Accessing SQLite Databases and Using Pandas in a Jupyter Notebook

1. Introduction

In this tutorial, we're going learn how to interact with SQL databases (sqlite, specifically) directly from within Python. So open up a new Jupyter notebook, and make sure that you have copies of our two database files (`rexon_metals.db` and `weather_stations.db`) in the same folder as the notebook.

2. Setting Up Your Environment

- **Installing Necessary Libraries**: You already have Pandas installed, and sqlite3 is part of Python's core packages, so you should be all set.
- One optional thing you can install is `ipython-sql`. We're not actually going to use it in this tutorial, but it might be fun to play with on your own:

```
conda install ipython-sql  # Optional, for running SQL directly within cells
```

3. Connecting to an SQLite Database

- **First, import the `sqlite3` package**:

```python
import sqlite3
```

If that import doesn't work, start Googling...

- **Using sqlite3 to Connect**:

  Now we have to "connect" to the database. This is old school. Think of it as opening the database file.

```python
# Establish a connection to the database.
connection = sqlite3.connect('rexon_metals.db')
```

- **Creating a Cursor Object**:

  A curser object is the object that has an `execute()` method to make your SQL queries and store the results, and methods such as `fetchone()`, `fetchmany()`, and `fetchall()` to access the results. This use of the word "cursor" is from the latin definition of the word – "messenger" – and does not refer to what we think of as a "computer cursor" on a screen.

```python
# Create a cursor object using the connection
cursor = connection.cursor()
```

4. Executing SQL Queries

- **Basic Query Execution**:

  Here, we are going to fetch data and look at it. To do this, were are going to

  1. Use `cursor.execute()` to make the query
  2. Use `cursor.fetchall` to take the table resulting from the query and put it into the ***interable*** variable `results`, and
  3. Loop through `results`, printing the rows of the table as we go

```python
# This example retrieves all entries from the PRODUCT table
cursor.execute("SELECT * FROM product")
results = cursor.fetchall()
for row in results:
    print(row)
```

- **Using Parameters in Queries**:

  We can use the question mark, “?”, to stand for variable names in queries. This gives us the flexibility to write code without hardcoding values in queries.

```python
# Using the placeholder '?' to query with variable names
product_id = 3
cursor.execute("SELECT * FROM product WHERE product_id = ?", (product_id,))
print(cursor.fetchone()) # fetchone gives us a single row
```

The parameters that get plugged in to the query go in a tupple as the last argument. We can use multiple placeholders and parameters if we wish. For example, we can get all the products in a certain price range that start with “S” like this:

```python
# Define your query with placeholders
query = "SELECT * FROM product WHERE price BETWEEN ? AND ? AND description LIKE ?"

# Parameters for the placeholders
min_price = 10
max_price = 14
description_pattern = '%S%'

# Execute the query with the parameters
cursor.execute(query, (min_price, max_price, description_pattern))

# Fetch and print the results
results = cursor.fetchall()
for result in results:
    print(result)
```

Nifty!

> Note: the single comma in the tuple when there is only one placeholder and parameter is an idiosyncrasy of Python – the comma is how Python knows that the thing is a tuple, not just something inside parentheses.

5. Using Pandas to Work with SQL Queries

Using the `cursor` is good to know and may be necessary in some situations, but it's cumbersome. Happily for us, once we open a connection to a database, we issue queries directly using Pandas and have the results assigned directly to a data frame! This makes importing views into Pandas as easy as reading a .csv file!

First, we must import Pandas of course.

```
import pandas as pd
```

- **Fetching Data into DataFrame**:

  Now we just use `pd.read_sql_query()` to grab our view.

```
# Querying data and loading directly into a DataFrame
df = pd.read_sql_query("SELECT * FROM product", connection)
print(df.head())  # Display the first few rows of the DataFrame
```

Too easy!

The query can be as complicated as you need it to be. For clarity and to maintain our sanity, it's best to put long queries in strings, and then pass the string as an argument to `pd.read_sql_query()`.

```
# Make the query a (multi-line) string
query = """
SELECT c.customer_name, co.order_qty * p.price AS total_price
FROM customer_order co
JOIN customer c ON co.customer_id = c.customer_id
JOIN product p ON co.product_id = p.product_id;
"""
df = pd.read_sql_query(query, connection)
print(df)
```

Now, of course, we have the entire power of Pandas (and Seaborn, and MatPlotLib, etc.) at our disposal! You could try one or more of the following to convince yourself that our query is now safely in an actual Pandas dataframe:

- `df.describe()` for basic statistics.

- `df.sort_values(by='total_price', ascending=False)` to sort data.
- `df.groupby('customer_name').sum()` for aggregating data.

6. Closing the Connection

Because SQL is was developed long ago when computers were huge things in basements not featherweight things that we can use on our laps in airplanes, we have to do some old-school cleanup.

- **Important Cleanup Steps**:

```
cursor.close()
connection.close()
```

This probably seems weird but, back in the day, this is how we interacted with ***all*** files. Believe it or not, to use a file, you had to `open` it first in “read”, “write”, or “append” mode. Then, when you were done reading or modifying your file, you had to `close` it. *If you forgot to close it, nobody else could access the file*.

If you are feeling brave, you could experiment with not closing your cursor or connection but, remember, the computer gods are subtle, quick to anger, and can be extremely patient when plotting and exacting their revenge.

7. Exercise

Make a query on the weather stations database, and make some sort of plot from the results (it can be as simple as a histogram of temperature values from on or more stations, or a plot of temperature vs. precipitation, etc.).

8. Summary

This tutorial was short but sweet. In it, we learned how to access SQLite databases with Python in a Jupyter Notebook, making use of Pandas for data manipulation. This gives us a very powerful tool in that, with a minimum of effort, we have the whole world of SQL databases at our Pythonic fingertips!