# Assignment No. 10

**Group ID:** 2018BCGRP10
**Batch:** T2
**PRN No:** 2018BTECS00072 & 2018BTECS00086

- **Title: Neo4j Graph Database**

- **Aim:** To design the python based desktop application using Neo4j Graph Database

  Consider the “Research Papers Database” scenario as follows :

  The research papers have authors (often more than one). Most papers have a classification (what the paper is about). The classifications form a hierarchy in several levels (for example, the classification “Databases” has the sub-classifications “Relational” and “Object-Oriented”). A paper usually has a list of references, which are other papers. These are called citations.

  1. Design/model the graph database using Neo4j for above scenario.
  2. Download the raw data from Cora Research Paper Classification Project : http://people.cs.umass.edu/~mccallum/data.html The database contains approximately 25,000 authors, 37,000 papers and 220,000 relationships.
  3. Load this data using Neo4j Data Browser
  4. Design the python based desktop application for any kind of search on above database. The application should able to answer queries like

a) Does paper A cite paper B? If not directly, does paper A cite a paper which in its turn cites paper B? And so on, in several levels.
b) Show the full classification of a paper (for example, Databases / Relational)

- ## Introduction:

**What is a Graph Database?**

A graph database is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a pre-defined model.

Instead, the data is stored like we first draw it out - showing how each individual entity connects with or is related to others.

**Why Graph Databases?**

We live in a connected world! There are no isolated pieces of information, but rich, connected domains all around us. Only a database that natively embraces relationships is able to store, process, and query connections efficiently. While other databases compute relationships at query time through expensive JOIN operations, a graph database stores connections alongside the data in the model.

Accessing nodes and relationships in a native graph database is an efficient, constant-time operation and allows you to quickly traverse millions of connections per second per core.

Independent of the total size of your dataset, graph databases excel at managing highly-connected data and complex queries. With only a pattern and a set of starting points, graph databases explore the

neighboring data around those initial starting points — collecting and aggregating information from millions of nodes and relationships — and leaving any data outside the search perimeter untouched.

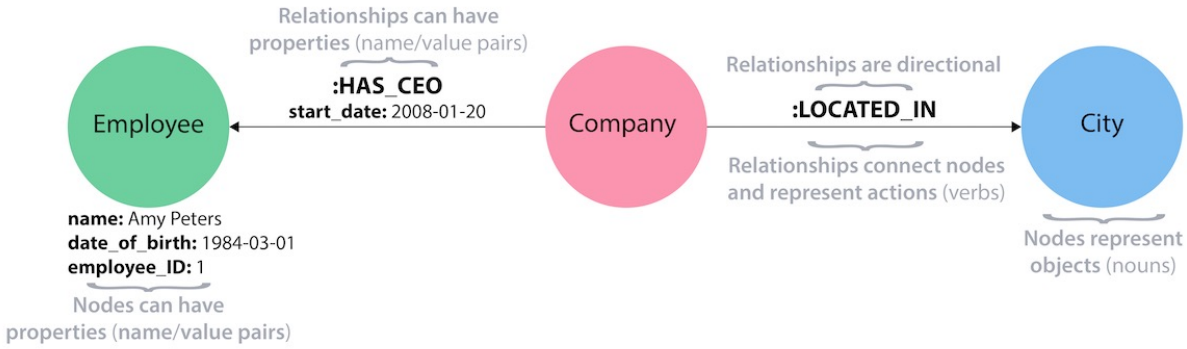
## The Property Graph Model

As with most technologies, there are few different approaches to what makes up the key components of a graph database. One such approach is the property graph model, where data is organized as nodes, relationships, and properties (data stored on the nodes or relationships).

***Nodes*** are the entities in the graph. They can hold any number of attributes (key-value pairs) called *properties*. Nodes can be tagged with *labels*, representing their different roles in your domain. Node labels may also serve to attach metadata (such as index or constraint information) to certain nodes.

***Relationships*** provide directed, named, semantically-relevant connections between two node entities (e.g. Employee *WORKS_FOR* Company). A relationship always has a direction, a type, a start node, and an end node. Like nodes, relationships can also have properties. In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths. Due to the efficient way relationships are stored, two nodes can share any number or type of relationships without sacrificing performance. Although they are stored in a specific direction, relationships can always be navigated efficiently in either direction.

## Building blocks of the property graph model :




## What is Neo4j?

Neo4j is a native graph database platform, built from the ground up to leverage not only data but also data *relationships.* Neo4j *connects* data as it's stored, enabling queries never before imagined, at speeds never thought possible.

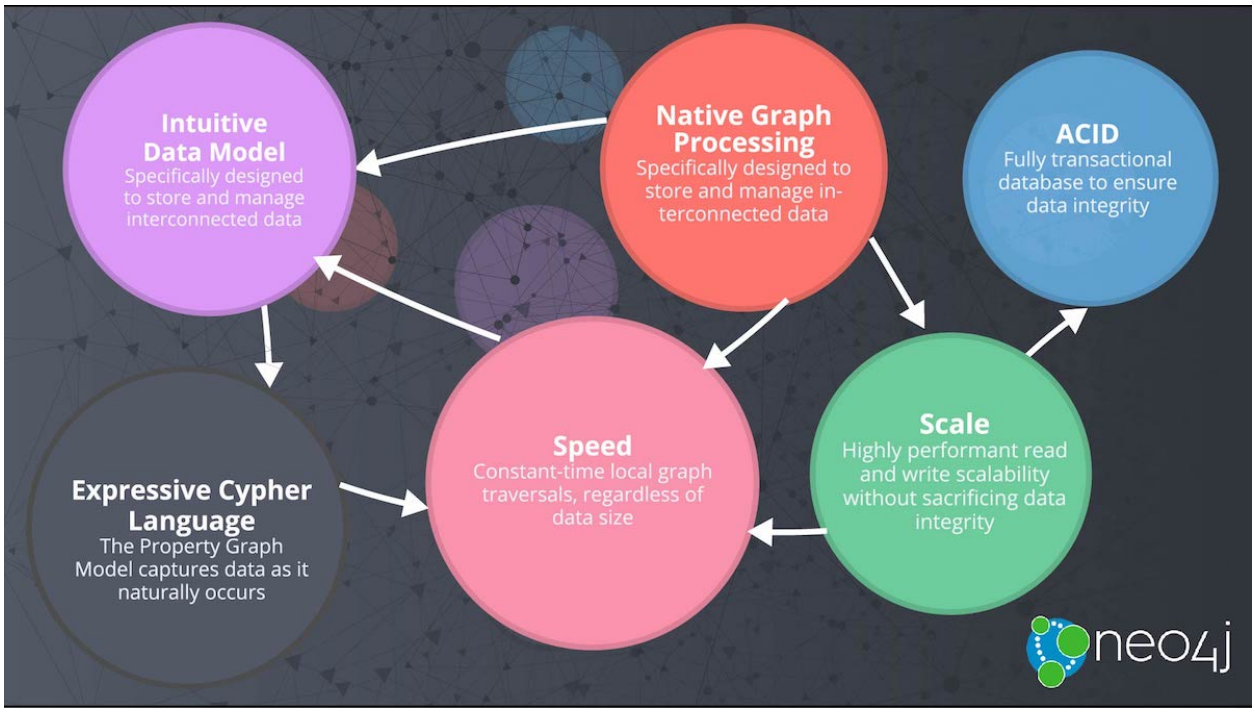
Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend for your applications. Initial development began in 2003, but it has been publicly available since 2007. The source code, written in Java and Scala, is available for free on GitHub or as a user-friendly desktop application download. Neo4j has both a Community Edition and Enterprise Edition of the database. The Enterprise Edition includes all that Community Edition has to offer, plus extra enterprise requirements such as backups, clustering, and failover abilities.

Neo4j is referred to as a native graph database because it efficiently implements the property graph model down to the storage level. This means that the data is stored exactly as you whiteboard it, and the database uses pointers to navigate and traverse the graph. In contrast to graph processing or in-memory libraries, Neo4j also provides full database characteristics, including ACID transaction

compliance, cluster support, and runtime failover - making it suitable to use graphs for data in production scenarios.

**Some of the following particular features make Neo4j very popular among developers, architects, and DBAs:**

- Cypher, a declarative query language similar to SQL, but optimized for graphs. Now used by other databases like SAP HANA Graph and Redis graph via the openCypher project.
- Constant time traversals in big graphs for both depth and breadth due to efficient representation of nodes and relationships. Enables scale-up to billions of nodes on moderate hardware.
- Flexible property graph schema that can adapt over time, making it possible to materialize and add new relationships later to shortcut and speed up the domain data when the business needs change.
- Drivers for popular programming languages, including Java, JavaScript, .NET, Python, and many more.



## Neo4j Use Cases

Neo4j is used today by thousands of companies and organizations in almost all industries, including financial services, government, energy, technology, retail, and manufacturing.

The thriving, active community surrounding the technology continues to help us improve our product and services for developers and businesses alike.

- **Functional Block Diagram:**

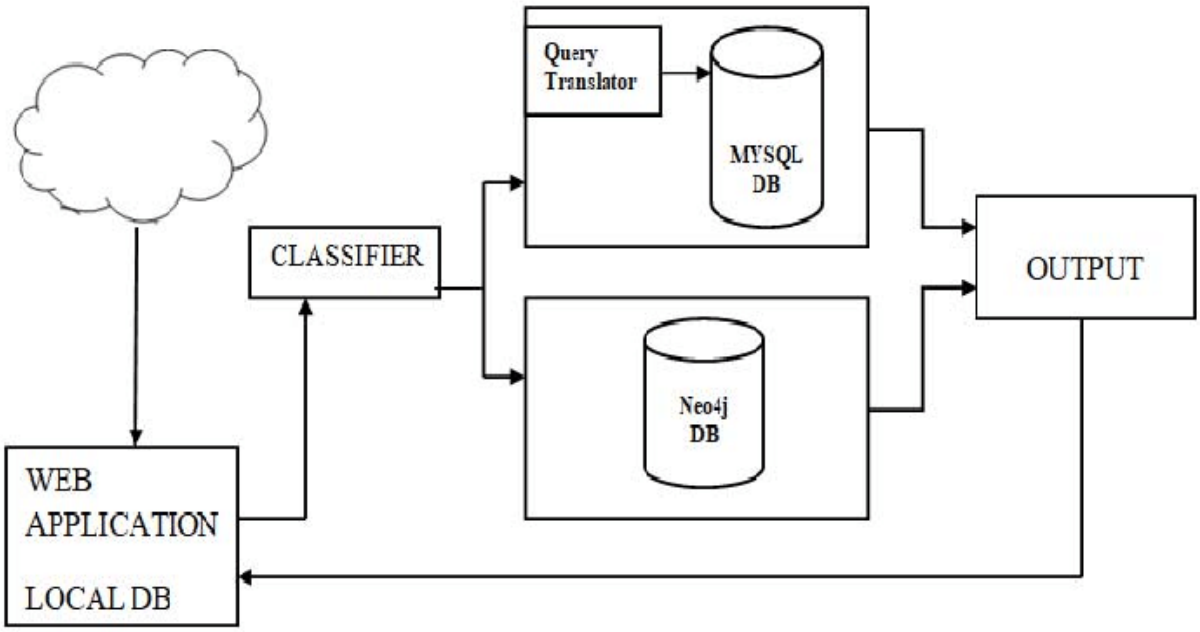



Fig. Neo4j Database in application

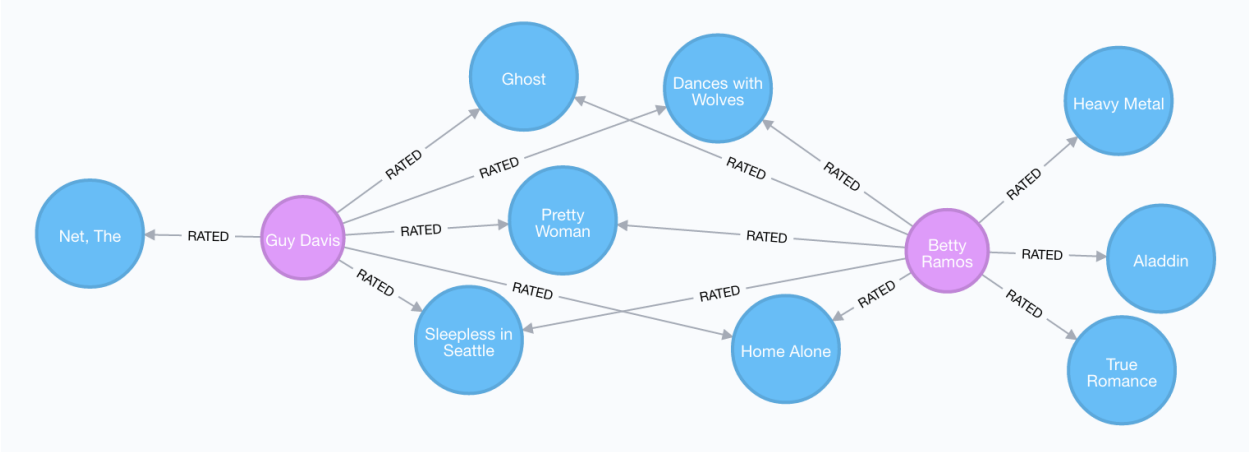
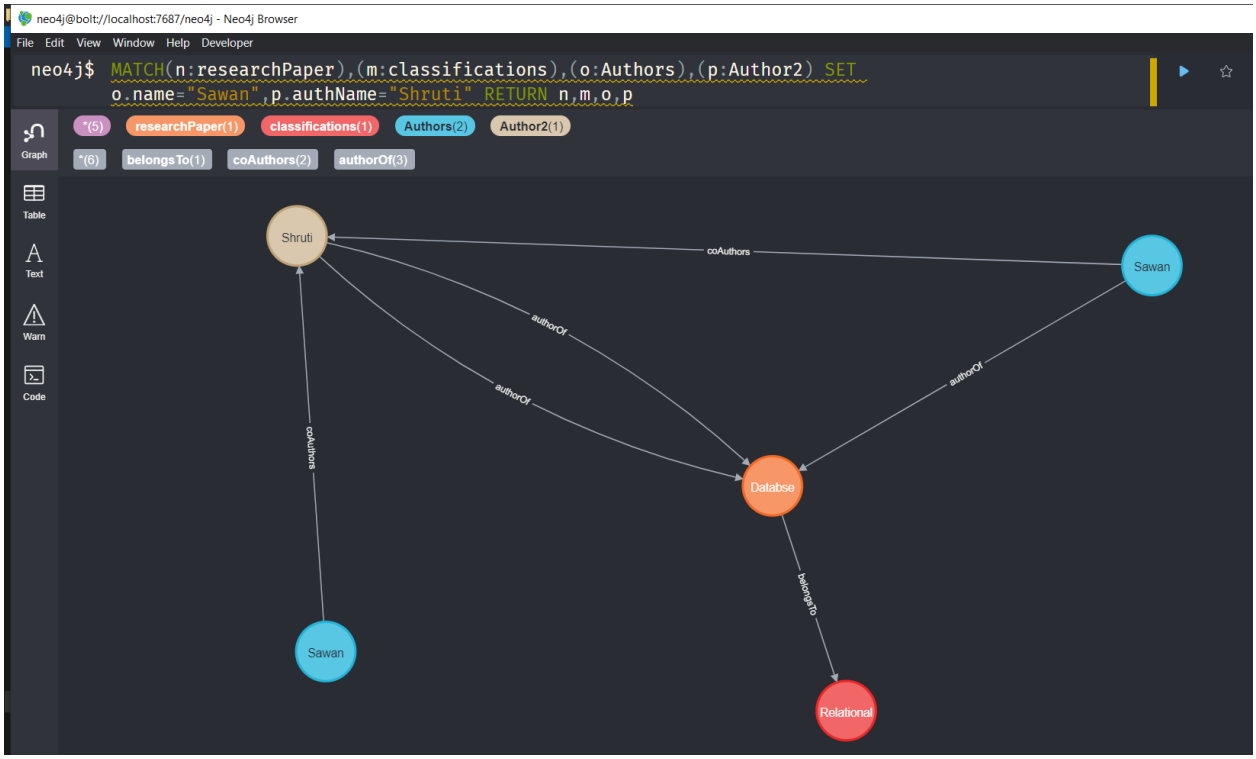



Fig. Personalized Product Recommendations with Neo4j

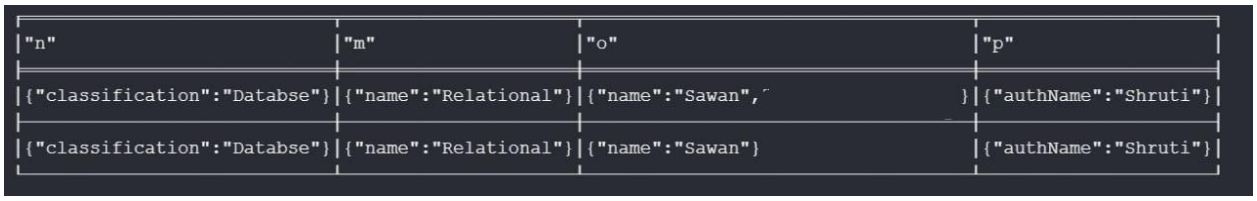
- **Screenshots Of Experiment:**

Property Graph Model:

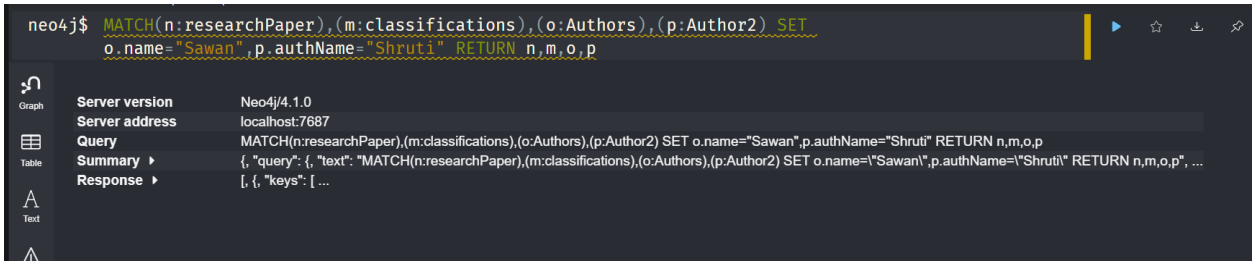



Entries in the database:


Table showing data about Authors:




Queries used for the Neo4j Graph Database:




- **Conclusion:**

This project demonstrated a python based desktop application with Neo4j Graph Database.

- **References:**

1. https://neo4j.com/
2. https://www.youtube.com/watch?v=5TI8WcaqZoc&list=PL9HI4pk2FsvWM9GWaguRhlCQ-pa-ERd4U
3. https://neo4j.com/graph-databases-book/
4. https://rubygarage.org/blog/neo4j-database-guide-with-use-cases
5. https://neo4j.com/developer/graph-database/
6. https://www.tutorialspoint.com/neo4j/neo4j_overview.htm