Paper DV01

Graphical overview for understanding, planning and execution of statistical programming activities using Neo4j.

Madhusudhan Reddy Mule Venkata, Novo Nordisk, Soeborg, Denmark
Kamal Chadha, Novo Nordisk, Boston, USA

ABSTRACT

The world of statistical programming is changing dynamically to meet the complex requirements of clinical trials. As a statistical programmer, our involvement spans across the horizon of clinical trial conduct: from inputs to protocol creation, to finalization of end of text material in clinical study reports. Along with these, we are also associated with other activities like alignment of clinical studies within a project, resource planning & allocation etc.

To get a good hold on the activities and understanding the dependencies, graph theory comes to our rescue with data visualization. Some of the examples applied include, displaying dependencies between SDTM and ADaM, Resource utilization in programming SDTM datasets/ ADaM datasets/ TFL outputs, interconnections between SOPs etc. This paper attempts to address the requirements by introducing graph theory and finding solutions using Neo4j® applications.

INTRODUCTION

Graph database

A graph database stores data points as “Nodes” and connections between them as “Relationships”. It is in the same way as we may sketch ideas on a whiteboard, where as relational databases stores data in tables presented in columns (variables) and rows (records).

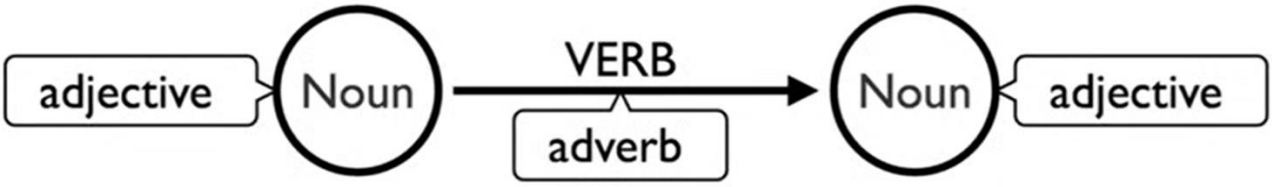
Nodes and Relationships

Nodes can be tagged with *labels*, representing different roles in the dataset. They can also have any number of key-value pairs as *properties*.

Relationships provide directed, typed or attributed connections between two node entities. Relationships always have a *direction*, a *type*, a start node and an end node. They can also have properties, just like nodes.

Nodes can have any number of relationships without compromising performance.

In english language terminology, above image can be shown as



What is Neo4j

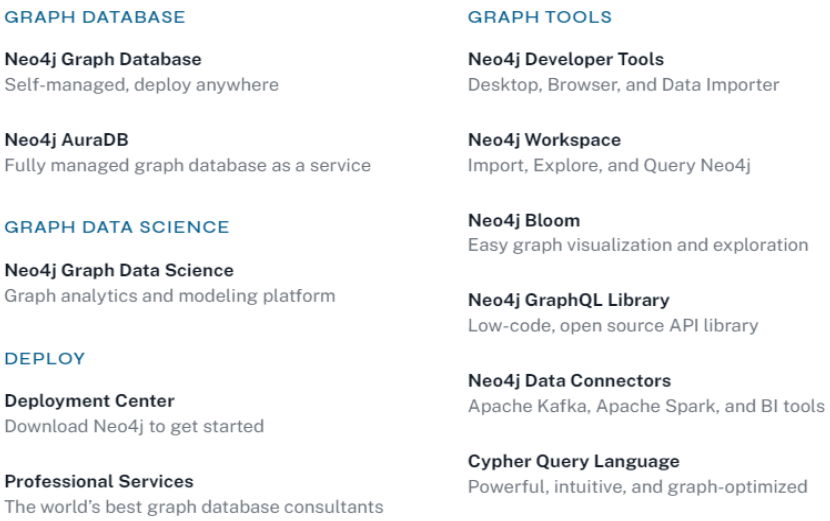
Neo4j founded by Emil Eifrem, is a powerful graph database that enables organizations to store, model and analyze complex relationships between discrete data entities. It is a native graph database, which means the underlying architecture of how the data is stored is not built on top of tables.

Neo4j works on Cypher query language (Basics of Cypher Query Language, is presented in **Appendix I**). Cypher is a pattern matching query language made for graphs, in other words - it is SQL for graphs.

Cypher is a declarative query language - where we express what we want and then it is database job to figure out what is the best query execution plan to do that.

Products

Here are different products available from Neo4j**.** In this paper, we use **Neo4j AuraDB** to visualise the data stored in a .csv file.



Neo4j AuraDB

Neo4j AuraDB is a fully managed graph database as a service (DBaas). It helps the user to find relationships in data with fast quering to obtain real-time analytics and data insights. It has both free and professional instances.

The free instance is designed for the user to rapidly learn, prototype and experiment without any cost. It includes one AuraDB graph database that scales up to 200k nodes and 400k relationships, with a memory of 1GB. It includes access to Neo4j Bloom, Browser and Data Importer and also a workspace for the user.

Location of AuraDB: As of date, AuraDB Free and Professional are available on Google Cloud Platform (GCP), while AuraDB Enterprise is available on Google Cloud Platform (GCP) and Amazon Web Services (AWS).

Appendix II walks through the data visualisation process using the free version (ie., Aura DB Free) step-by-step.

CASE STUDY

Purpose:

Visualise the flow of data and hidden dependencies with respect to resource contribution and project management.

Now, let us take a case study to see the output created by Neo4j Aura DB. We are using the below two .csv files (with hypothetical data) for the purpose. The contents of the datafiles are as follows:

Progplan.csv

A	B	C	D
Program	**Programmer**	**Reviewer**	**Status**
adsl.sas	Daniel	Kate	Final
adcm.sas	Robert	Daniel	Ready for review
admh.sas	Robert	Kate	Ready for review
adlb.sas	Kate	Robert	Ready for review
advs.sas	Daniel	Kate	In progress
adpp.sas	Robert	Kate	Ready for review
adqs.sas	Daniel	Robert	In progress

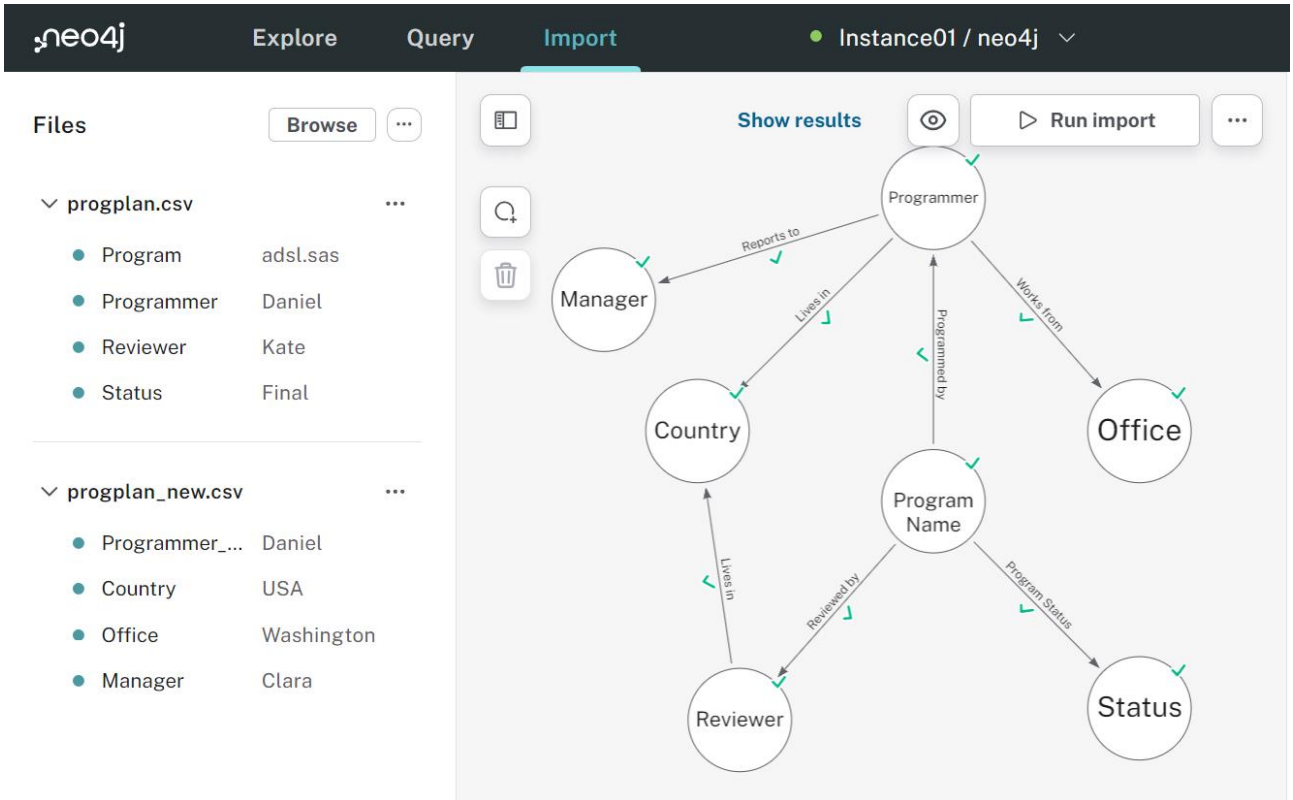
Progplan_new.csv

A	B	C	D
Programmer_new	**Country**	**Office**	**Manager**
Daniel	USA	Washington	Clara
Robert	UK	Heathrow	Bill
Kate	INDIA	Mumbai	Rahul

Methodology:

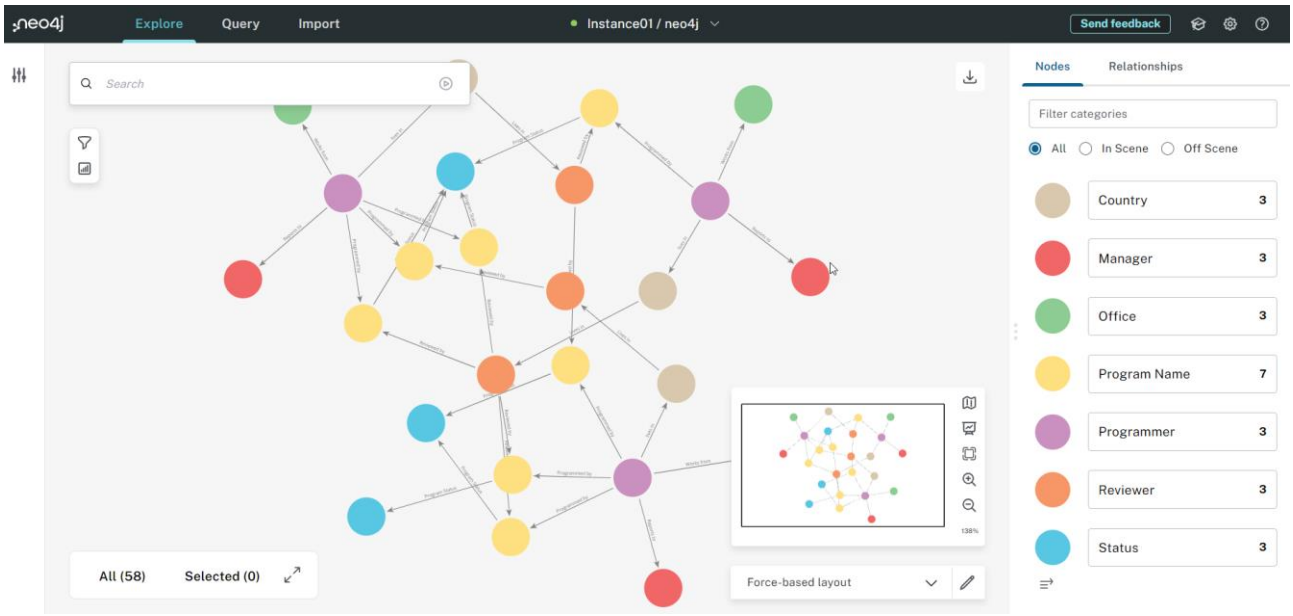
The data from the two .csv files were loaded into Neo4j Aura DB (as described in Appendix II), and a graph model of our desire was created :

Fig.1



After executing the import using "**Run import**" (located at the right top corner in fig.1), we get the data imported into the graph model (shown in fig.1).

After successful importing of the data into the desired graph model, the output looks as:



Now, using "**Query**" and "**Explore**" tabs (at the left top corner) faciliated us to traverse across the datapoints and understand data lineage for better data analysis and exploration. We also used Cypher Query Language (Syntax available in **Appendix I**) for filtering the data as required.

Outcome:

Above data visualisation helped us to address several important points related to resource management and project planning by:

1.) Analyzing the contributions made by each individual programmer interms of ADaM programs developed/reviewed.
2.) Identify the individuals who made the most significant contributions to the project.
3.) The programming status at any given point of time.
4.) Associated management team information and geographical location of the resource.

because of the above, we ensured that the workloads were distributed equally among the programmers, timelines were planned effectively, and updates were provided to management on an ongoing basis. These learnings paved way for planning the upcoming programming tasks and resource allocation in the new studies with data driven evidence.

Graph theory modelling can also be applied in other situations like

1.) Understanding of aCRF mappings to different SDTM domains.
2.) Interconnections between SDTM domains, ADaM datasets and Tables, Figures and Listings of the Clinical Study Report – which helps to address data *traceability* (one of the important principles of CDISC).
3.) Relationships between SOPs and Guidelines documents using their IDs/document numbering.
4.) Data understanding and overview in a clinical study (during conduct phase) with information from subjects, experienced adverse events, associated treament (with concomitant medication) and laboratory data for patient safety surveillance and monitoring.

and many more…

CONCLUSION

Every coin has 2 sides, and graph theory is no exception to it (with Pros and Cons)!.

Pros:

1.) Helps to traverse across data from point A to Z, for easy understanding of the intricacies and dependencies.
2.) The output to be created can be thought of like drawing on a white board, with thoughts in mind translated to presentation using relationships.
3.) It is possible to represent data in an intuitive and versatile way of visualizing data with enhanced query performance.

Cons:

1.) There is a learning curve – with concepts of graph theory, softwares and associated programming languages.
2.) Could be bit confusing for first time users, especially those used to work with relational databases.
3.) Constructing graph databases requires a different way of thinking and modelling data, with different query languages.

REFERENCES

https://neo4j.com/

https://neo4j.com/cloud/platform/aura-graph-database/faq/

Intro to Cypher: **https://www.youtube.com/watch?v=pMjwgKqMzi8**

Neo4j (Graph Database) Crash Course: **https://www.youtube.com/watch?v=8jNPelugC2s**

Neo4j Course for Beginners: **https://www.youtube.com/watch?v=_IgbB24scLI**

Training Series - Intro to Neo4j: **https://www.youtube.com/watch?v=otCRulNPUak**

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Madhusudhan Reddy Mule Venkata
Novo Nordisk A/S
Vandtaarnsvej 108, Soeborg, Denmark
Mobile: +45-30797695
mrmv@novonordisk.com

Kamal Chadha
Novo Nordisk
75 Hayden Avenue, Lexington MA 02421 USA
Mobile: +1-774-270-0793
kjca@novonordisk.com

APPENDIX I (Basics of Cypher Query Language)

#	Purpose	Syntax	Comment
1	Create a node, n	CREATE (n)	CREATE creates a node, n.
2	Create a node and return the node	CREATE (n) RETURN n	CREATE creates a node n, and RETURN returns the node information from the database.
3	Add a label while creating a node	CREATE (n:subjid)	Node n is assigned “subjid” as a label.
4	Assign multiple labels for a node	CREATE (s:label1:label2)	Creates a node s, and assigns label1 and label 2 as its labels.
5	Create a node with properties	CREATE (s: adsl {SUBJID:101001, COUNTRY: “Denmark”, REGION:”EU”})	Creates a node s, which has 3 properties assigned – Subjid, Country and Region.
6	Create multiple nodes and return them	CREATE (n), (m), (o) RETURN n,m,o	Creates and returns 3 nodes – n, m and o.
7	Check the existence of a node in the database	MATCH (n) RETURN n	MATCH matches/identifies the node n, and RETURN returns the node information from the database.
8	Check all the nodes with a specified label	MATCH (s:adsl) RETURN s	This returns all nodes with label = adsl.
9	Get a specific property of all nodes	MATCH (s:adsl) RETURN s.country	This returns “country” property present across all nodes with label = adsl
10	Get a specific property of nodes with two different labels	MATCH (s:subjid), (u:usubjid) RETURN s.country, u.country	This returns the properties (country) present on nodes with different labels (subjid and usubjid).
11	Get a specfic value of a property of all nodes across all its labels	MATCH (s:adsl {country=”Denmark”}) RETURN s	In SQL, this is equivalent to SELECT country FROM adsl WHERE country = “Denmark”
12	Add a property to an existing node	MATCH (s:adsl {country =”Denmark”}) SET s.siteid = 101 RETURN s	First identifies/matches the node, then uses SET clause to add a new property.
13	Remove a property from a node	MATCH (s:adsl {country =”Denmark”}) REMOVE s.siteid RETURN s	First identifies/matches the node, then REMOVE clause removes an assigned property.
14	Update a specific property as another property already exisiting for a node	MATCH (s:adsl {country = “Denmark”}) REMOVE s.siteid = 101 SET s.siteid1 = 101 RETURN s	This removes the property siteid with its associated value, and creates a new property siteid1 with 101 as its value.
15	Change the label of a node from one value to another, inheriting its properties	MATCH (s:adsl {country=”Denmark”}) REMOVE s:adsl SET s:adslsf RETURN s	This changes the label of node s, from adsl to adslsf.

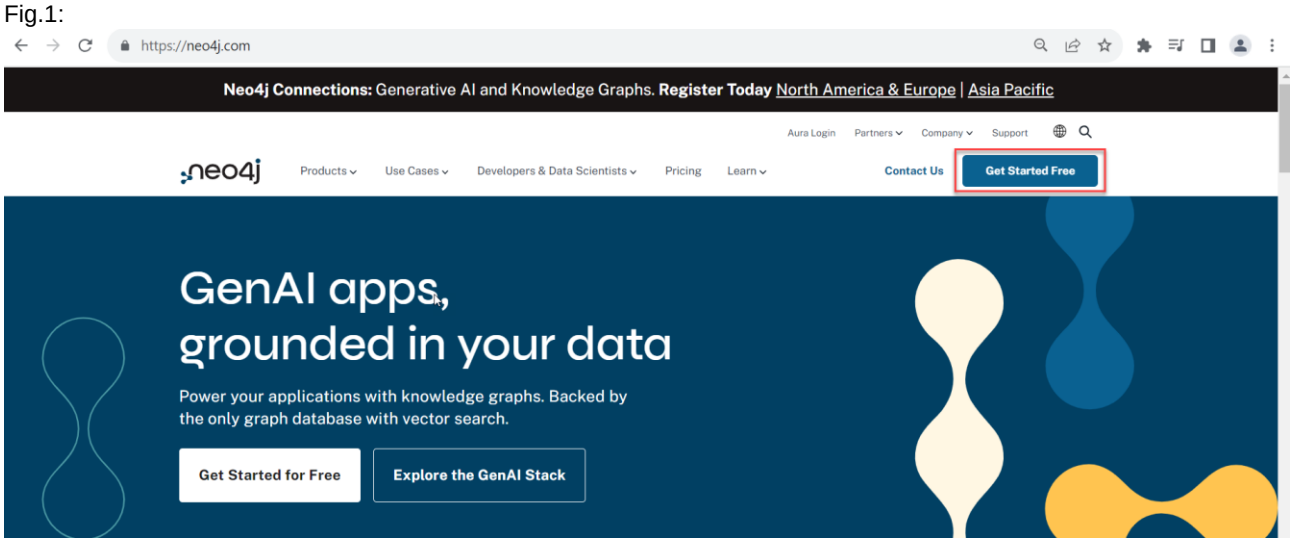
16	Create relationship between nodes	MATCH (s:adsl {subjid = 101001}), (a:adae {aeterm = “Nausea”}) CREATE (s) ➖ [e:experiences] ➡ (a) RETURN s,e,a	(node) ➖ [Relationship] ➡ (node) is the diagramatic presentation of creating relationship between nodes. ➡ Or ⬅ can be used to indicate the directions of the relationship.
17	Delete a specific node	MATCH (s:adsl {country=”Denmark”}) DELETE s	Delete clause is used to delete a node.
18	Delete all nodes from a database	MATCH (n) DELETE (n)	Deletes all nodes from the database.

APPENDIX II

Steps to download Neo4j Aura DB Free and Visualise data (located in a CSV file).

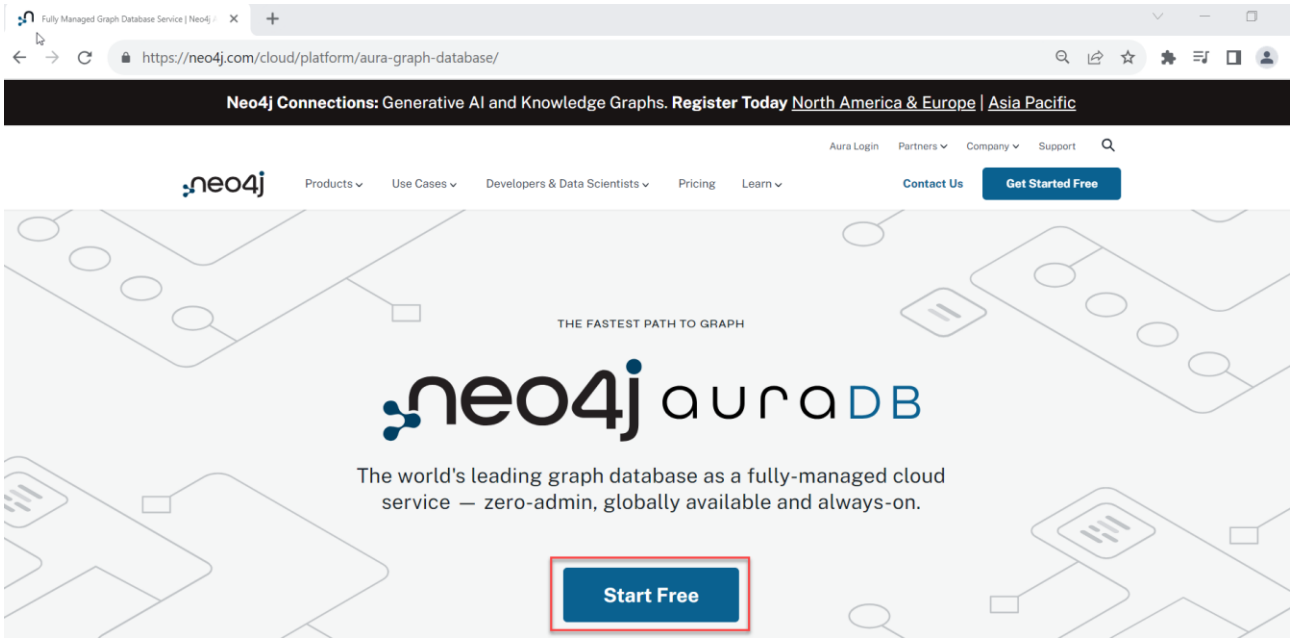
Step 1: Go to neo4j.com and click on "**Get Started Free**" (enclosed in a red box, in fig.1)

Fig.1:



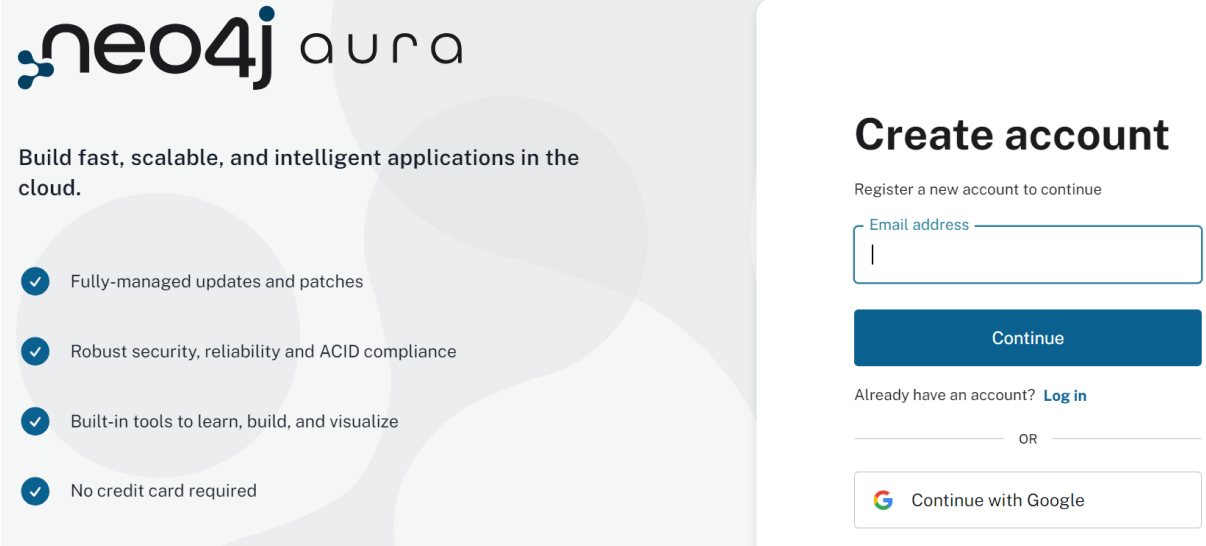
Step 2: This takes us to a new page of Neo4j aura DB, select "**Start Free**" (enclosed in a red box, in fig.2)

Fig.2:



Step 3: A new page opens, for signing up (as in fig.3):

Fig.3:



Create a profile with an email address and password or we can also continue with Google account, if we have one.

Once we create a profile, we will receive a verification link to registered mail id:

Click on the verification link to confirm authenticity.

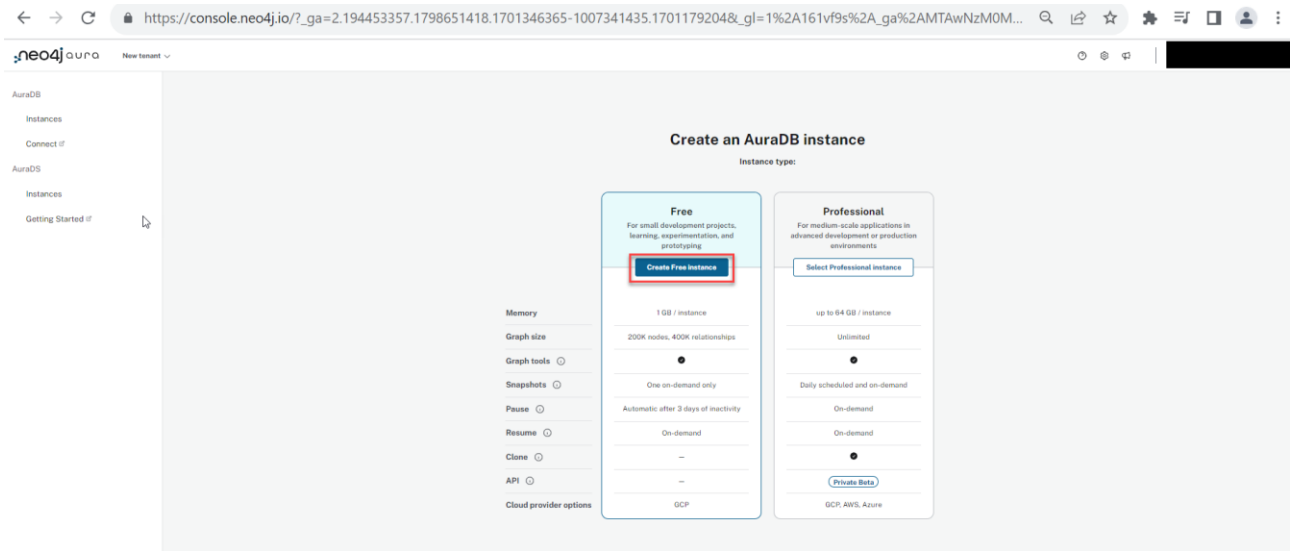
For subsequent log in's, we can proceed further by clicking on **Log in** of "Already have an account? Log in " as in fig.3.

Step 4: Select "**New Instance**", to move on further (enclosed in a red box, in fig.4)

Fig.4:

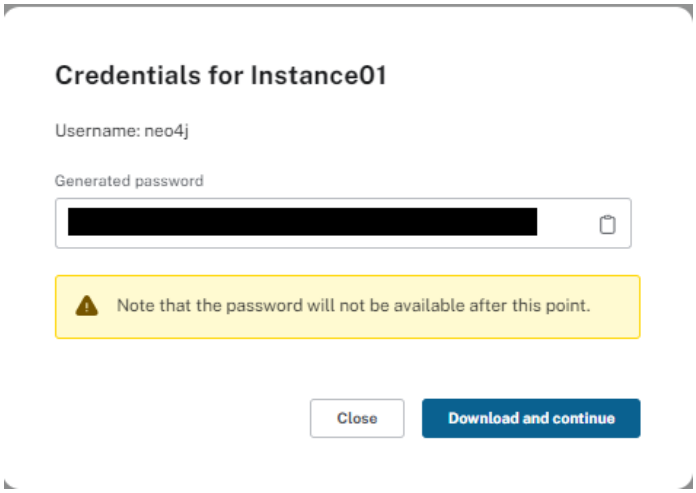
Step 5: Select “**Create Free Instance**” (enclosed in a red box, in fig.5)

Fig.5:



Step 6: Login credentials for the instance appears, with system generated password (masked with black strip in fig.6). Save the password in a secured location (this password is needed in further steps). Then, select “**Download and continue**”.

Fig.6:

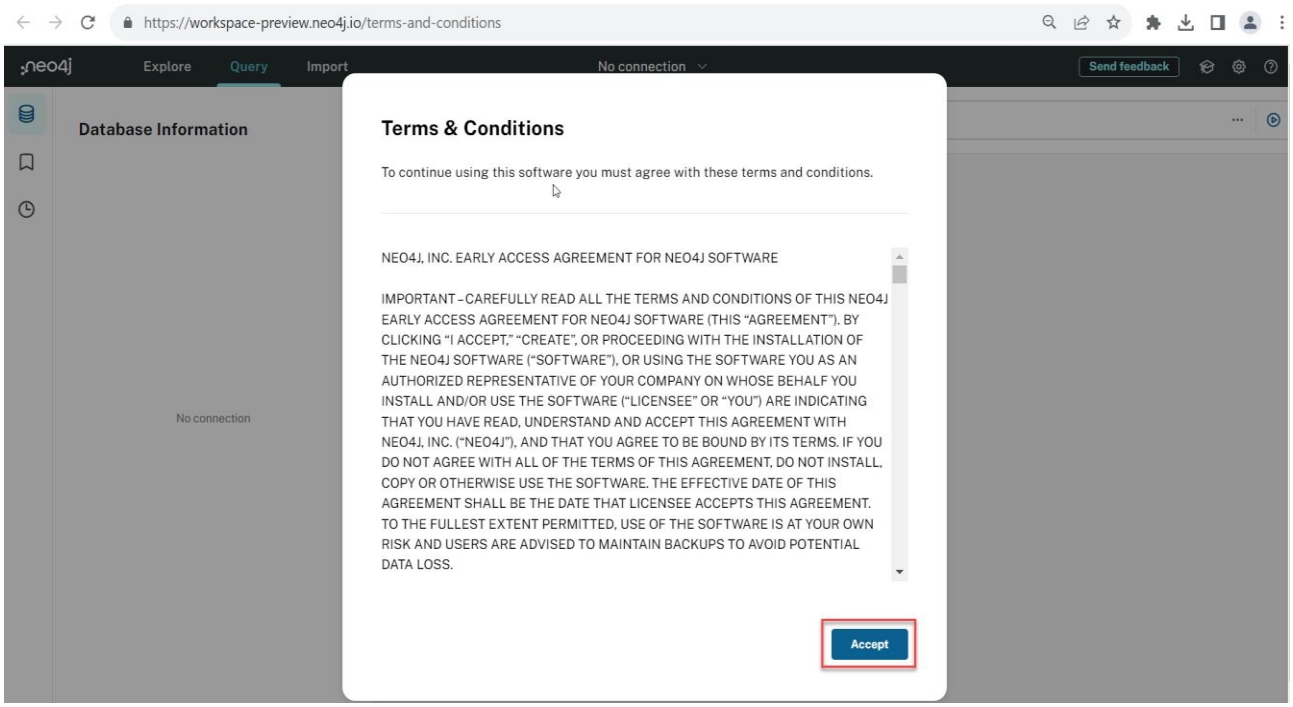


Step 7: A new working instance is created (takes few minutes). Now, click on “**Open**” (enclosed in a red box, in fig.7).

Fig.7:

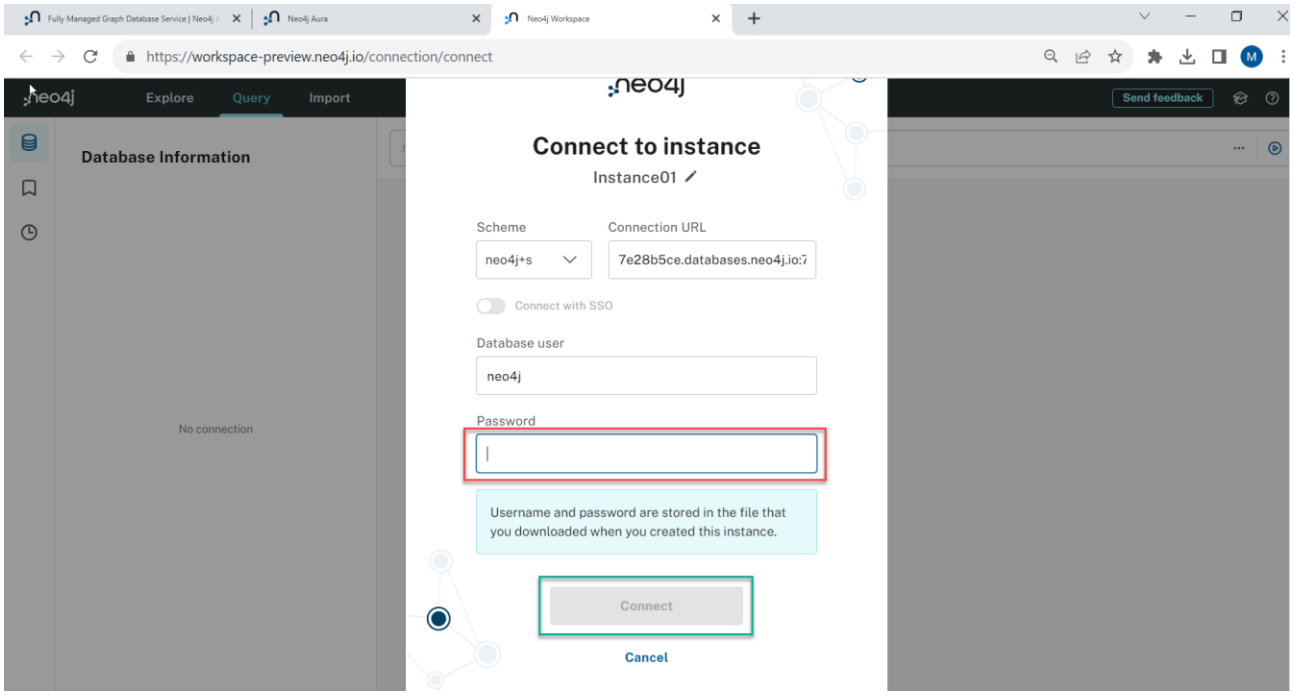
Step 8: Neo4j workspace opens as a new window, “**Accept**” Terms & Conditions, to move on further (enclosed in a red box, in fig.8).

Fig.8:



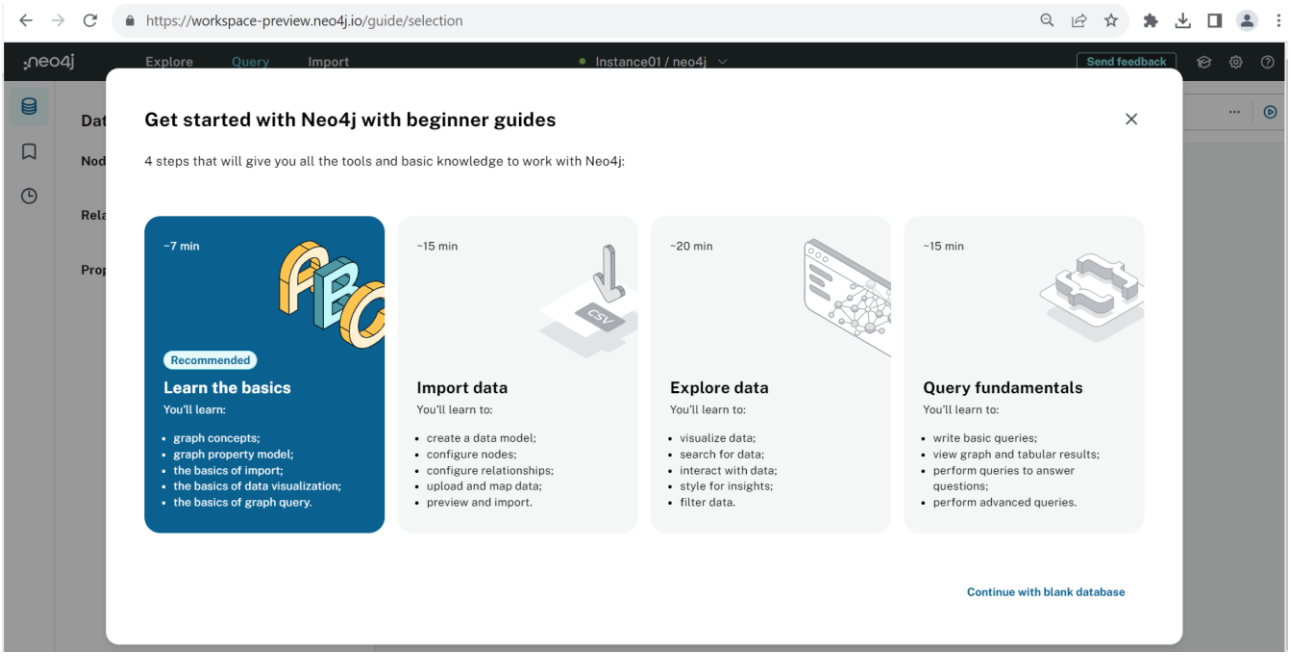
Step 9: Once accepted, the following page opens – Copy the password stored in a secured location (done in step 6) , and press **Connect** (enclosed in a green box, in fig.9).

Fig.9:



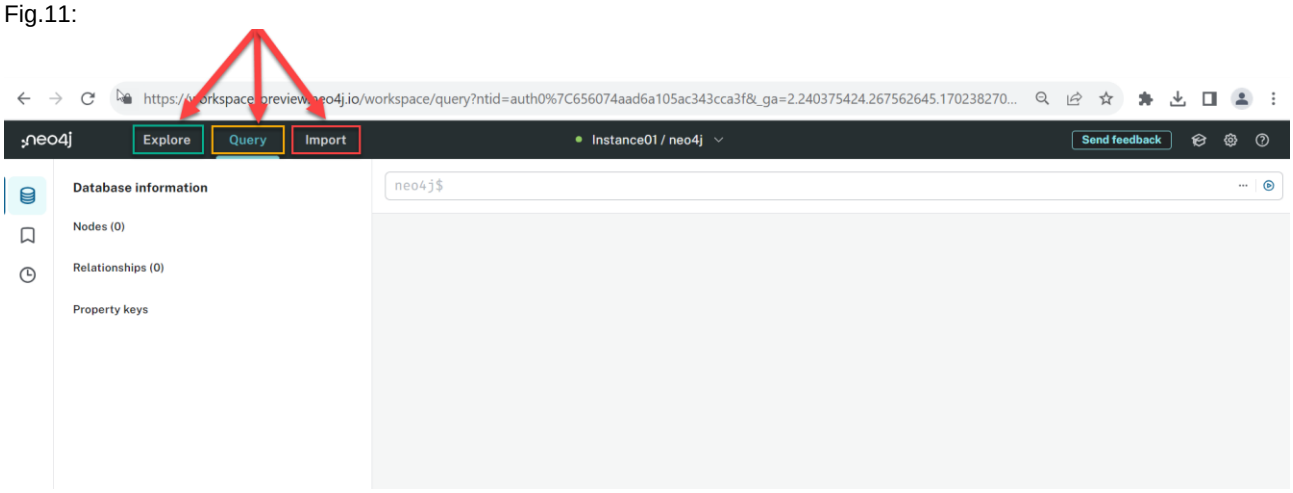
Step 10: As in fig.10, A beginners guide appears to “Learn the basics”, how to “Import data”, “Explore data” and “Query fundamentals”. It is highly recommendatable to go through the course material, as a first time user.

Fig.10:



Step 11: Once the connection is made, Neo4j work space appears as in fig.11

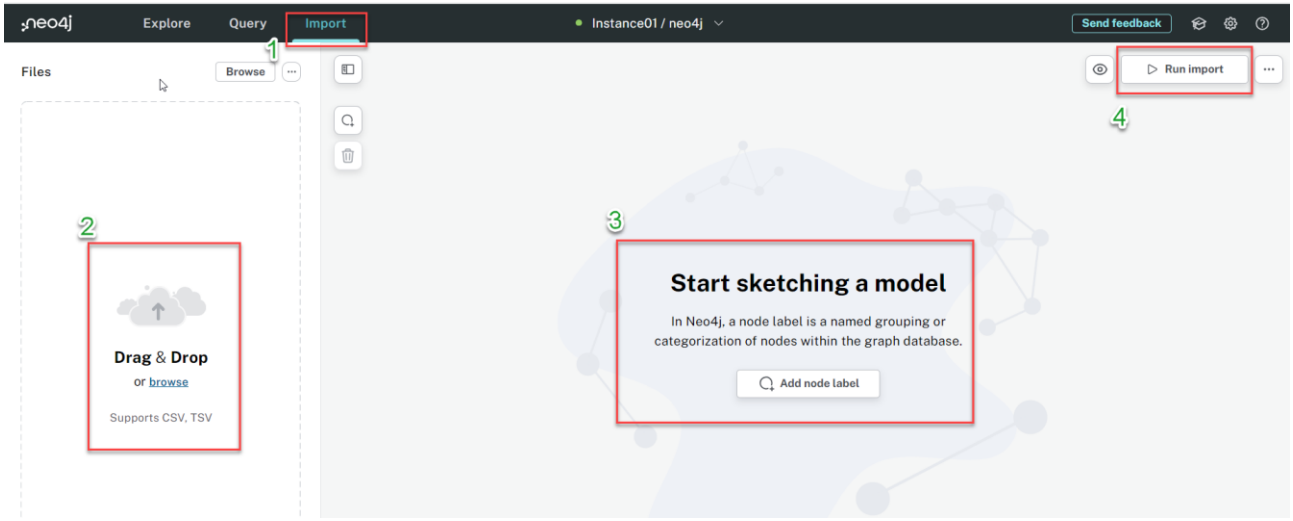
Fig.11:



The three tabs at the left top corner (**Explore**, **Query** and **Import**) are of high importance to work with visualisation. As the name suggests, **Import** tab helps to import a csv file, **Query** tab helps to filter/query the data imported, using cypher query language and finally **Explore** tab helps to explore the results after importing and/or quering.

Step 12: The **Import** page appears as in fig.12.

Fig.12:



1: Shows the location of the page you are on!.

2: Drag & Drop or Browse to the location of the data file.

3: The canvas here, helps to sketch a graphical model of our interest, using nodes and relationships to visualise the data, and finally,

4: Run the data import (of the loaded datafile) into the model we described.