
         print(cursor.fetchall())
```

```
[('USA', 'North America', 21.43), ('USA', 'North America', 21.43), ('USA', 'North America', 21.43), ('USA', 'Nor
th America', 23.0), ('USA', 'North America', 23.0), ('USA', 'North America', 23.0), ('Germany', 'Europe', 3.8),
('Germany', 'Europe', 3.8), ('Germany', 'Europe', 3.8), ('Germany', 'Europe', 4.2), ('Germany', 'Europe', 4.2),
('Germany', 'Europe', 4.2), ('India', 'Asia', 2.9), ('India', 'Asia', 2.9), ('India', 'Asia', 2.9), ('India', 'A
sia', 3.1), ('India', 'Asia', 3.1), ('India', 'Asia', 3.1)]
```

```
In [92]: # Countries with GDP above average
         query6 = '''
         SELECT country, gdp
         FROM economic
         WHERE gdp > (SELECT AVG(gdp) FROM economic WHERE year = 2021)
         AND year = 2021;
         '''
         cursor.execute(query6)
         print(cursor.fetchall())
```

```
[('USA', 23.0), ('USA', 23.0), ('USA', 23.0)]
```

```
In [94]: # Total GDP per year
         query7 = '''
         SELECT year, SUM(gdp) as total_gdp
         FROM economic
         GROUP BY year;
         '''
```

```
cursor.execute(query7)
print(cursor.fetchall())
```

```
[(2020, 84.39), (2021, 90.9)]
```

In [96]:
```
# View: average inflation by region
cursor.execute('''
CREATE VIEW IF NOT EXISTS regional_inflation AS
SELECT r.region, AVG(e.inflation) as avg_inflation
FROM economic e
JOIN regions r ON e.country = r.country
GROUP BY r.region;
''')

# Query the view
cursor.execute('SELECT * FROM regional_inflation;')
print(cursor.fetchall())
```

```
[('Asia', 5.849999999999999), ('Europe', 1.8), ('North America', 2.9499999999999997)]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js