Connecting to PostgreSQL databases with Python

Objectives

- Extracting database tables and loading them into a `pandas` dataframe.
- Exporting `pandas` dataframes into an PostgreSQL database.

PostgreSQL is an Object Relational Database Management System (ORDMS), and in order to connect to it with Python we will be using the library `psycopg2` .

Imports

`pip3 install psycopg2` : C compiler required, use for commercial prefered.
`pip3 install psycopg2-binary` : pre-compiled binary version of the module.
`pip3 install sqlachemy` : Used to pass a pandas dataframe into PostgreSQL.

In [ ]:
```
import psycopg2
import pandas as pd
from sqlalchemy import create_engine
```

Connecting to the PostgreSQL server

In [ ]:
```
# Establishing connection to PostgreSQL
conn = psycopg2.connect(
    host= 'localhost',
    port= 5432,
    dbname= 'analysis',
    user= 'postgres',
    password= 'admin'
)

# Initializing the cursor
cur = conn.cursor()
```

In [ ]:
```
# Creating a table in the database
cur.execute("""
CREATE TABLE IF NOT EXISTS persons (
id INT PRIMARY KEY,
name VARCHAR(255),
age INT,
gender CHAR
);
""")
```

In [ ]:
```
# Placing data into the database
cur.execute("""
INSERT INTO persons (id, name, age, gender) VALUES
(0001, 'Mike', 25, 'm'),
(0002, 'Hannah', 18, 'f'),
(0003, 'Michelle', 22, 'f'),
(0004, 'Josh', 35, 'm'),
(0005, 'Blake', 55, 'm');
""")

# Committing both scripts to the database
conn.commit()
```

Querying data from the PostgreSQL database with Python

To display the query results we use the method(s) `.fetchone()` , this method returns one result, there are other similar methods such as `.fetchall()` .

Example 1: `.fetchone()` method

In [ ]:
```
# Querying data from the database
cur.execute("""
SELECT *
FROM persons
WHERE name = 'Mike';
""")

# Displaying query, returning the data queried
print(cur.fetchone())
```

```
(1, 'Mike', 25, 'm')
```

Example 2: `.fetchall()` method

In [ ]:
```
# Another query example
cur.execute("""
SELECT *
FROM persons
WHERE age < 25;
""")

# Display query
print(cur.fetchall())
```

```
[(2, 'Hannah', 18, 'f'), (3, 'Michelle', 22, 'f')]
```

Example 3: for-loop on the method `.fetchall()`

In [ ]:
```
# Another query example
cur.execute("""
SELECT *
FROM persons
WHERE age < 25;
""")

# Alternative data recall
for row in cur.fetchall():
    print(row)
```

```
(2, 'Hannah', 18, 'f')
(3, 'Michelle', 22, 'f')
```

Example 4: PostgreSQL query into pandas dataframe

By using the iterable `descrip[0]` in the list comprehension, we extract the first element of each tuple, which represents the column name. This allows us to create a list (columns) containing only the column names from the result set, which is then used to create the Pandas DataFrame with appropriate column names.

In [ ]:
```
# Execute the query
cur.execute("""
SELECT *
FROM persons;
""")

# Fetch all rows of the query result
result = cur.fetchall()

# Get column names from the cursor description
columns = [descrip[0] for descrip in cur.description]

# Create a Pandas DataFrame from the query result
persons_df = pd.DataFrame(result, columns=columns)
```

In [ ]:
```
# Displaying the use of list comprehension
print(cur.description)

# Displaying extracted df
print(persons_df)
```

```
(Column(name='id', type_code=23), Column(name='name', type_code=1043), Column(name='age', type_code=23), Column(name='gender', type_code=1042))
   id      name  age gender
0   1      Mike   25      m
1   2    Hannah   18      f
2   3  Michelle   22      f
3   4      Josh   35      m
4   5     Blake   55      m
```

Passing dynamic placeholders through a SQL statement

The `.execute()` method is suitable when you want to execute a SQL statement with parameters and retrieve the result set or perform data manipulation operations like inserting, updating, or deleting records.

`.mogrify()` : The `.mogrify()` method is used to generate an SQL string with properly escaped and formatted parameter values but does not execute the query. It allows you to examine the resulting SQL string before executing it. This method is useful when you want to inspect the SQL statement with the parameter values interpolated, such as for debugging purposes.

In [ ]:
```
# Creating the query
script = """
SELECT *
FROM persons
WHERE name LIKE %s
AND age >= %s;
"""

# Passing arguments to query
sql = cur.mogrify(script, ('M%', 25))

# Executing the query
cur.execute(sql)

# Displaying query
for row in cur.fetchall():
    print(row)
```

```
(1, 'Mike', 25, 'm')
```

Passing a pandas dataframe into a PostgreSQL server

In order to port a `pd.Dataframe()` into PostgreSQL we have to use the library SQLalchemy as the pandas method `df.to_sql()` does not provide support for PostgreSQL. We first create a dummy dataframe, then create an engine object by using the SQLalchemy function `create_engine()` , after initializing the engine object we pass it to the `df.to_sql()` method via an arguement.

NOTE: `engine = create_engine(f'postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}')`

In [ ]:
```
# Creating Dataframe
df = pd.DataFrame({
    'user': ['Mike', 'Jones', 'Kyle'],
    'bank': ['USAA', 'Ameris', 'BOA']
})


# Importing pandas dataframe into Postgresql
engine = create_engine('postgresql://postgres:admin@localhost:5432/analysis')
df.to_sql('bankuser', engine)
```

Out[ ]:
```
3
```

Closing all sesson connections

In [ ]:
```
# Ending session
cur.close()
conn.close()
```