The Ultimate Guide to Integrating Graph Databases and SQL Databases in a **Streamlit** Web Application

Noam Chen

Technion

March 5, 2025

Abstract

In contemporary data-driven projects, it is frequently necessary to manage and visualize heterogeneous data sources. In particular, **Graph Databases** (e.g., `Neo4j`) excel at representing highly interconnected datasets, whereas **SQL Databases** (e.g., `PostgreSQL`) remain fundamental for structured and transactional information storage.

This tutorial is divided into two main parts to streamline the learning process:

- **Part I (Setup):** Covers the environment setup, local installation of `Neo4j` and `PostgreSQL`, populating them with sample data, and preparing a Python virtual environment for `Streamlit` and other libraries.
- **Part II (Integration & Visualization):** Explains how to integrate both relational (`PostgreSQL`) and graph (`Neo4j`) data into a single `Streamlit` application, covering data querying, conversions (to `NetworkX`), and visualization (with `PyGraphviz` and `PyVis`).

Although we specifically demonstrate **`PostgreSQL`** and **`Neo4j`** in our examples, the methods showcased are equally applicable to any SQL or graph database solution. This comprehensive guide, delivered with a touch of levity, aims to equip both novice and experienced practitioners with the necessary tools and best practices for unifying disparate data sources in a modern analytics pipeline.

Contents

1 Introduction **4**
1.1 Structure of This Tutorial . . . 4
1.2 Main Use-Cases for `Streamlit` . . . 4
1.3 When to Choose **PyGraphviz**, **Graphviz**, or **PyVis**? . . . 4
1.4 Why `Neo4j` and `PostgreSQL`? . . . 4

Part I: Environment Setup & Database Installation **6**

2 What is Streamlit, and Why Use It? **7**
2.1 An Overview of Streamlit . . . 7
2.2 Key Features . . . 7

3 Python Virtual Environments **8**
3.1 Creating and Activating a Virtual Environment . . . 8
3.2 What is `pip`? . . . 9

4 Installing Streamlit and Core Dependencies **10**

5 Installing Graphviz and PyGraphviz **10**
5.1 Graphviz System Package . . . 10
5.2 PyGraphviz in Your Virtual Environment . . . 10

6 Installing and Running Local Databases: `Neo4j` & `PostgreSQL` **11**
6.1 Setting Up `Neo4j` Desktop . . . 11
6.2 `Neo4j` Query Language (`Cypher`) . . . 11
6.2.1 What is `cypher-shell`, and Do I Need It? . . . 11
6.3 Setting Up `PostgreSQL` . . . 12
6.4 Populating the Databases with Example Data . . . 12
6.4.1 Populating `Neo4j` Desktop . . . 13
6.4.2 Populating `PostgreSQL` . . . 13
6.5 Connecting to `Neo4j` and `PostgreSQL` from `Python` . . . 14

Part II: Integration & Visualization **15**

7 Converting `Neo4j` Data into a `NetworkX` Graph **16**
7.1 A Simple Approach: converting `Neo4j` to `NetworkX` . . . 16
7.2 Loading a `DOT` File into `NetworkX` . . . 16

8 Visualizing Graphs in `Streamlit` **18**
8.1 PyGraphviz + `Streamlit` + `NetworkX` . . . 19
8.2 An Interactive Approach (PyVis) + Advanced Customization . . . 20

9 Visualizing SQL Data in `Streamlit` **22**

10 Building a Combined Graph + SQL `Streamlit` Application **23**

11 Advanced Tips & Troubleshooting **25**
11.1 `PyGraphviz` Not Finding `Graphviz` . . . 25
11.2 `Neo4j` Query Issues . . . 25
11.3 `PostgreSQL` Access Problems . . . 25
11.4 `NetworkX` Graph Too Large? . . . 25

12 Conclusion **26**

1 Introduction

Who is this tutorial for? This guide is intended primarily for **students** and **developers** who want to build a `Streamlit` application that integrates both **relational databases** (like `PostgreSQL`) and **graph databases** (like `Neo4j`). It's especially useful for those seeking a straightforward way to visualize and interact with data from different sources in one dashboard.

1.1 Structure of This Tutorial

Part I (Setup): We introduce `Streamlit`, virtual environments, and walk you through installing and populating `Neo4j` (Desktop version) and `PostgreSQL`. By the end of Part I, you'll have a fully configured environment with sample data in both databases.

Part II (Integration & Visualization): We show how to connect `Neo4j` and `PostgreSQL` from `Python` and build a `Streamlit` application that fetches, displays, and visualizes the data. This includes examples of graph conversions to `NetworkX`, as well as rendering in `Streamlit` via `PyGraphviz` or `PyVis`.

1.2 Main Use-Cases for `Streamlit`

- **Data Dashboards** for academic projects, prototypes, or internal apps combining multiple data sources.
- **Interactive Graph Exploration** when you have relationship-centric data (e.g., social networks) and still need relational queries (e.g., transaction logs).
- **Learning/Teaching** environment to quickly spin up a user interface without advanced web dev knowledge.

1.3 When to Choose PyGraphviz, Graphviz, or PyVis?

- **Graphviz** is a `C/C++` library and set of command-line tools for generating graph layouts from `.dot` files.
- **PyGraphviz** is a `Python` wrapper over Graphviz. Perfect for creating static images or diagrams you can embed in `Streamlit`.
- **PyVis** is a different approach—using `JavaScript` libraries under the hood to allow an **interactive** view of your graph in the browser, with node dragging, zooming, etc.

Use **PyGraphviz** if you prefer static or quick layouts. Use **PyVis** if you need an **interactive** user experience inside `Streamlit`.

1.4 Why `Neo4j` and `PostgreSQL`?

- **Neo4j**: well-known, robust `graph database` for exploring and visualizing relationships (e.g., social, recommendation systems).
- **PostgreSQL**: open-source, feature-rich `SQL database` that handles structured data and supports advanced queries.

They're widely used in industry and academia, and combining them covers a **broad range of use-cases**:

- **Social Network Analysis** (connections in `Neo4j`, user profiles in `PostgreSQL`)

- **E-Commerce/Transactions** (product relationships in `Neo4j`, user transactions in `PostgreSQL`)
- **Knowledge Graphs** (hierarchical or linked data in `Neo4j`, logs/tables in `PostgreSQL`)

We highly recommend checking out the official `Streamlit` getting-started guides at Streamlit Official Docs for a more in-depth introduction to `Streamlit`'s features and capabilities.

Part I: Environment Setup & Database Installation

2 What is Streamlit, and Why Use It?

2.1 An Overview of Streamlit

Streamlit is a lightweight `Python` library for rapidly building interactive web apps without wrestling an alligator of front-end frameworks. You write a simple `Python` script (`app.py`), run:

```
python -m streamlit run app.py --server.port 8501 --server.address localhost
```

and—*presto!*—you have a local web server at `http://localhost:8501` with a fully functional UI. Streamlit handles the heavy lifting of layout, widgets, and reactive updates.

2.2 Key Features

- **Minimal Boilerplate**: No specialized HTML or JS knowledge required.
- **Live Reloading**: Edit your script, refresh your browser, and changes appear.
- **Interactive Widgets**: Sliders, checkboxes, text inputs, file uploads, etc.
- **Seamless Integration**: Works with `pandas`, `matplotlib`, `Plotly`, and more. We'll specifically look at integrating **PyGraphviz** and **NetworkX**.

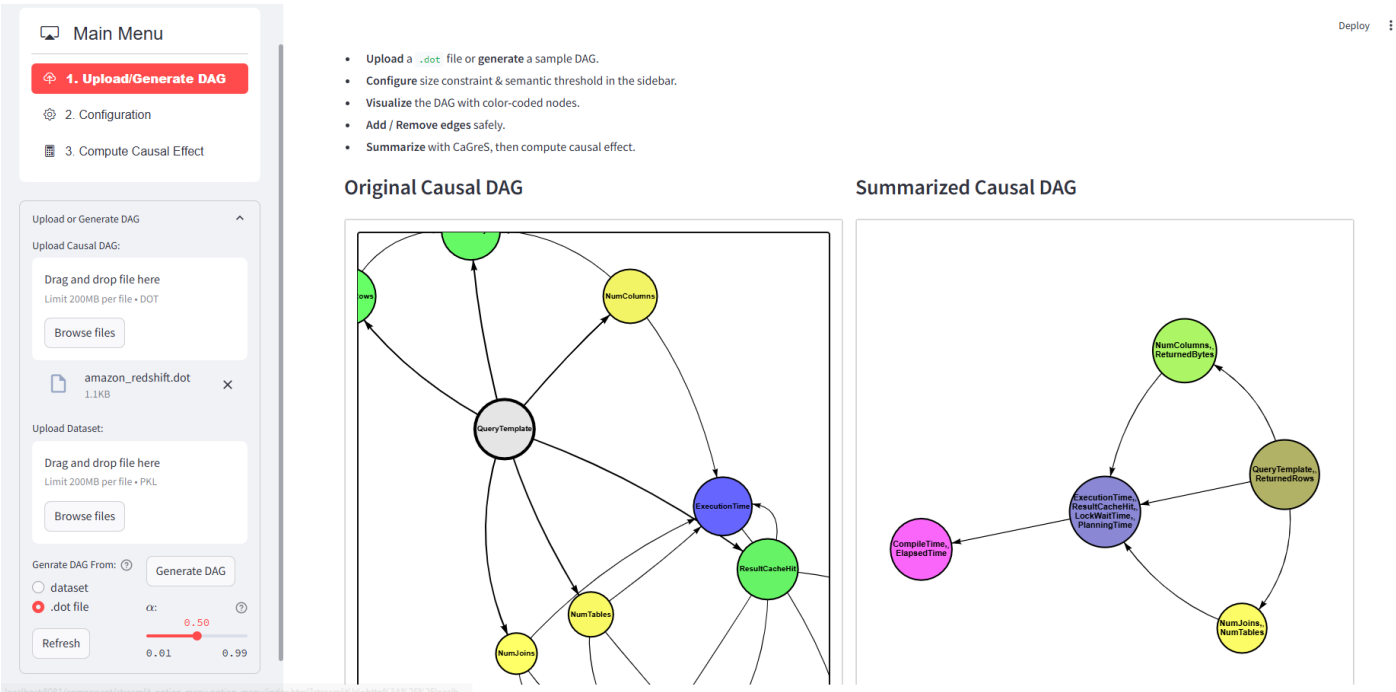

Figure 1: A basic Streamlit UI in action

3 Python Virtual Environments

Python Virtual Environments

The venv module supports creating lightweight `"virtual environments"`, each with their own independent set of `Python` packages installed in their site directories. A `virtual environment` is created on top of an existing `Python` installation, known as the `virtual environment`'s "base" `Python`, and may optionally be isolated from the packages in the base environment, so only those explicitly installed in the `virtual environment` are available.

- They let you create isolated "bubbles" for your project's dependencies, so installing a new package for one project won't break another.
- They help manage different versions of libraries or `Python` itself.
- They are recommended for almost any serious `Python` work to avoid dependency chaos (a.k.a. "*ModuleNotFoundError* meltdown").

3.1 Creating and Activating a Virtual Environment

Creating & Activating a Virtual Environment

```
# 1. Create a new virtual environment named "myenv"
python -m venv myenv

# 2. Activate the environment:
# On Linux/Mac:
source myenv/bin/activate

# On Windows (PowerShell or cmd):
myenv\Scripts\activate
```

3.2 What is pip?

`pip` is the default package manager for `Python`, included with most modern `Python` installations (versions later than 3.4). It helps you install libraries from the *Python Package Index* (PyPI) and manage dependencies automatically.

Installing Packages and Freezing Requirements

```
# Install a couple of libraries
pip install requests flask

# Check installed packages
pip list

# Freeze (record) exact installed versions into requirements.txt
pip freeze > requirements.txt
```

When working in a team or switching computers, you can easily recreate the same environment:

Recreating an Environment from Requirements

```
python -m venv myenv
source myenv/bin/activate
pip install -r requirements.txt
```

Deactivating the Virtual Environment

```
# On Linux/Mac or simply close your terminal
(myenv) deactivate

# On Windows (PowerShell or cmd):
(myenv) myenv\Scripts\deactivate
```

4 Installing Streamlit and Core Dependencies

With your **virtual environment** activated, let's install **Streamlit** itself. We'll also install **NetworkX**, since it's integral to graph manipulation in our tutorial:

```
pip install streamlit networkx
```

You can test your `Streamlit` installation by creating a file `app.py`:

Basic Streamlit Application

```python
import streamlit as st

st.title("Hello Streamlit!")
st.write("This is my first Streamlit app, woohoo!")
```

Then run:

```
streamlit run app.py
```

Open `http://localhost:8501` in your browser, and watch the magic happen.

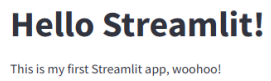

Figure 3: The generated Streamlit web app

5 Installing Graphviz and PyGraphviz

Graphviz is a suite of command-line tools for rendering graphs (often from ".dot" files). **PyGraphviz** is a `Python` library that wraps these tools, letting you generate images programmatically.

5.1 Graphviz System Package

Install the system-level **Graphviz**:

- **`Ubuntu/Debian`**:

  ```
  sudo apt-get update
  sudo apt-get install graphviz
  ```

- **`macOS`** + `Homebrew`:

  ```
  brew install graphviz
  ```

- **`Windows`**:
  - Download the installer from graphviz.org/download/
  - Ensure `dot.exe` is on your `PATH`.

5.2 PyGraphviz in Your Virtual Environment

Once Graphviz is installed, jump back into `(myenv)`:

```
pip install pygraphviz
```

If `PyGraphviz` can't find the `dot` executable, ensure your system `PATH` includes Graphviz. Refer to the `CausalDAG_Sum` README if troubleshooting is needed.

6 Installing and Running Local Databases: Neo4j & PostgreSQL

It's time to get both a graph database and an SQL database up and running on your machine. In this tutorial, we focus on `Neo4j` (Desktop version) for graphs and `PostgreSQL` for relational data.

6.1 Setting Up Neo4j Desktop

- **Download and Install:** Head to neo4j.com/download and choose `Neo4j Desktop`.
- **Launch Neo4j Desktop:** After installation, open the app. You can create or import a "project" in the left sidebar.
- **Create a Local Database:** Inside your new project, click "Add" → "Local DB." You can specify the name and version of `Neo4j`.
- **Start the Database**: Once created, you'll see a Start button. `Neo4j Desktop` automatically handles the `bolt://` server. By default, it's at `bolt://localhost:7687`.
- **Credentials:** The default credentials might be `neo4j / password`, but you'll be asked to change them upon first login if you haven't already.

6.2 Neo4j Query Language (Cypher)

`Cypher` is a declarative language for working with nodes and relationships. It allows you to create, read, update, and delete data by focusing on pattern matching rather than complex joins:

Sample Cypher Commands

```
// Create a node labeled "Person" with name property "Alice"
CREATE (p:Person {name: 'Alice'})

// Match the node with name "Alice"
MATCH (p:Person {name: 'Alice'})
RETURN p
```

6.2.1 What is cypher-shell, and Do I Need It?

`cypher-shell` is a command-line interface for running `Cypher` queries. In `Neo4j Desktop`, you can use its built-in "Open Browser" or "Open Terminal" to run queries. If you want a standalone shell, you'd install it from `Neo4j`'s repositories. However, the `Neo4j Desktop` GUI is sufficient for most local use cases.

6.3 Setting Up PostgreSQL

- **Ubuntu/Debian**:

  ```
  sudo apt-get install postgresql
  ```

- **macOS** + Homebrew:

  ```
  brew install postgresql
  ```

- **Windows**:

  – Download from postgresql.org/download/ and run the wizard.

- **Start the Server:**

  ```
  sudo service postgresql start
  # or
  brew services start postgresql
  ```

- **Create a Database and User:**

  ```
  createdb mydb
  psql -c "CREATE USER postgres WITH SUPERUSER PASSWORD 'mypassword';"
  ```

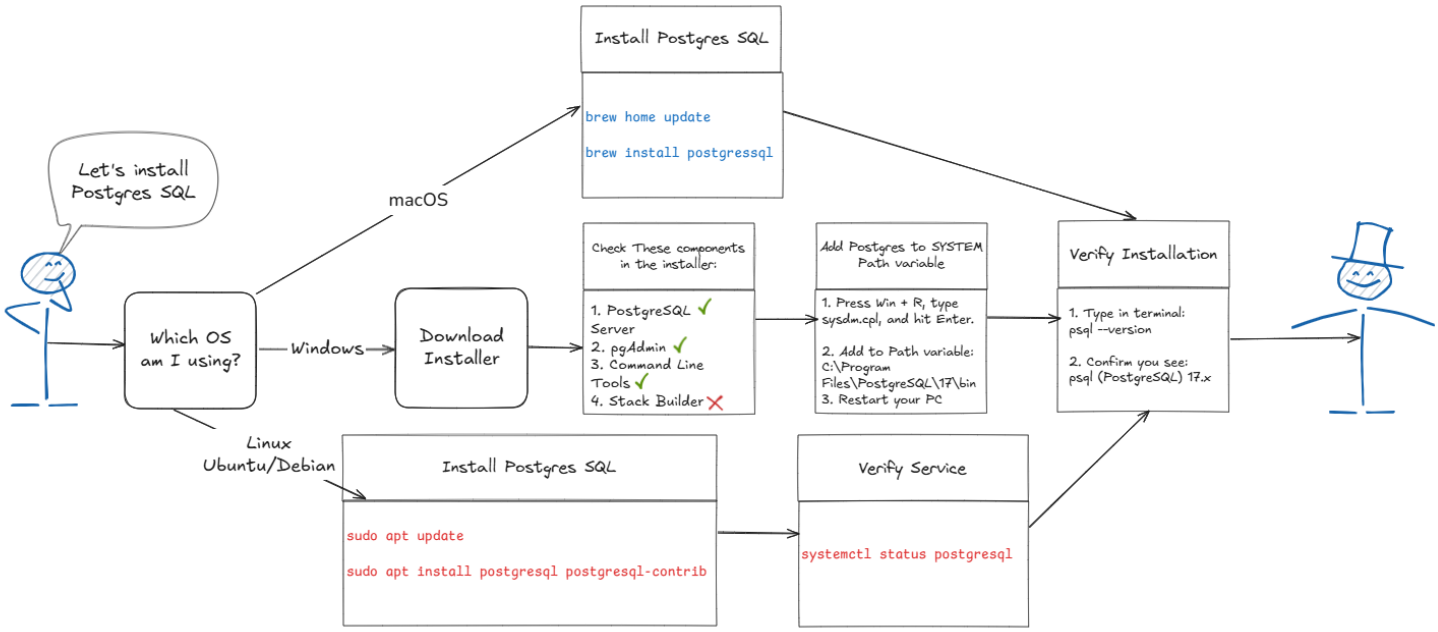

Figure 4: Local installation summary for PostgreSQL

6.4 Populating the Databases with Example Data

Having installed Neo4j Desktop and PostgreSQL, let's insert some simple data so that, in the next sections, we can demonstrate queries, visualizations, and more.

6.4.1 Populating Neo4j Desktop

Open the Neo4j Browser in your local database and run:

Populating Neo4j with Sample Data

```
// before running that query run :use neo4j
CREATE (alice:Person {name: "Alice", age: 30})
CREATE (bob:Person {name: "Bob", age: 40})
CREATE (charlie:Person {name: "Charlie", age: 25})
WITH alice, bob, charlie
MATCH (a:Person {name: "Alice"}), (b:Person {name: "Bob"})

CREATE (a)-[:KNOWS]->(b)
WITH a, b
MATCH (b:Person {name: "Bob"}), (c:Person {name: "Charlie"})
CREATE (b)-[:KNOWS]->(c)
```

You'll have three Person nodes (Alice, Bob, Charlie) plus relationships (Alice)-[:KNOWS]->(Bob) and (Bob)-[:KNOWS]->(Charlie).

6.4.2 Populating PostgreSQL

PostgreSQL provides the psql CLI. After creating (or identifying) a database mydb and user postgres, do:

Populating PostgreSQL with Sample Data

```
-- connect to the db:
psql -U postgres -d mydb

CREATE TABLE example_table (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  amount INT
);

INSERT INTO example_table (name, amount) VALUES
  ('Alice', 100),
  ('Bob', 200),
  ('Charlie', 150);

\q -- quit psql
```

Now you have an example_table with three records. That's sufficient data to illustrate queries in the upcoming sections.

6.5 Connecting to Neo4j and PostgreSQL from Python

With local databases seeded with example data, let's connect via Python.
Before diving into code examples, ensure to install the required packages with:

```
pip install py2neo psycopg2
```

Make sure you're in (myenv) in order to maintain the isolation of the execution environment.

Using py2neo to Query Neo4j

```python
from py2neo import Graph
def connect_neo4j(uri, user, password):
    """
    Connects to a Neo4j database and returns a Graph object.
    """
    graph = Graph(uri, auth=(user, password))
    return graph

def sample_cypher_query(graph):
    query = """
    MATCH (n)
    RETURN n
    LIMIT 5
    """
    return graph.run(query).data()

if __name__ == "__main__":
    graph = connect_neo4j("bolt://localhost:7687", "neo4j", "your-password")
    data = sample_cypher_query(graph)
    print(data)
```

Similarly, for PostgresSQL:

Using psycopg2 to Query PostgreSQL

```python
import psycopg2

def connect_postgres(dbname, user, password, host="localhost", port=5432):
    """
    Connects to a PostgreSQL database and returns a connection object.
    """
    conn = psycopg2.connect(
        dbname=dbname,
        user=user,
        password=password,
        host=host,
        port=port
    )
    return conn

def fetch_example_table(conn):
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM example_table LIMIT 5;")
        return cur.fetchall()

if __name__ == "__main__":
    conn = connect_postgres("mydb", "postgres", "mypassword")
    rows = fetch_example_table(conn)
    print(rows)
```

Part II: Integration & Visualization

In the second part, we'll show how to work with your newly installed databases from `Streamlit`, highlighting how to convert `Neo4j` data into `NetworkX` and how to visualize everything (graphs, SQL tables) in a single web application.

7 Converting Neo4j Data into a NetworkX Graph

NetworkX is a Python library for analyzing/manipulating graph structures. If your data starts in Neo4j, you'll need to move it into a networkx.DiGraph for advanced analyses or visualizations.

7.1 A Simple Approach: converting Neo4j to NetworkX

NetworkX 101

NetworkX lets you create a graph object like nx.DiGraph(), then add nodes/edges. It supports algorithms for shortest path, centrality, clustering, etc.

Fetching Graph Data from Neo4j into NetworkX

```python
import networkx as nx
from py2neo import Graph

def neo4j_to_networkx(uri, user, password):
    # 1. Connect
    graph = Graph(uri, auth=(user, password))

    # 2. Query relationships (assumes "name" property)
    query = """
    MATCH (a)-[r]->(b)
    RETURN a.name AS source, b.name AS target
    """
    results = graph.run(query).data()

    # 3. Construct NetworkX DiGraph
    G = nx.DiGraph()
    for row in results:
        source = row['source']
        target = row['target']
        G.add_node(source)
        G.add_node(target)
        G.add_edge(source, target)
    return G

if __name__ == "__main__":
    G = neo4j_to_networkx("bolt://localhost:7687","neo4j","password")
    print(f"Nodes: {list(G.nodes())}")
    print(f"Edges: {list(G.edges())}")
```

7.2 Loading a DOT File into NetworkX

If you already have a .dot file (e.g., from another system), you can load it via read_dot:

Loading a DOT File into NetworkX

```python
import networkx as nx
from networkx.drawing.nx_pydot import read_dot
import tempfile
import logging
import streamlit as st

logger = logging.getLogger(__name__)

def load_dag_from_file(file):
    """
    Loads a DAG from a DOT file by parsing the file content and
    passing the file path to read_dot().
    """
    try:
        content = file.getvalue().decode("utf-8")
        logger.debug("Received DOT file content for parsing.")

        with tempfile.NamedTemporaryFile(delete=False, suffix='.dot') as tmp_file:
            tmp_file.write(content.encode('utf-8'))
            temp_filepath = tmp_file.name
            logger.debug(f"Temporary DOT file at: {temp_filepath}")

        G = read_dot(temp_filepath)
        DG = nx.DiGraph(G)

        if not is_valid_dag(DG):
            st.warning("Uploaded graph is not a valid Directed Acyclic Graph
            ↪  (DAG).")
            logger.warning("Invalid DAG.")
            return None
        else:
            logger.info("Uploaded graph is a valid DAG.")

        return DG

    except Exception as e:
        logger.exception(f"Error loading DOT file: {e}")
        return None

def is_valid_dag(G: nx.DiGraph) -> bool:
    return nx.is_directed_acyclic_graph(G)
```

8 Visualizing Graphs in Streamlit

Below, we build a Streamlit app that shows NetworkX graph data either as a static image (PyGraphviz) or an interactive view (PyVis).

8.1 PyGraphviz + Streamlit + NetworkX

Using PyGraphviz in Streamlit

```python
# File: app_graphviz.py
import streamlit as st
import networkx as nx
import pygraphviz as pgv
from py2neo import Graph

def convert_to_networkx(neo4j_graph):
    """
    Similar logic as 'neo4j_to_networkx', returning a networkx.DiGraph
    """
    query = """
    MATCH (a)-[r]->(b)
    RETURN a.name AS source, b.name AS target
    """
    results = neo4j_graph.run(query).data()

    G = nx.DiGraph()
    for row in results:
        src = row['source']
        tgt = row['target']
        G.add_node(src)
        G.add_node(tgt)
        G.add_edge(src, tgt)
    return G

def draw_graph_via_pygraphviz(G):
    A = nx.nx_agraph.to_agraph(G)
    A.layout(prog='dot')
    A.draw('temp_graph.png')  # static .png

def main():
    st.title("NetworkX + PyGraphviz in Streamlit")

    # Connect to Neo4j
    graph = Graph("bolt://localhost:7687", auth=("neo4j","password"))
    G = convert_to_networkx(graph)

    if len(G.nodes()) == 0:
        st.write("No nodes found in the graph. Insert some data in Neo4j!")
        return

    # Draw & display
    draw_graph_via_pygraphviz(G)
    st.image("temp_graph.png", caption="My Graph Visualization")

if __name__ == "__main__":
    main()
```

To run:

```
streamlit run app_graphviz.py
```

Open `http://localhost:8501`, and you'll see a static image of your graph.

Figure 5: Streamlit displaying a PyGraphviz-rendered graph (placeholder).

8.2 An Interactive Approach (PyVis) + Advanced Customization

Using PyVis in Streamlit (Advanced)

```python
# File: app_pyvis_advanced.py

import streamlit as st
import networkx as nx
import streamlit.components.v1 as components
from pyvis.network import Network

def visualize_pyvis_advanced(G: nx.DiGraph,
                             height="700px",
                             width="100%",
                             physics=True):
    net = Network(height=height, width=width, directed=True)
    net.from_nx(G)

    # Example node colorization
    for node in net.nodes:
        if node["id"] == "Alice":
            node["color"] = "red"
        elif node["id"] == "Bob":
            node["color"] = "blue"
        else:
            node["color"] = "green"

    # Toggle physics
    if physics:
        net.barnes_hut(
            central_gravity=0.1,
            spring_length=220,
            spring_strength=0.02,
            damping=0.09,
            overlap=0
        )
    else:
        net.toggle_physics(False)

    return net.generate_html(notebook=False)

def main():
    st.title("Advanced PyVis Demo")
    G = nx.DiGraph()
    G.add_edge("Alice","Bob")
    G.add_edge("Bob","Charlie")
    G.add_edge("Charlie","Dora")

    physics_enabled = st.sidebar.checkbox("Enable Physics?", value=True)
    html_str = visualize_pyvis_advanced(G, physics=physics_enabled)
    components.html(html_str, height=700, scrolling=True)

if __name__ == "__main__":
    main()
```

This approach lets users explore the graph with zoom, drag, and real-time physics layout—very handy for big or complex networks.

9 Visualizing SQL Data in Streamlit

If you want to display table data from PostgreSQL in your Streamlit app:

Querying PostgreSQL in Streamlit

```
# File: app_sql.py
import streamlit as st
import psycopg2
import pandas as pd

def main():
    st.title("SQL + Streamlit")

    conn = psycopg2.connect(
        dbname="mydb",
        user="postgres",
        password="mypassword",
        host="localhost",
        port=5432
    )

    df = pd.read_sql_query("SELECT * FROM example_table LIMIT 10;", conn)
    st.dataframe(df)
    conn.close()

if __name__ == "__main__":
    main()
```

10 Building a Combined Graph + SQL Streamlit Application

Below is a minimal app that queries both **Neo4j** (converted to `NetworkX`) and **PostgreSQL**, then displays them in a single `Streamlit` page:

Combining Neo4j and PostgreSQL in a Single Streamlit App

```python
# File: app_combined.py
import streamlit as st
import psycopg2
import pandas as pd
import networkx as nx
from py2neo import Graph
import pygraphviz as pgv
import os

def draw_graph_via_pygraphviz(G, filename="temp_graph.png"):
    A = nx.nx_agraph.to_agraph(G)
    A.layout(prog='dot')
    A.draw(filename)

def main():
    st.title("Neo4j + PostgreSQL + Streamlit")

    # Graph Data from Neo4j
    st.subheader("Graph Data from Neo4j")
    neo4j_graph = Graph("bolt://localhost:7687", auth=("neo4j","password"))
    query = """
    MATCH (a)-[r]->(b)
    RETURN a.name AS source, b.name AS target
    LIMIT 10
    """
    results = neo4j_graph.run(query).data()
    G = nx.DiGraph()
    for row in results:
        G.add_edge(row["source"], row["target"])

    if len(G.nodes()) > 0:
        draw_graph_via_pygraphviz(G)
        st.image("temp_graph.png")
        # os.remove("temp_graph.png") # optional cleanup
    else:
        st.write("No graph data found in Neo4j.")

    # SQL Data from PostgreSQL
    st.subheader("SQL Data from PostgreSQL")
    conn = psycopg2.connect(
        dbname="mydb", user="postgres",
        password="mypassword", host="localhost", port=5432
    )
    df = pd.read_sql_query("SELECT * FROM example_table LIMIT 10;", conn)
    st.dataframe(df)
    conn.close()

if __name__ == "__main__":
    main()
```

Run:

```
streamlit run app_combined.py
```

One app + two data sources = a unified Streamlit dashboard.

11 Advanced Tips & Troubleshooting

11.1 `PyGraphviz` Not Finding `Graphviz`

Ensure `dot` (the main `Graphviz` executable) is on your system path. Test by running `dot -V` in your terminal. If missing, reinstall or add it to `PATH`. See CasualDAG_Sum for more pointers.

11.2 `Neo4j` Query Issues

- Double-check `auth=("neo4j","password")` or your changed credentials.
- Ensure the server runs on `bolt://localhost:7687`.

11.3 `PostgreSQL` Access Problems

- Confirm user/password in `psycopg2.connect`.
- Verify `dbname`, `host`, and `port`.
- If connecting remotely, `pg_hba.conf` might need adjusting.

11.4 `NetworkX` Graph Too Large?

For huge graphs, consider subsetting or using an interactive approach (like `PyVis`) to avoid performance issues.

12 Conclusion

Congratulations! You've reached the end of this two-part tutorial. Here's a quick summary of what we achieved:

- **Part I (Setup)**:
  - Installed and used `virtual environments` to keep dependencies neat
  - Set up `Streamlit` for rapid UI creation
  - Installed `Graphviz` & `PyGraphviz` for network visualization
  - Deployed local instances of `Neo4j` (Desktop) and `PostgreSQL`, each handling distinct but complementary data needs
  - Populated both databases with sample data
- **Part II (Integration & Visualization)**:
  - Converted `Neo4j` data into `NetworkX` for advanced graph operations
  - Visualized graphs in `Streamlit` using `PyGraphviz` (static) and `PyVis` (interactive)
  - Displayed SQL data from `PostgreSQL` with minimal fuss, harnessing `st.dataframe`
  - Combined both data sources into one cohesive `Streamlit` app

With these tools, you can now create powerful, unified dashboards that blend graph relationships and relational data in a single, user-friendly environment. Customize your visualizations, add interactive widgets, or scale up to larger deployments—wherever your data takes you!

Happy coding, querying, and graphing!
For further reading or deeper dives into `Streamlit`, check out their official documentation and getting-started guides at: Streamlit Official Docs.