1. Introduction

The FIU Case Network database is setup on Neo4j Enterprise, Graph Database and is visualized through the Linkurious Graph visualization tool. This Document includes the Solution Design Specifics using Neo4j and Linkurious.

The Document is divided in to four main sections and these are

1. Case Graph/Network Model Imperatives, - This section mentions the design principles adopted to build the FIU Network Model and showcases a representative model. The final model shall be prepared once all the data elements for the FIU Case Network is completely incorporated; however, care has been taken to classify and categorize data elements semantically and logically so that they can be easily assimilated into the global design imperatives at any time and through the entire lifecycle of the project.
2. Case Graph/Network Visualization Imperatives: - This section covers in fair detail the wide spectrum of visualization artifacts that will be part of the final solution. Care has been taken that the basic design principles of network visualization are adhered across all the visualization artifacts. These include the following but are not limited to just these.
    a. Depending on the area of display the visualization will ensure that the context of case/entity or transaction will be given topmost priority while displaying the network/graph.
    b. To ensure clarity and always avoid clutter, the content on display will be limited enough for a single eye-span to absorb and ingest.
    c. Hover-over and click for additional information for key sections of display will be always available.
    d. To distinguish specific weights and or areas of interest, specific color coding or sizing parameters shall be used.
3. Advanced Network Analytics – This Section will cover the advanced features that are additionally available with Neo4j and Linkurious Enterprise, that can be effectively utilized by FIU to get better insights into the case/entity network. This section will showcase a few examples that may have direct relevance to FIU and will provide notes on how these could be easily built into the existing model as the model is adopted on the lines of providing to evolve and grow to cater to demands for more advanced set of Network algorithms. The section will include a set of pre-built and out of the box graph algorithms for use.
4. Case Risk Scoring Model. Though it is at a preliminary stage, we have built a basic risk scoring model that can be used with a case for better decision making. The current model includes the information that we currently have and will be reviewed and analyzed with real cases for validation.
5. Network Graph Visualization Integration with FIU FINNET 2.0 main system including the Case Inspector Tool.

2. The Graph Model

Graph data modeling is a collaborative effort where the application domain is analyzed by stakeholders and developers to come up with the optimal model for use with Neo4j. The domain along with questions about how the business at hand operates. Applications will mainly retrieve the data for business use cases by traversing the graph. Traversal means anchoring a query based upon a property value, then traversing the graph to satisfy the query.

To design the FIU model We has followed a multi-step process with the first step being understanding the domain. Based on the Physical and Logical data models of FINNET and FINGATE databases, including the FINCORE subsystem, a preliminary Network Model that includes Entity, Attributes, Case, and Risk Scoring parts is shown below. Figure -1 represents the Preliminary Network Data Model.

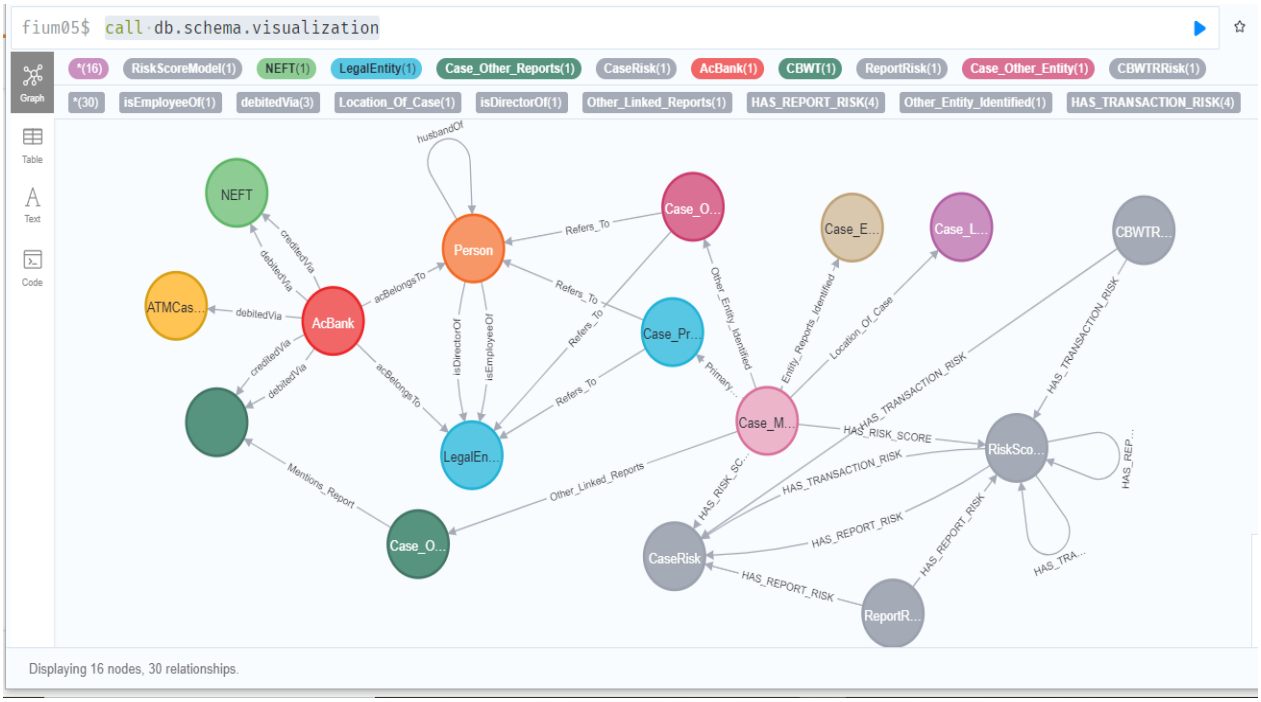

Figure-1 – Representative - FIU-NETWORK-DATA-MODEL

For nodes, we have made sure that the model address uniqueness of nodes and separates complex data node properties as nodes connected to its parent.

An extreme implementation of **fanout** is shown below. For modeling complex data, the previous example with all properties in a node and this example where each property is in its own node are usually suboptimal.

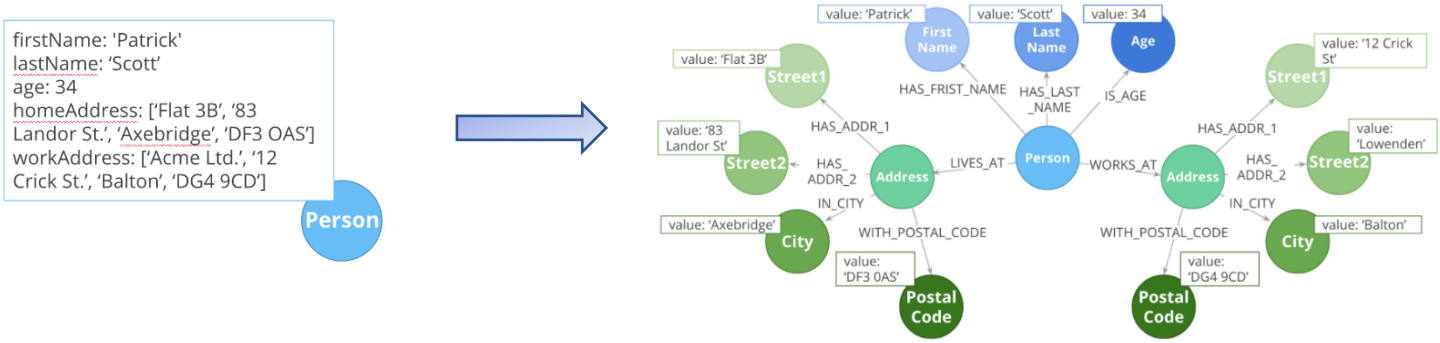

Figure-2 – Fanout of Complex node

In general, fanout is used to do one of two things.

Reduce duplication of properties. Instead of having a repeated property on every node, all of those nodes are connected to a parent node with that property. This can make data updates massively easier.

Reduce gather-and-inspect behavior during a traversal. If suppose all address in a particular city need to be checked, instead of checking every person's properties and then filtering out relevant nodes, we can traverse to the node of the city in question and check all the address nodes connected to it.

In the figure below we see a screenshot from the actual FIU representative model where a complex node (Address) is fanned out into multiple address meta nodes available from FINNet. Each address is mapped to a pin code which is mapped to a city which is mapped to a district and so on.

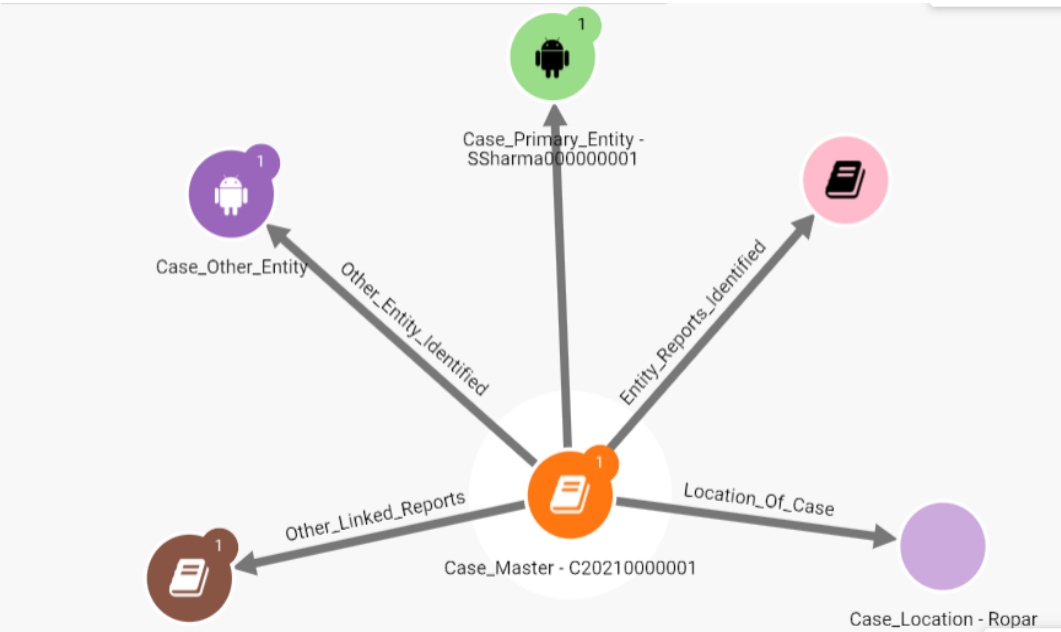

Figure-3 – Fanout of Complex node Case_Master

Author: Shrey Iyengar | Resident Director | Slink Information Systems

For relationships, Softlink has ensured that the model uses specific relationship types , reduces symmetric relationships and using types vs properties.

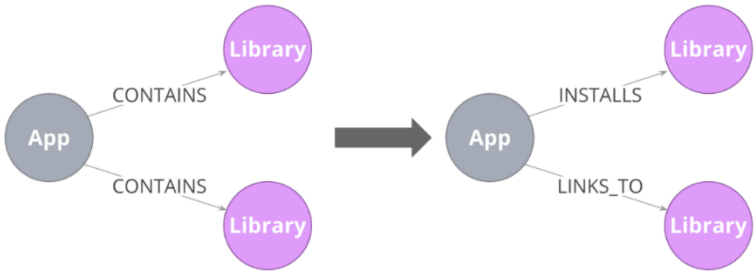

Figure-4 – General Vs Specefic Relationships

In general, using specific relationship types help to reduce gather-and-inspect behavior. In this case, if supposed the usecase requires the knowledge of what libraries will be INSTALLED by an app, the specific types on the right saves some wasted traversal.

In the FIU graph model, we have created specific relationships wherever possible. In the following figure instead of having one relationship type called CONTAINS the case master node uses the fanout technique having Specific relationship types linking it to its children case entities.

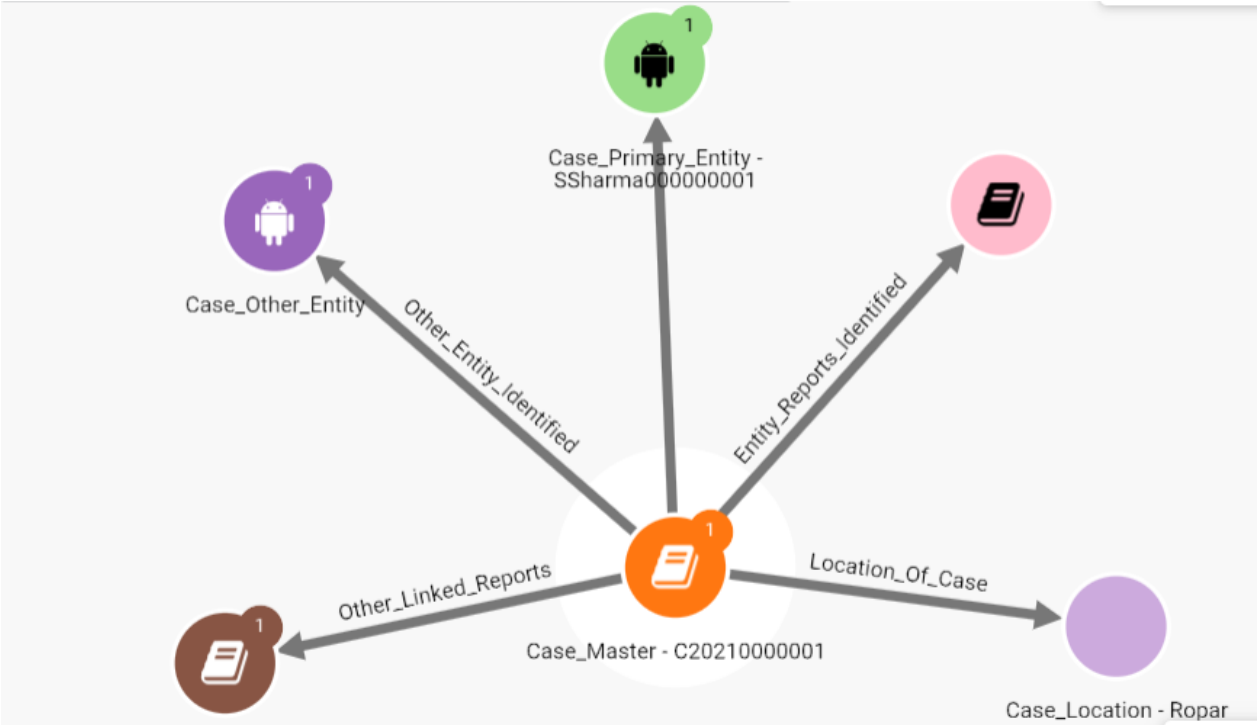

Figure-5 –Specefic Relationships in the FIU model

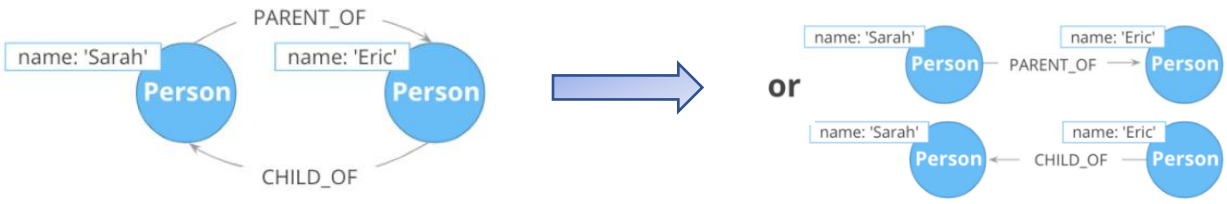

Figure-6 – Semantically Symmetric Relationships

Semantically symmetric relationships present two problems.

First, they are a form of needless data duplication. PARENT_OF and CHILD_OF mean exactly the same thing. There is no way to have one be true and the other not.

Second, they enable the user to violate the **Cypher expectation of relationship uniqueness**. Semantically, you have two identical relationships—they just look different technically. This will allow to traversal of the same relationship twice which is undesirable and contributes to wasted traversal.

For the FIU graph model , the relationships were modeled keeping these core principles in mind. For example only the husbandOf relationship has been created instead of both husbandOf and WifeOf or a generic SpouseOf relationship.

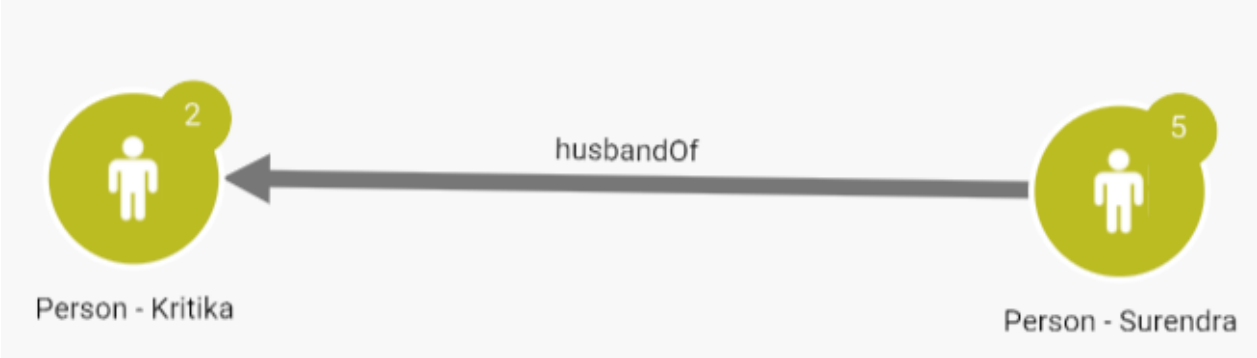

Figure-7 – Semantically Symmetric Relationships are not present in the FIU graph model

Data Accessibility and general refactoring principles

Anchor node data, which could be a node label or a node property value, is cheapest to access—it involves no traversal at all, and thus no wasted traversal. Indexed anchor properties are much more accessible than non-indexed anchor properties

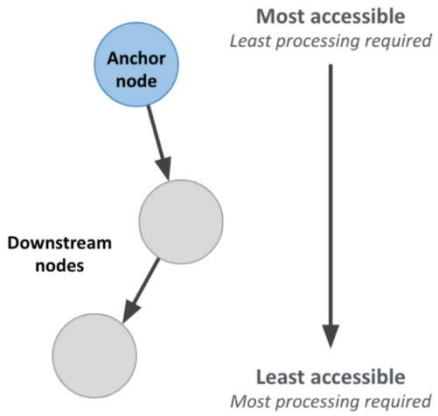

Figure-8 – Most Accessible to Least Accessible data points

Neo4j is designed for easy traversal, so relationship types are also very cheap to access. In fact, we recommend using types in Cypher queries always, because the cost of the type lookup is minimal. And if you do not use a type name in your traversal path, you will likely use something even less accessible, like downstream properties.

Downstream information such as labels and properties further down the traversal path are the most expensive thing to access. The further downstream, the more expensive they are. That is not to say they are necessarily deal-breakers but elevating downstream data via a model change is one of the most reliable ways to improve query performance. For example, reduce how far downstream a query traversal must go to get the necessary data, or change the model so that the downstream data is available in a relationship type instead of a node label or property. However, query performance is not the only metric that matters! Query simplicity, write/update speed, and the human-intuitiveness of a model are also important factors. When considering whether to elevate some data along this hierarchy, speed must be weighed against the impact this will have on those other factors.

Keeping this in mind, based on our understanding of the relevant use case for network visualization from the case inspector tool, we have ensured we have the right anchor nodes accessed for the relevant visualization. An example is shared below.

Case Visualization – Here the **Case Master Id** is the one that will have to be the anchor node and hence the view below, all the downstream nodes and relationships from the anchor node can be fetched almost immediately. This method can also be used to fetch all other cases for a specific primary entity, since the search would filter all cases which has that primary entity, instead of searching for cases from the Entity ID.

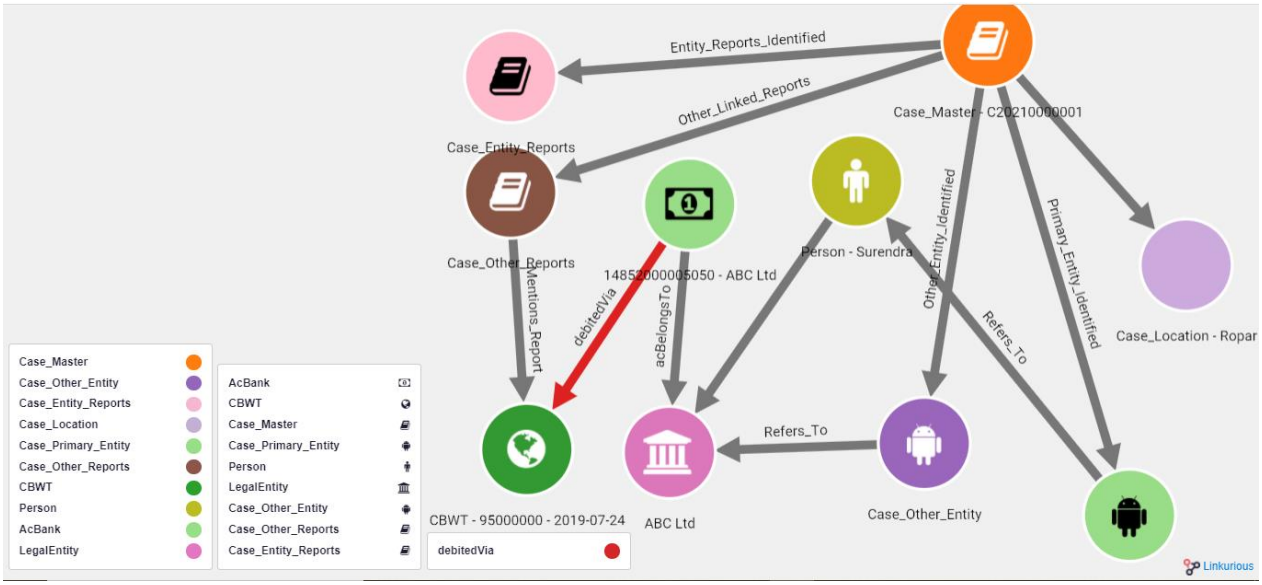

Figure-9 – Case Master Id will be the anchor node for subsequent case queries

3. Case Graph/Network Visualization Imperatives

For FIU analysts the Network Visualization is available from the Case Inspector Tool when they hit the Network tab from the menu.

The analyst will then be presented with the following sequence of visualizations, and She/He can move back and forth between these visualizations to get a more insights and or understand the case better.

The First Screen will be the Case View that is being referred to in the Case Inspector Tool, and it would appear shown below in Figure-10 – Case View.

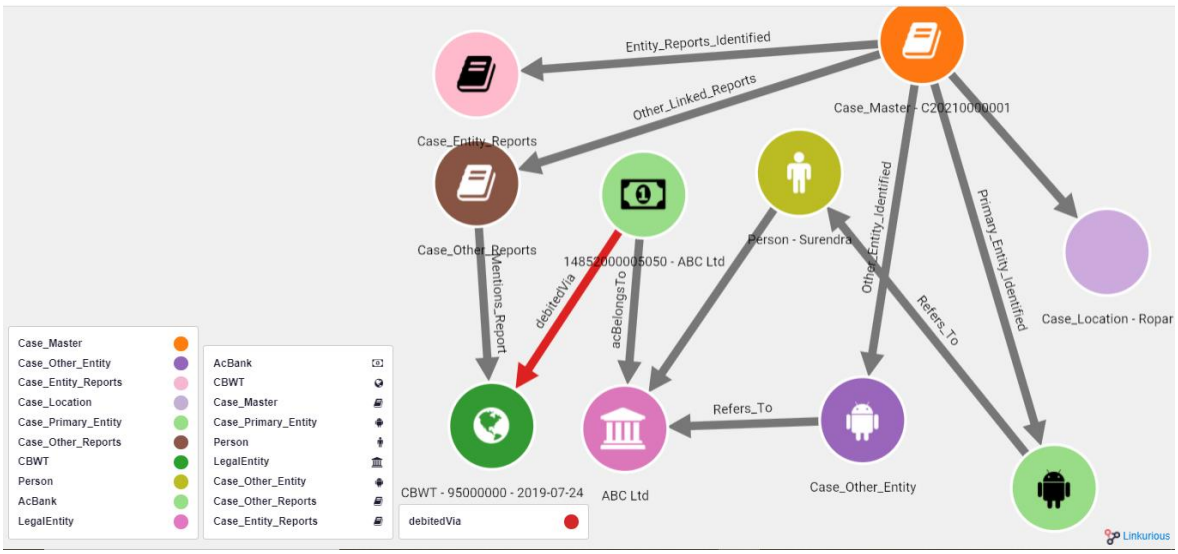

Figure-10 – Case View

The Case will anchor on the Case with the specific CaseMaster Id. What it would show include the first level linkages as built/created by the system based on the information compiled from the RE report and subsequent Case creation process.The legend above represents what will be shown here and how they are represented. When the Analyst hover over an entity or relationship and clicks it, details of all the properties associated with the entity or relatiohship will be displayed in a small scrollable window next to it.

The figure below shows the GOS description below for specific STR that includes the GOS information.

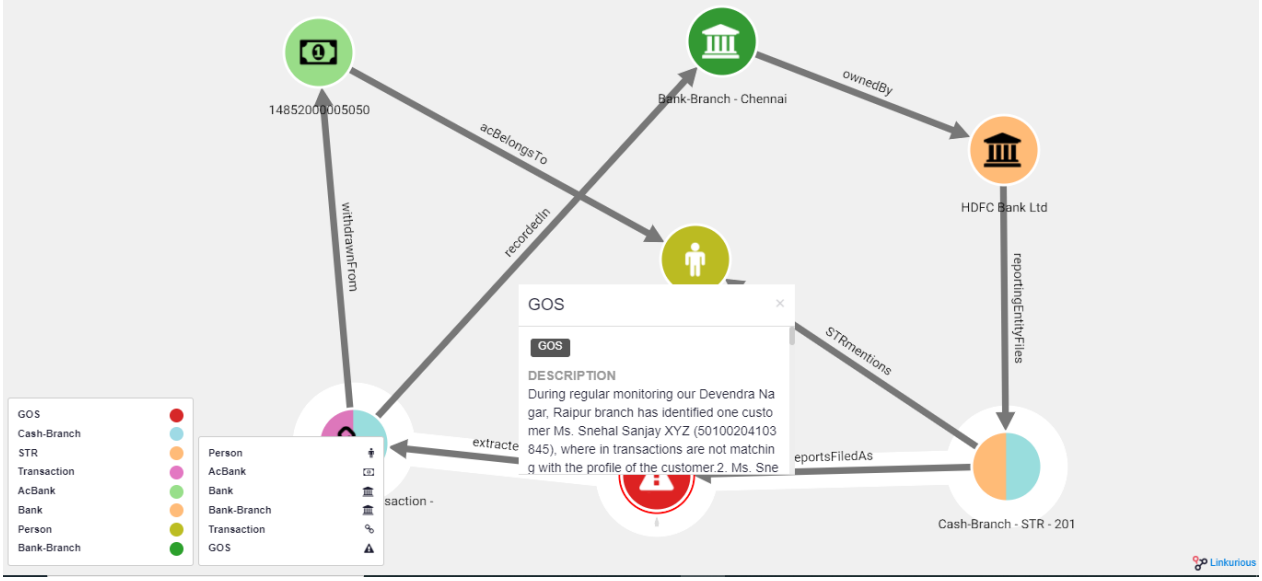

Figure-11 – GOS Description for Specific STRs

From the Case View Screen, the Analyst can chose to view the Entities and their key attributes, here is a specific entity view. The Primary Entity Surendra Sharma's entity view could look like Figure-12 – Primary Entity View.

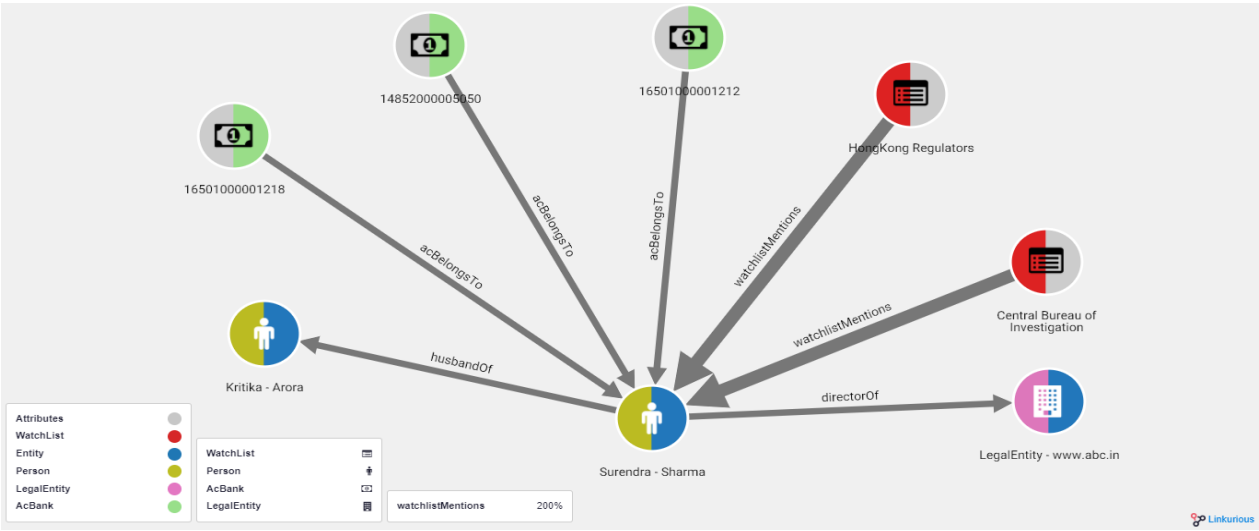

Figure-12 – Primary Entity View

Author: Shrey Iyengar | Resident Director | Slink Information Systems

From Here the Analyst could dive further into the transactions so far between the Primary and other entity of the case over multiple periods, here are couple of views, the first is all transactions for the period view, including the CBWT transaction for which the case was filed.

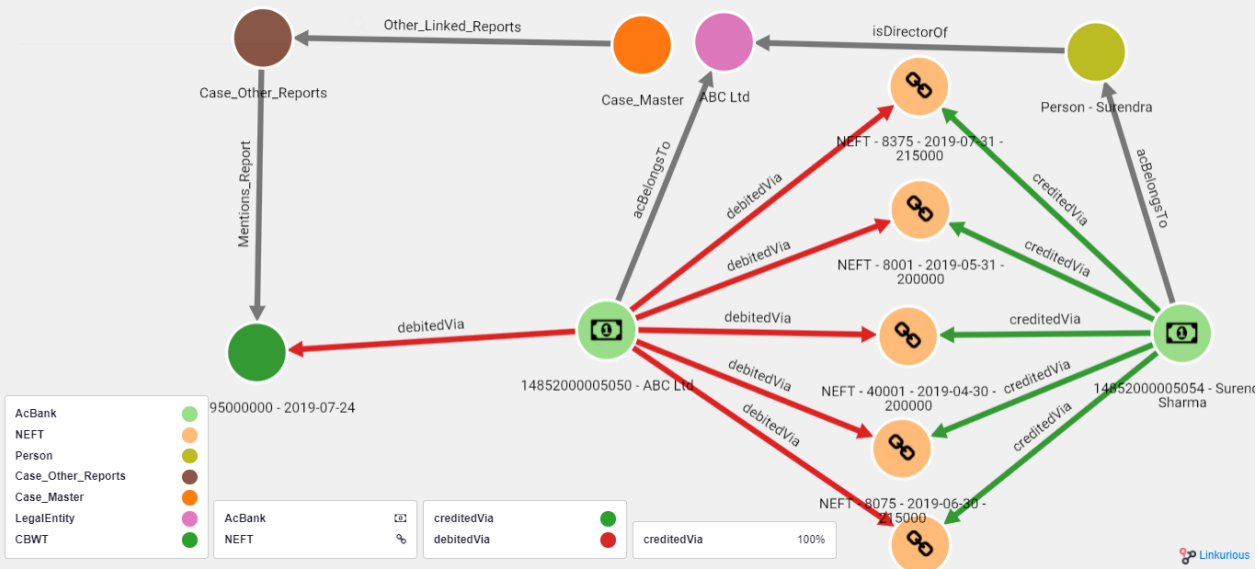

Figure-13 – Transaction for a period-View

If the Analyst decides to view transactions over a timeline here are some snapshopts of that view below.

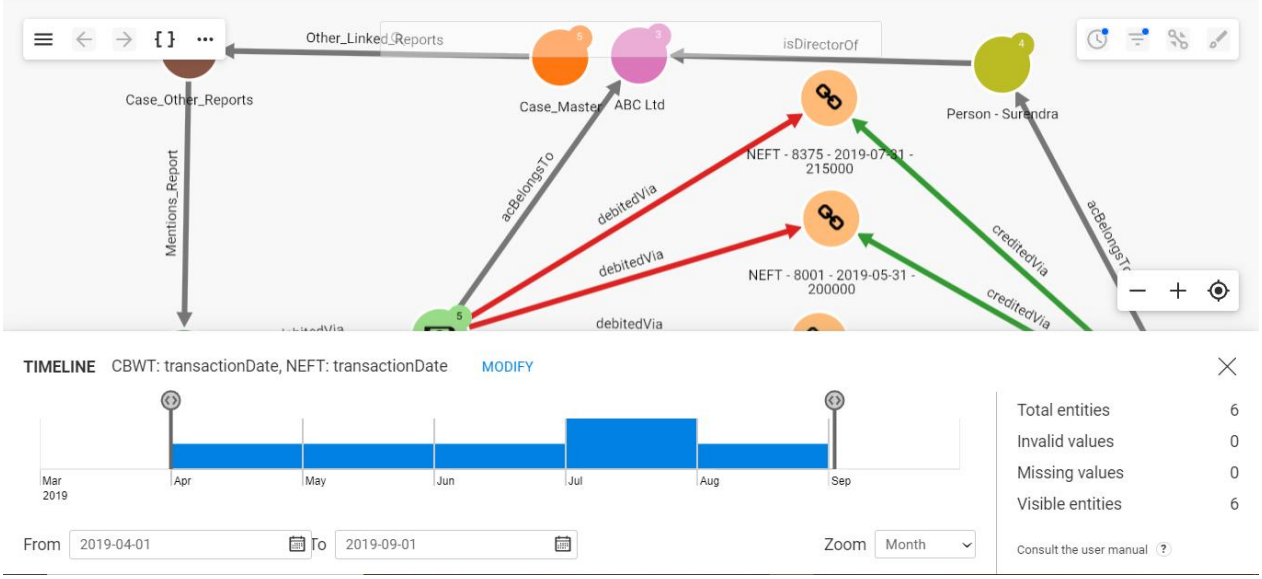

Figure-14 – Transaction for a period-View(Timeline)

Author: Shrey Iyengar | Resident Director | Slink Information Systems

The above view is for the complete period, instead the analyst chooses the transaction timeline around the period the CBWT was conducted, The visualization would look like this.

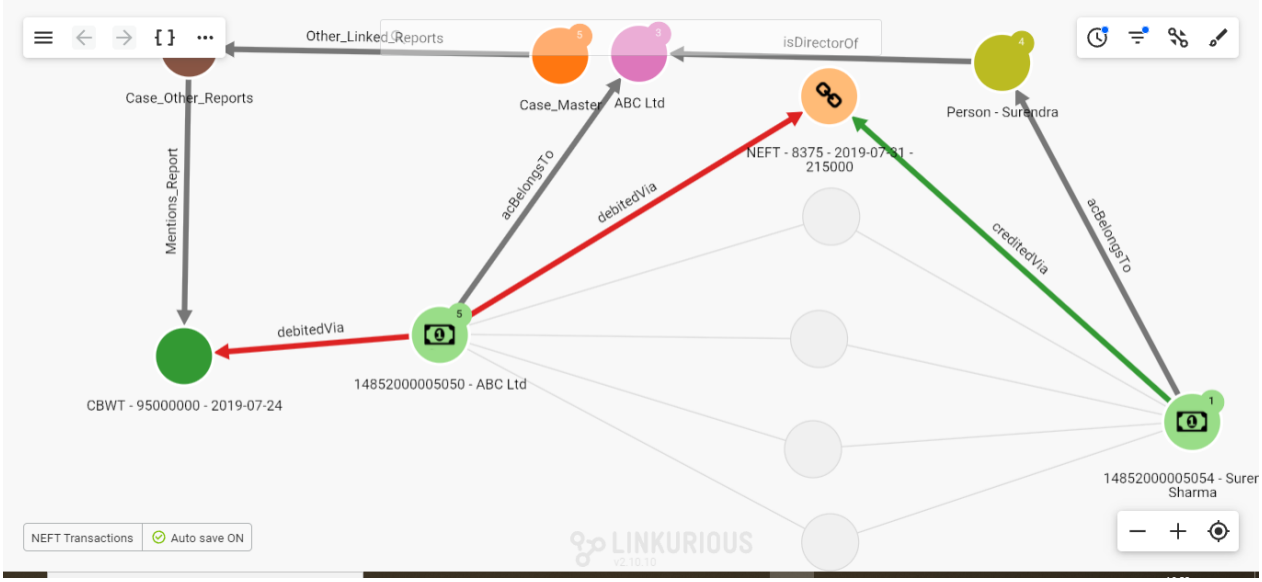

Figure-15 – Transaction for a period-View(Specific Time Period)

4. Case Risk Scoring Model

Though it is at a preliminary stage, we have built a basic risk scoring model that can be used with a case for better decision making. The current model includes the information that we currently have and will be reviewed and analyzed with real cases for validation.

Based on the Case Network View, which includes the relevant entities, their relationships, Transactions, and reports, including the amounts and volume of transactions, we have built a Case Risk Model which considers the above elements and scoring plan as per the information shared by the functional team. Here are the basic parts of the Case Risk Model.

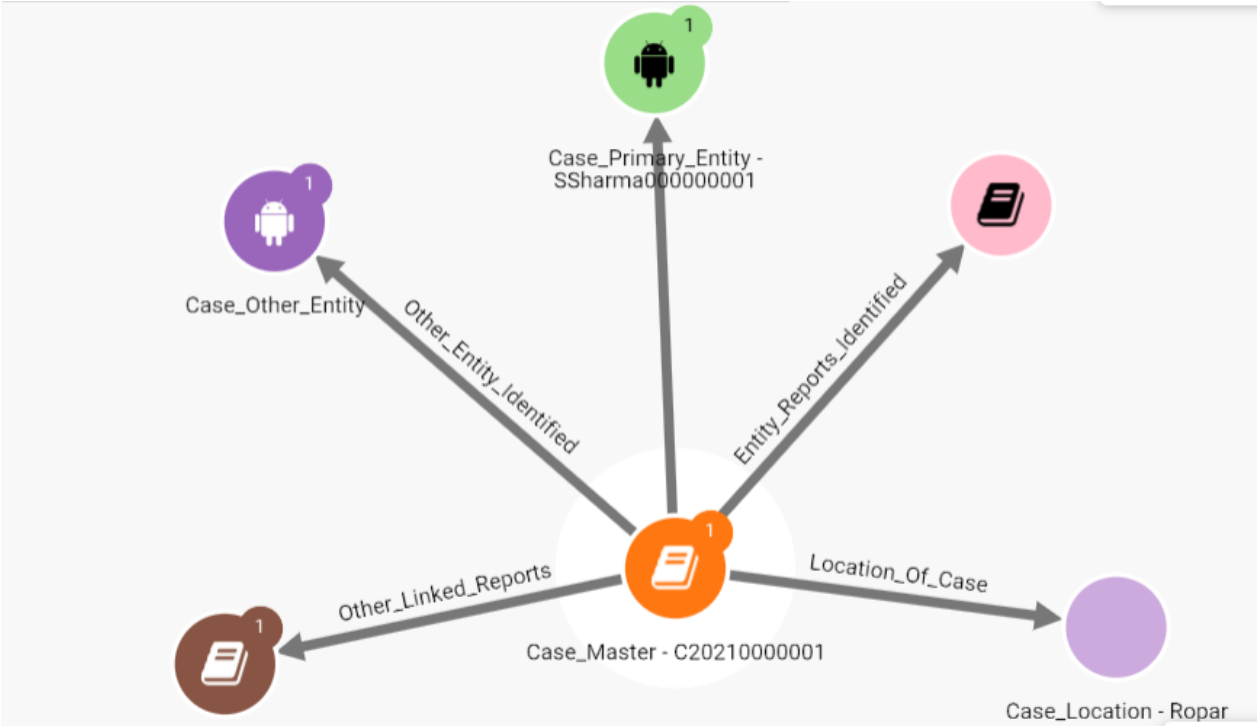

Figure-16 – Case_Id basic contents

The contents of the case basically are referred to while modelling the Risk associated first at each level of the case content, like Primary entity, Transactions, Reports which then contribute to the overall Network Risk, and finally onto Case Risk. The tables below represent the compute part of the Case Risk.

Computation formula
Tranx Risk= Avg(Tranx Type Risk, Vol Risk+ Value Risk, Location Risk, Freq Risk)
Report Risk=Average[Tranx Risk, Inherent Risk (Entity or Individual),, GoS Risk(for STR)]
Case Risk=Average(Report Risk, Network Risk)
Network ristk= AVG(report risk + Totalentitiyrisk)

Scenario 1: Salaried customer_CTR (Cash at Branch)

Date	Mode	Value/Amount of tranx Dr. (in Rs.)	Volume (No. of tranx)	Domestic Location	Risk Score of Domestic location
01-03-2021	Cash	₹ 3,00,000.00	3	Srinagar	10
08-03-2021	Cash	₹ 2,00,000.00	2	Amritsar	9
15-03-2021	Cash	₹ 3,00,000.00	2	Jammu	9
20-03-2021	Cash	₹ 3,00,000.00	4	Delhi	10
Monthly summation		₹ 11,00,000.00	11		38
Risk Score		10	7		9.5

Scenario 2: LLP_CTR (Cash at Branch)

Date	Mode	Value/Amount of tranx Cr. (in Rs.)	Volume (No. of tranx)	Domestic Location	Risk Score of Domestic location
02-03-2021	Cash	₹ 3,00,000.00	3	Chandigarh	6
09-03-2021	Cash	₹ 2,00,000.00	2	Chandigarh	6
16-03-2021	Cash	₹ 3,00,000.00	2	Chandigarh	6
21-03-2021	Cash	₹ 2,00,000.00	4	Chandigarh	6
Monthly summation		##########	11		24
Risk Score		7	6		6

Peer Group	Tranx Type Risk	Value Risk	Vol Risk	Location Risk	Freq of Tranx	*Tranx Risk*	Inherent Risk (Indicative)
Salaried	7	10	7	9.5	10	8.7	9
LLP	7	7	6	6	10	7.2	6

Figure-17 – Risk Score Computation Tables

The Network Risk Model view for the Case Network View will thus look like Figure-18.

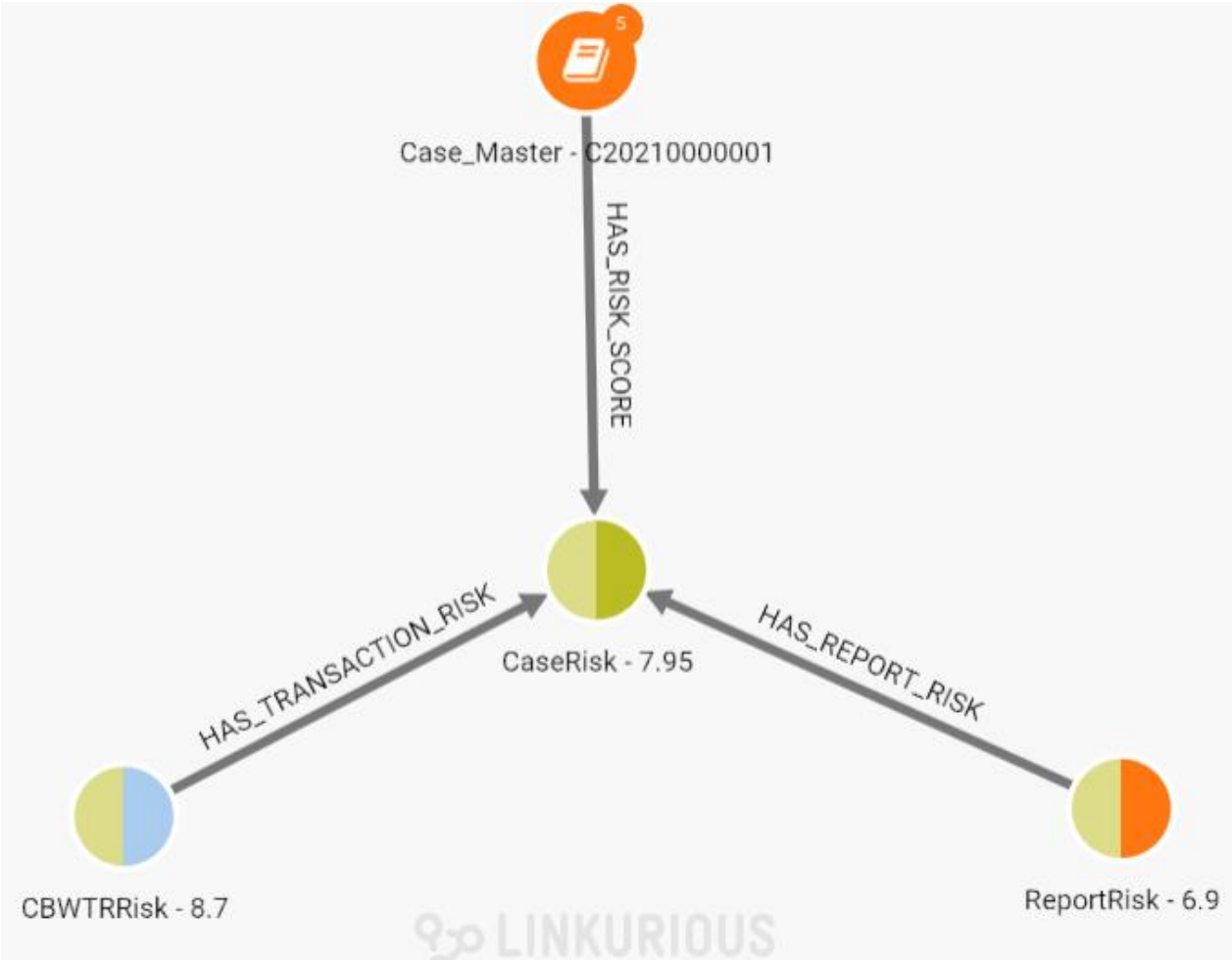

Figure-18 – Risk Model Network View

The Contents of the Risk Model include the

CASE RISK – The Node that stores the data, that helps us compute the overall Case Risk. We have captured the following information, based on the tables above.

"NetworkRisk": 9,

"ReportRisk": 6.9,

"CaseRisk ": 7.95

TRANSACTION RISK – Since we only have CBWT for this case, the transaction risk is computed at each transaction type node for its associated Risk. We have captured the following information, based on the tables above.

"TotalNoOfTransaction": 1,

"FrequencyOfTransaction": 1,

"TransactionTypeRisk": 7,

"LocationRisk": 7,

"TransactionRisk": 8.7,

"FrequencyRisk": 10,

"TotalAmount": 95000000,

"ValueRisk": 10,

"VolumeRisk": 1,

"Location": "Ropar"

REPORT RISK – The Node that stores the data, that helps us compute the overall Report Risk. We have captured the following information, based on the tables above.

"GOSRisk": 7,

"TransactionRisk": 8.7,

"ReportRisk": 6.9,

"EntityInherentRisk": 5

The Risk Score Model View imposed on the Case Network View would look something like this.

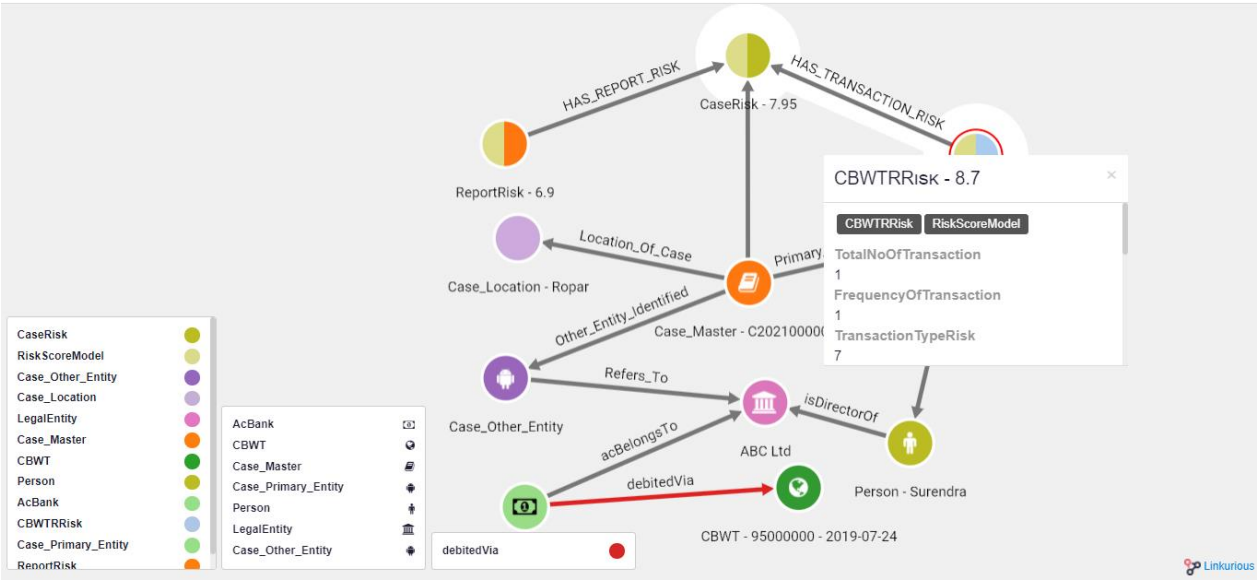

Figure-19 – Case View with Superimposed Risk Score Model

5. Advanced Analytics

Graph analytics, also known as network analysis, focuses on the study of relationships in real-world networks. Due to the sheer volume of data, the insights are sometimes hard to spot with the naked eye. Graph algorithms are so exciting because they are the methods, we use to understand real-world networks. They help us uncover the essence of complex systems by analyzing their connections. Local graph patterns can be observed and retrieved with query languages such as Cypher. The term graph algorithms usually refer to more global and iterative analysis, where, for example, to learn how influential the neighbors of a particular node are, given their position in the global network, graph algorithms would be used.

Graph algorithms are used to compute metrics for graphs, nodes, or relationships. They can provide insights on relevant entities (centralities, ranking) in the graph or inherent structures such as communities (community-detection, graph-partitioning, clustering).

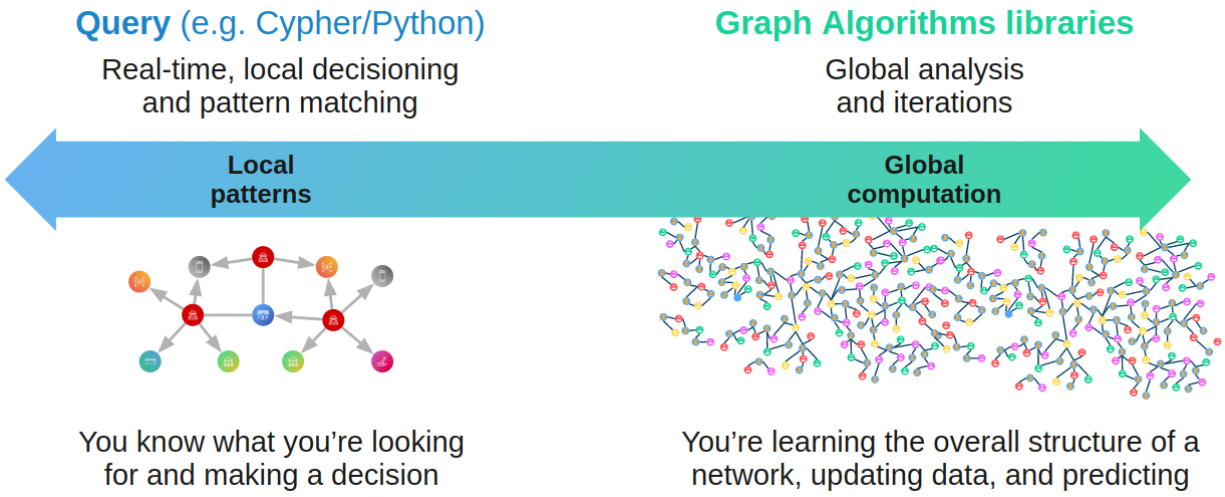

Figure-20 – Local vs Global Graph Analytics

Neo4j for Graph Data Science is a toolbox that combines a native graph analytics workspace and graph database with scalable graph algorithms and graph visualization for a reliable, easy-to-use experience. This framework enables data scientists to confidently operationalize better analytics and machine learning models that infer behavior based on connected data and network structures. As the graph evolves over time through GDS higher levels of insight can be retrieved as more entities, parameters and metrics and algorithmic layers are added to achieve the final solution as shown below.

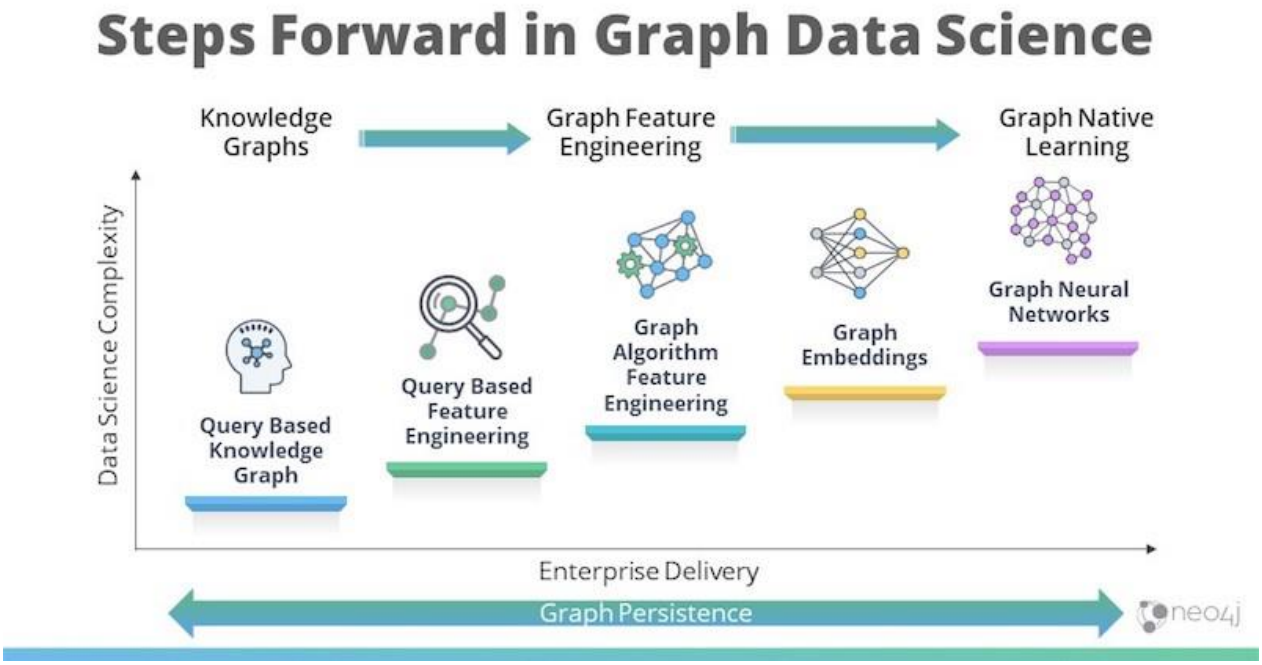

Figure-21 – Graph Data Science Complexity Evolution

Advanced Network Analytics for FIU:

Different analytical approaches to identify and detect potentially suspicious patterns in the property graph are described. These approaches will not only result in new insights and patterns being discovered but also enrich the graph with inferred relationships and threshold scores. These approaches will be contained by an algorithmic recipe, which not only provides specific instructions to achieve the goal but also serves as a general standard for similar use cases. In each approach when the recipe steps are highlighted, the relevant graph algorithm is explained in detail.

The complete list of graph algorithms can be found on: https://neo4j.com/docs/graph-data-science/current/algorithms/

Identifying Potential fraudsters:

Fraud Categories:

- First-party Fraud

  An individual, or group of individuals, misrepresent their identity or give false information when applying for a product or services to receive more favourable rates or when have no intention of repayment.

- Second-party Fraud

  An individual knowingly gives their identity or personal information to another individual to commit fraud or someone is perpetrating fraud in his behalf.

- Third-party Fraud

  An individual, or group of individuals, create or use another person's identity, or personal details, to open or takeover an account.

Algorithmic Recipe for identification of First party fraudsters:

1. Identifying clients sharing personally identifiable information (PII):

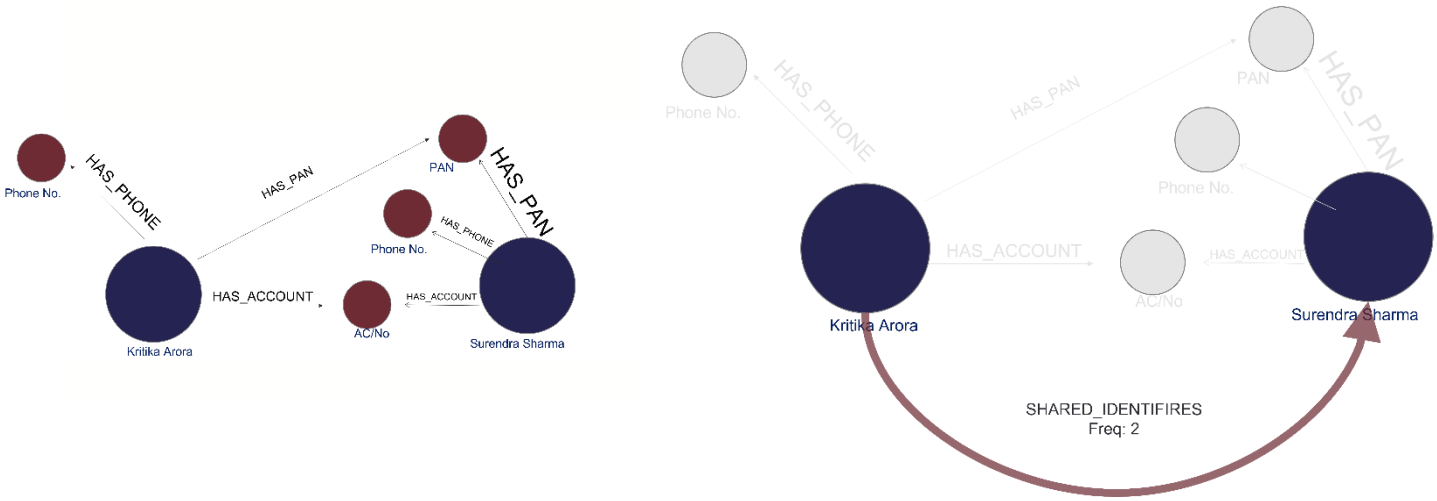

Figure-22 – Creation of Inferred relationship "SHARED_IDENTIFIERS"

Personally identifiable information includes Phone Numbers, PAN Numbers, KYC Details etc. Generally, under normal circumstances such kind of information is unique for every individual. For example, no two people can share the same PAN numbers. If 2 individuals have the same PAN and Phone and email then, they are likely to be the same person and one of them could be a synthetic plant. The first step in this recipe is identifying which individuals share personally identifiable information and how many individuals (PII) they share amongst themselves. A new relationship is then created between individuals which can be called SHARED_IDENTIFIERS having the frequency as its property.

2. Identifying clusters of clients sharing PII using community detection algorithms (Weakly Connected Components)

   The WCC algorithm finds sets of connected nodes in an undirected graph, where all nodes in the same set form a connected component. WCC is often used early in an analysis to understand the structure of a graph. Using WCC to understand the graph structure enables running other algorithms independently on an identified cluster. As a preprocessing step for directed graphs, it helps quickly identify disconnected groups.

   In this case it is used to cluster communities using only the newly created SHARED_IDENTIFIERS relationship

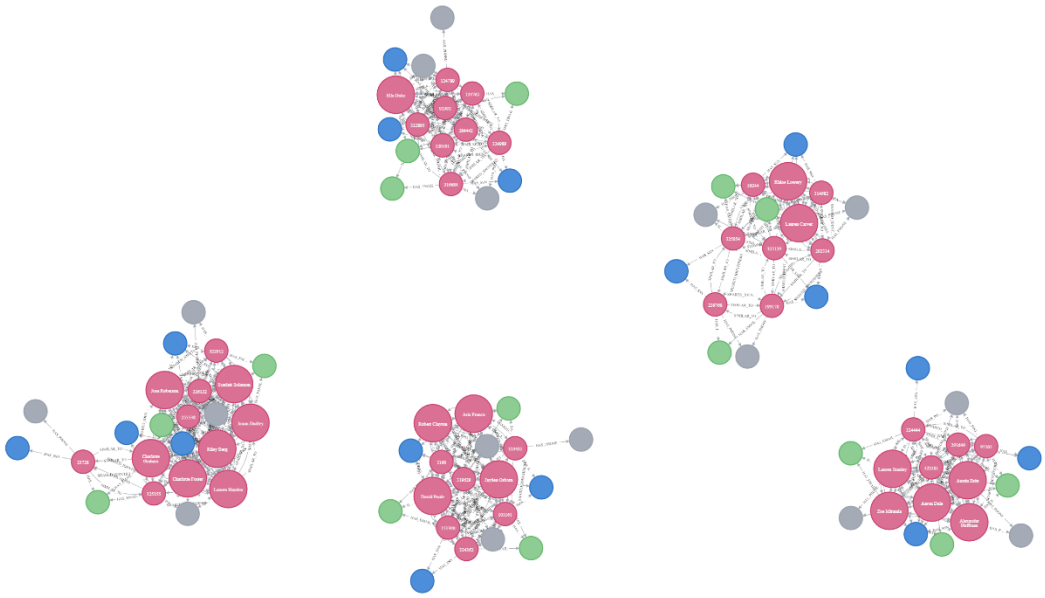

Figure-23 –Clusters of clients sharing PII

3. Finding similar clients within the clusters based on shared identifiers using pairwise similarity algorithms (Node Similarity)

   Finding similar clients with a cluster is accomplished by running one of the pairwise similarity algorithms implemented in the GDS library. Node similarity is used to find similar nodes based on the relationships to other nodes. Node similarity uses Jaccard metric which computes similarity score for a pair of nodes by looking at related nodes in the network that are common to both the nodes divided by the sum of all nodes related to both the nodes.

   Node similarity algorithms work on bipartite graphs (two types of nodes and relationships between them). Here we project client nodes (one type) and three

identifiers nodes (that are considered as second type) into memory. The clients who have these identifiers in common are similar to each other.

Two nodes are considered similar if they share many of the same neighbors.

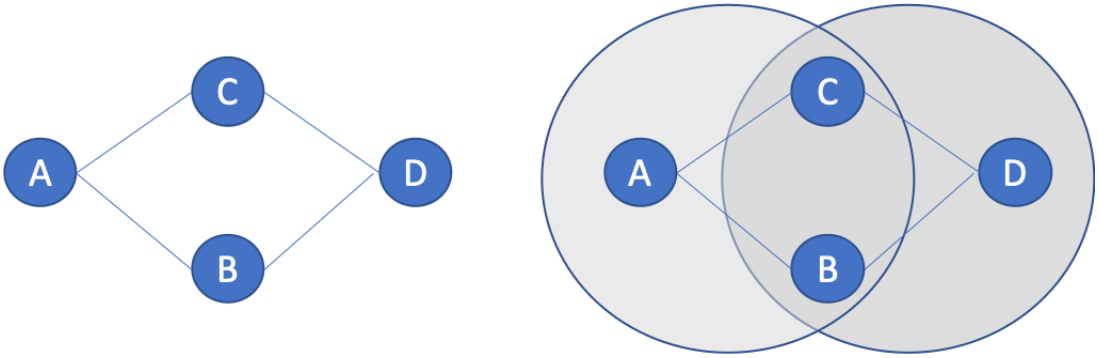

Figure-24 – Node Similarity Algorithm

The results of this algorithm are stored in a new relationship which can be called SIMILAR_TO having a property containing the resultant similarity scores from the algorithm.

4. Calculating and assigning fraud score to clients in clusters using centrality algorithms (Degree Centrality)

   In this step, the fraud score is computed and assigned (firstPartyFraudScore) to clients in the clusters identified in previous steps based on SIMILAR_TO relationships weighted by jaccardScore. We use one of the centrality algorithms (Degree Centrality) to add up jaccardScore on the incoming and outgoing relationships for a given node in a cluster and assign the sum as the corresponding firstPartyFraudScore. This score represents clients who are similar to many others in the cluster in terms of sharing identifiers. Here the assumption is higher the firstPartyFraudScore greater the potential for committing fraud.

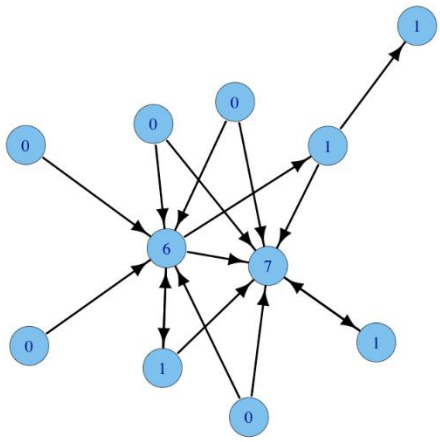

Figure-25 – Degree centrality measures the number of incoming or outgoing (or both) relationships from a node, depending on the orientation of a relationship projection.

Using assigned fraud scores, clients can be labeled as potential fraudsters if their scores fall above a certain threshold.

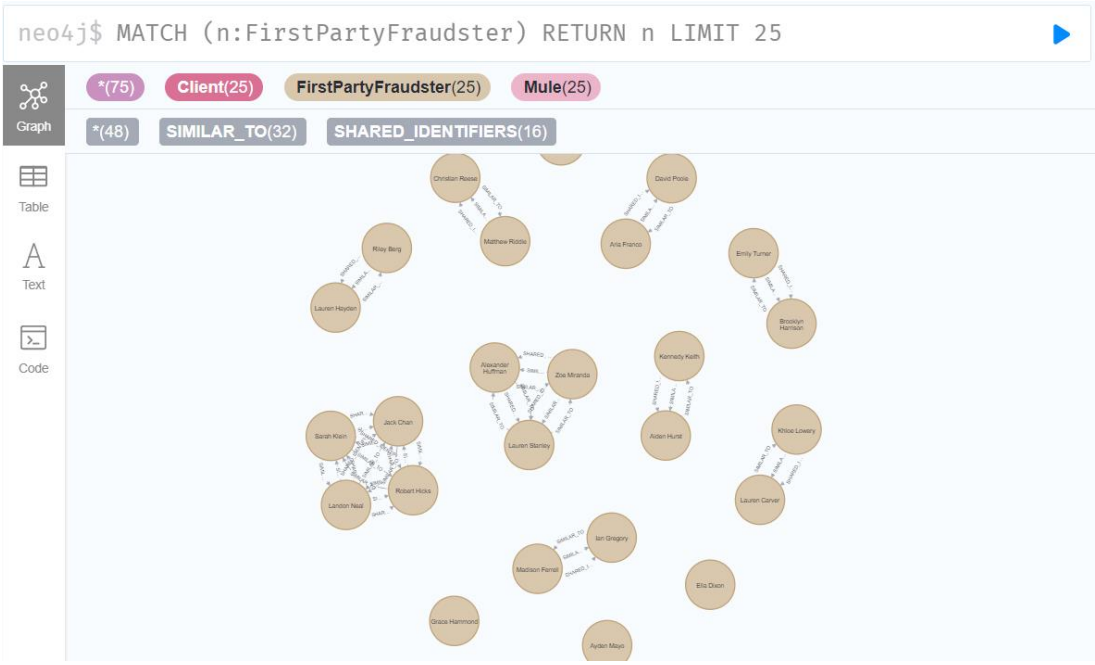

Figure-26 – First Part Fraudsters Identified using their newly created Labels

Circular Trade Ring identification:

Circular money flows is a type of “money laundering” activity in which the money is sent to different accounts, and sent back to the starting node after a certain number of hops. Each participant in the money flow may send a marginally less money in the next transaction compare to the previous one. The total “payment” for the whole travel should not exceed some % loss of the source amount. The flow of transactions will expect that each “next” transaction happens only after the “previous” one has completed. Transactions should be ordered by the creation timestamp.

For a circular trade ring identification the following general schema is required:

Entity Nodes:

1. (:Company) — a business entity, that can send and receive money.
2. (:Client) — a business entity, that can send and receive money.

In a real system, nodes would likely contain specific information about entities, such as identifiers and other attributes. However, for the purposes of this example, only the the existence of these nodes is considered.

Modeling Transactions:

(:Transaction) — an entity, that represents an operation of money transfer between two members. It is a self-sufficient fact, that entity “A” sends money to entity “B”. Relationships between (A) — (Transaction) — (B) helps us to understand who is a sender and who is a receiver.

[:OUT] relationship type — connects the entity of the source node and Transaction. Relationship means that Entity is a money sender.
[:IN] relationship type — connects the entity of a destination node and Transaction. Relationship means that Entity is the receiver of money.

In the FIU database, the transaction modeling complies with the above model. In fact we can see a direct screenshot from the FIU model. We can map the OUT and IN relationships as debitedVia and creditedVia.

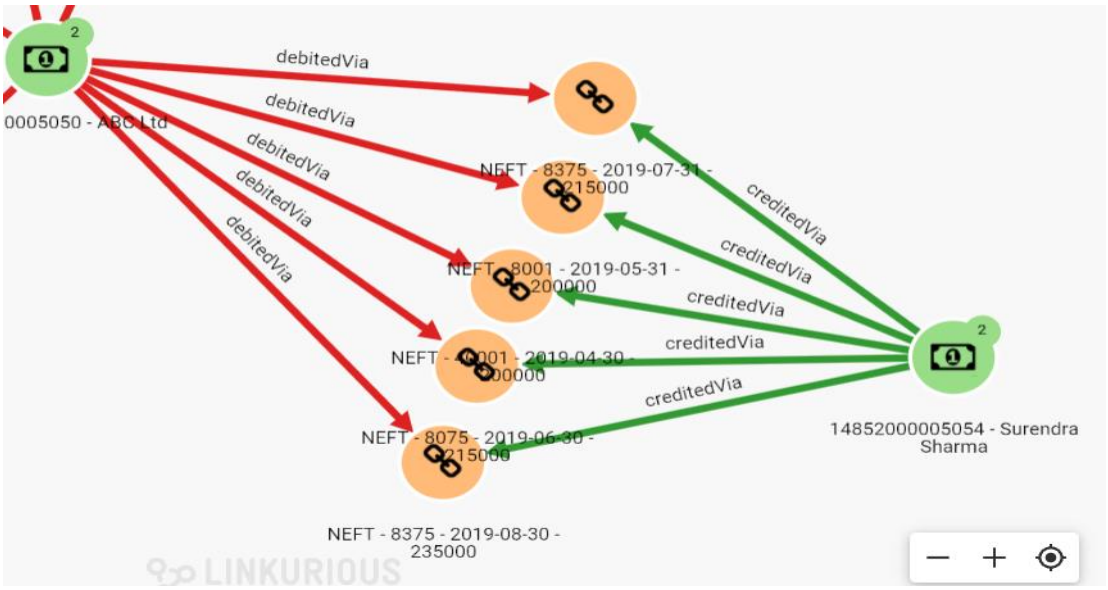

Figure-27 – Transactions modeled in the FIU graph Model

Solution Methodology:

A query needs to be written to find all the chains, starting and ending at the same Client node. In addition, the amount of money on each Transaction node within the chain should not lose more than x% of the source amount. The number of transactions involved in the flow can be up to 10, and they should be ordered by time.

A path like the following pattern needs to be found.

```
(c1)->(t1)->(e2)->(t2)->(e3)->(t3)->...->(t9)->(e10)->(t10)->(c1)
```

Here the starting node c1 (client node) is the same as the ending node. Each client sends money through a transfer (tn) to another client (e1,e2…en).

Each neighboring Transaction should match the following conditions:

- `t1.timestamp < t2.timestamp`
- `t1.amount > t2.amount`

First and last Transactions also should match these additional conditions:

- `(t1.amount – tN.amount) / t1.amount < X`

- start and ends in the same Client: `(t1)<-[:OUT]-(:Client)-[:IN]->(tN)`

What is the money flow? It is a chain of Transactions. Let's forget about business entities for a while and actually find-out, that all we need is actually just **Transactions**. a new "subgraph" of only Transactions can be built. This "transaction subgraph", can be further explored by writing a query to detect circular money flows.

The transaction subgraph can be built by creating a new [:HOP] relationship on the following conditions:

- Transaction A { timestamp } earlier than Transaction B { timestamp }
- Transaction B { amount } less than Transaction A { amount }
- Transaction B { amount } compare to Transaction A { amount } not less than a defined loss percentage

Without the last condition, the subgraph becomes wide-range acceptable.

The following images illustrate the creation of the transaction subgraph.

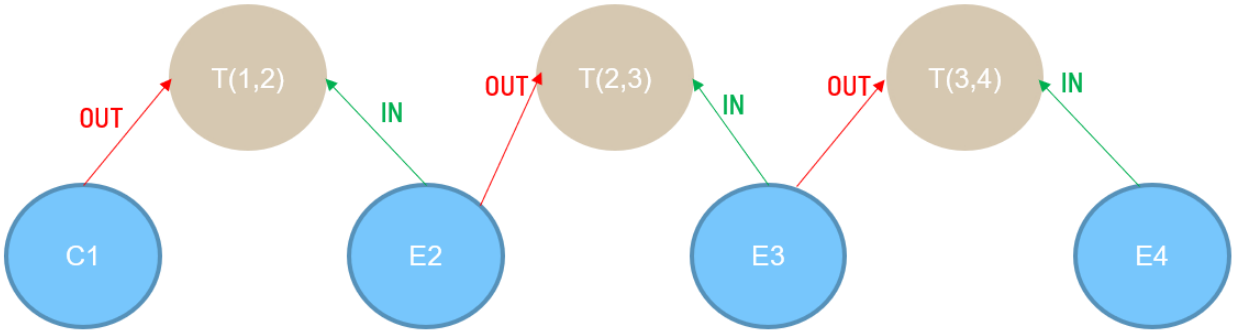

Figure-28 – Initial data model of entities and transactions

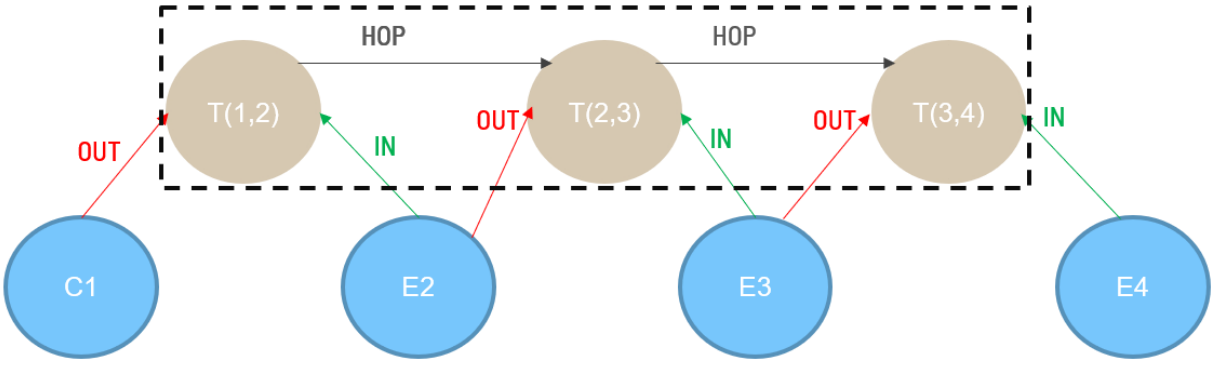

Figure-29 – The transaction subgraph illustrated by dotted lines comprising only of the Transaction nodes and the HOP relationship

The final solution query takes into account the HOP relationships and returns circular rings of transactions. One such ring is shown below.

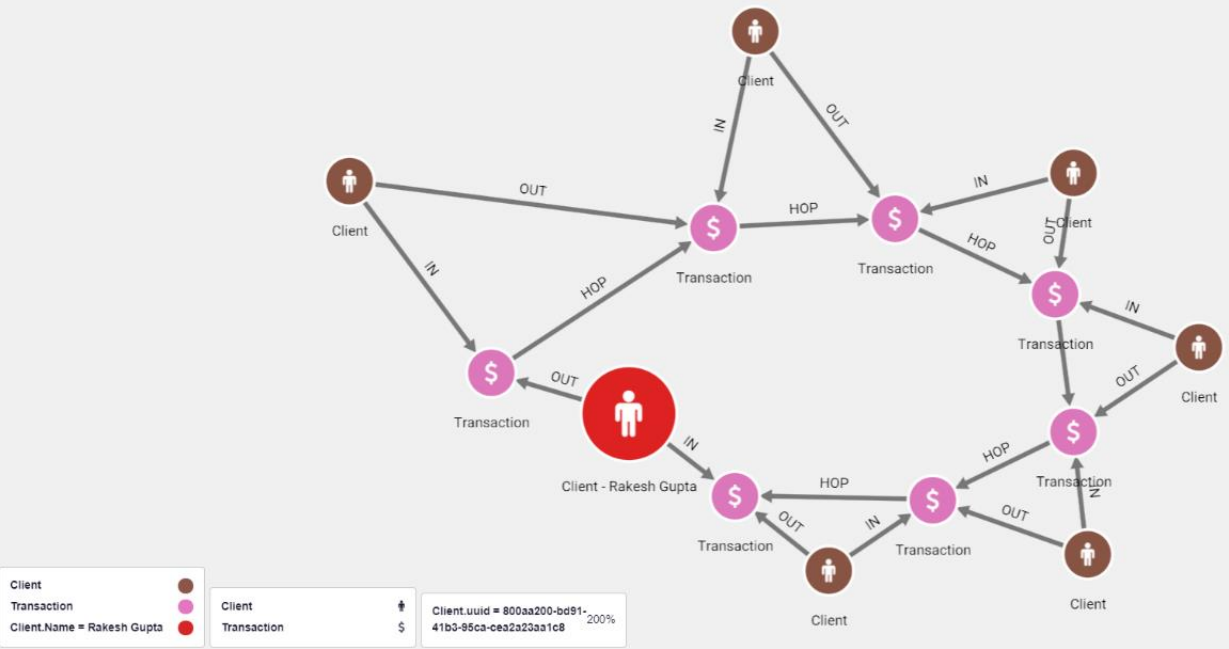

Figure-30 –Circular rings Visualization

The general cypher query to return circular rings given below can be modified to fit the FIU model.

```
MATCH path = (t1:Transaction)-[:HOP*3..10]->(t2:Transaction)
WHERE (t1)<-[:OUT]-(:Client)-[:IN]->(t2)
  AND t1.timestamp < t2.timestamp
  AND (t1.amount - t2.amount) / t1.amount < 0.25
UNWIND nodes(path) as t
MATCH (e)-[:OUT]->(t)
WHERE e:Client OR e:Company
RETURN e, t
```

Further Advanced Network Analytics using GDS :

As mentioned in the previous sections, GDS contains many out of the box algorithms ready to be used directly. These algorithms can be run in memory and directly on the database to achieve the desirable results. A brief summary of all the algorithms available and the categories they belong to is shown in the graphic below.

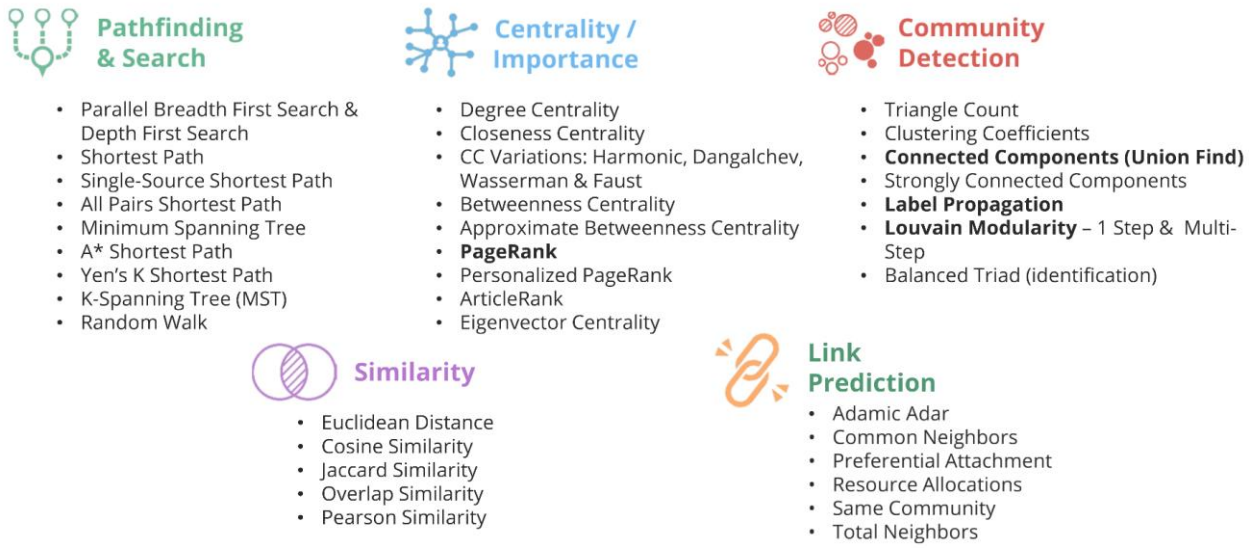

Figure-31 – Graph Data Science Algorithm Catalog

These algorithms when applied in succession create an algorithmic recipe which helps derive level 3 and up insights from the data model thereby enriching the graph and consolidating the knowledge graph model.

6. Network Graph Visualization Integration with FIU FINNET 2.0 other applications and subsystems.

Network Graph Visualization Integration with FIU FINNET 2.0 other applications and subsystems.

Neo4J Data Refresh from SQL Server:

Neo4j Enterprise provides multiple options to import/ingest data from FINcore (MS SQL server), it provides the standard JDBC connector, and host of procedures to manage the data ingestion task, it additionally provides the ETL tool, that allows us to build the connection using the JDBC driver for MSSQL and then manage the task from the ETL task interface. Here, the data of FINcore will be injected to Neo4j using one of the two ways:

1. Neo4j ETL and Data Integration

Neo4j ETL reveals data connections within tabular data stored in an RDBMS and delivers a graphical view by moving data into the Neo4j graph database.

- Connect to MSSQL (FINcore) via JDBC.
- Use a graphical interface to add or remove entities.
- Arrange table structures as labelled nodes and identify JOINs and JOIN tables as graph relationships.
- Map or change labels and properties of nodes prior to execution

Graph connections are materialized through relational joins as data is imported and persisted permanently after import. For more information, the ETL tool can get your data into Neo4j from the live RDBMS.

The data from FINcore to Graph database will go through the following process:

- From FINgate , the data will move to FINcore after applying all the CPS rules.
- Neo4j ETL tool will be provided with the connection details of the MS SQL database like IP address, db name, port, etc.
- Testing of a successful connection will be done.
- Mapping of the entities will be done with the Nodes in the Graph DB. For example, the Case will be mapped with the Case Node with the properties like caseId, batchId, reId, etc.
- The next step will be connecting the nodes with the corresponding relationship using the FINcore tables. For example, bank node will relate to corresponding account.
- Finally, the Neo4j database will be connected with the Linkurious which will later be integrated in the Network Tab of Case Inspector tool.

With Neo4j ETL we will connect Neo4j with:

Supported Sources
Enterprise Edition Oracle MS SQL Server

2. Neo4j Connector in Mosaic

This is the alternate method to import data into Neo4j. Mosaic tool contains Neo4j connector in which we will map the data to nodes and relationships.
The data coming in FINcore will be mapped using the Mosaic neo4j connector and will be stored in the Graph database in the form of Nodes and Relationships.

The data from FINcore to Graph database will go through the following process:

- From FINgate , the data will move to FINcore after applying all the CPS rules.
- Mosaic neo4j connector will be provided with the connection details of the MS SQL database like IP address, db name, port, etc.
- Testing of a successful connection will be done.
- Mapping of the entities will be done with the Nodes in the Graph DB. For example, the Case will be mapped with the Case Node with the properties like caseId, batchId, reId, etc. and the corresponding query will be :
- MERGE (c:Case{case_id: Case_ID, batch_id: batchId, re_id: reId);
- The next step will be connecting the nodes with the corresponding relationship using the FINcore tables. For example, bank node will relate to corresponding account and the query for that will be:
  MATCH (b:Bank),(a:Account)
  MERGE (b)<-[:WITH_BANK]-(a);
- Finally, the Neo4j database will be connected with the Linkurious which will later be integrated in the Network Tab of Case Inspector tool.

Neo4J – Linkurious Enterprise Integration with Portal

The FINnet 2.0 portal will connect with be having Case Inspector Tool containing the Network Tab. It will contain Case network of entities, their attributes and their related relationship with a confidence score and a timeline view capability and with a capability to create a What-If network diagram.

Hence, the portal integration will be done via Linkurious enterprise visualization interface using iFrame.
A brief on the method of doing that is explained in the notes below.

The Linkurious REST API is the core way to interact with the Linkurious server.

- Request:

All API access occur over HTTP (or HTTPS, if enabled). All data is sent and received as JSON. We use HTTP response codes (and a few code extensions from RFC 4918) to indicate API errors. The base URL for each API method looks like the following:
http://yourdomain.com/api/service/method?parameters
or, if the service is relevant to a given data Source:
http://yourdomain.com/api/:dataSource/service/method?parameters
Where possible, our API try to use the appropriate HTTP verb for each User action:

Verb	**Description**
GET	Used for retrieving resources
POST	Used for creating resources
PATCH	Used for updating resources
PUT	Used for replacing resources
DELETE	Used for deleting resources

- Parameters

In GET requests, any parameter not specified as a segment in the path have to be passed as an HTTP query string parameter.
https://your-linkurious-instance.com/guest/?key=your-datasource-key
For example, in the GET /api/:dataSource/graph/nodes/:id we will have the parameters dataSource and id, but we can also specify the optional with edges parameter as such:
http://yourdomain.com/api/a2e3c50f/graph/nodes/2?with_edges=true
In POST, PATCH, PUT and DELETE requests, any parameter not included in the URL is supposed to be sent in the JSON request body.

- Successful responses

The HTTP status codes 200, 201 or 204 indicates a successful response, for example:

```
HTTP/1.1 200 OK
{
  "status": {
    "code": 200,
    "name": "initialized",
    "message": "Connection Succesful",
    "uptime": 122
  }
}
```

Errors

HTTP status codes greater than or equal to 400 indicates an error. The response contains an error key and a human-readable message, for example:

```
HTTP/1.1 401 Unauthorized
{
  "key": "unauthorized",
```

```
  "message": "Unauthorized."
}
```

Possible errors can be:

Error key	HTTP Code	Description
invalid_parameter	400	A parameter provided to the API does not have the expected type/value
unauthorized	401	The current action required to be authenticated
readonly_right	403	You do not have the right to edit
not_found	404	The node selected with this API was not found
critical	500	An unexpected technical error has occurred

Neo4J Graph Changes in SQL Server

Data from Linkurious graph visualization enterprise could provide output streams in JSON objects, which could then be parsed and ingested into MSSQL server with a relevant connector.
The flow of process and rules which will be done when the FIU Analyst wants to add or delete any particular entity or relationship are given below:

Addition of entity or adding a relation:
• Dissemination with Approval, /Retention or RE Report rejection
  ○ The case will be enhanced and refreshed with revised case and entity risk after the final approver's approval.
  ○ The changes will be made permanent in the entire network system(including RDBMS) after the final approver but before ingestion to Finex.
• Dissemination without Approval
  ○ The case will be enhanced and refreshed with revised case and entity risk after the Analyst submits dissemination action but before ingestion to FINex.

Removal of the entity from the network graph will be restricted to the specific case level only. • The reason for the removal has to be recorded.   ○ On a predefined frequency of period, as deemed appropriate by FIU-IND, a dump of such cases (in which entities have been removed) will be evaluated by FIU-IND and post-approval the change will be made permanent in the network system including RDBMS.
Entities can not be added or removed from the case network if the case is already disseminated and no reprocessing will happen when the case has been disseminated.

However, these methods are still in exploratory level and need further investigation, testing and review before introducing them formally into the solution.

Alternatively, the Mosaic connector to Neo4j could be deployed for bi-directional data movement between Neo4j and SQL server. Current version of the connector can move data from SQL server to Neo4j, however the Mosaic team was confident about providing a bidirectional data movement subsequently.

Author: Shridhar Iyengar | CEO | Slink Information Systems