Tvam Graph Document

Version 1.1

Introduction

The document contains the details of Graph implementation for Tvam within BangDB. The following details are covered in the document

- The Entities and their properties
- The relationship between entities and their properties
- Efficiency of this Graph model
- Flexibility and how to edit/update the graph model for future changes
- Cluster and Similarities, and the properties used for creating them
- Query templates and examples
- Query benchmarking for ~300K data in the schema [using REST QUERY]

The document provides a detail for each entity and relation with several properties that are attached to these. Sample view of the part of the graph is also given everywhere. In the end the overall graph is also printed.

The graph structure needs to be flexible and extensible in nature with logical hierarchy also preserved as far as possible. And above this, it is also very important that the graph scales well and performance for both read and write is high. Further, we must also ensure that data preserves the natural groups, cliques, segments etc. Finally, we must also ensure that the Graph uses nomenclatures which is natural to the domain and aligns with RDF models.

To ensure that, these flexibilities are there in the structure, the document also lists several queries that might be performed in production and to enable the use cases. The queries are benchmarked for REST call (which is most conservative and would kind of provide the baseline). The results of the queries are also depicted pictorially for clarity. Further, these queries can be extended or more can be added as needed. As of now it lists over 40 such query templates with over 100 possible variations for different permutations and combinations for different contexts. This is to ensure that there is enough flexibility in the structure to query in many ways.

Note: The Graph, and related details (including queries) are subject to the receipt and test on actual data. These may change once we get the data, but the current structure should take care of the change easily

Entities, relationships, and their properties

1. Person
    - CustRefID (Customer ID and TvamCustRefID): Node

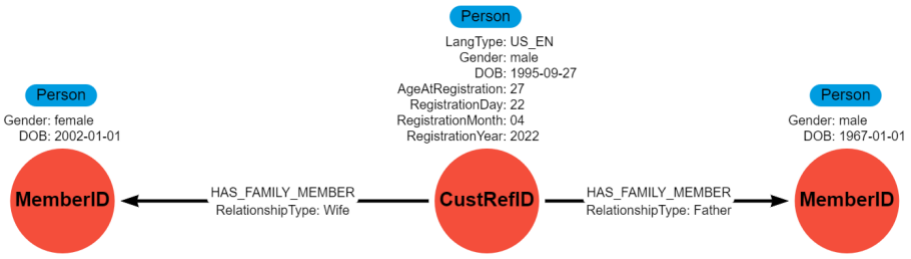
Nullable	No
Properties	LangType, Gender, DOB, AgeAtRegistration, RegistrationDay, RegistrationMonth, RegistrationYear, CreatedDate
Node Modifiability	No
Property Modifiability	Yes
Property Extensibility	Yes

    - MemberID (FamilyMemberID and FamilyMemID): Node

Nullable	No
Properties	Gender, DOB
Node Modifiability	No
Property Modifiability	Yes
Property Extensibility	Yes

    - HAS_FAMILY_MEMBER: Relation between CustRefID and MemberID

Properties	RelationType
Modifiability	No
Extensibility	Yes

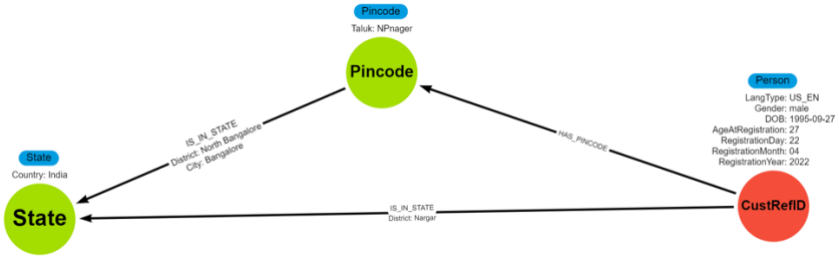


    - IS_IN_STATE: Relation between CustRefID to State

Properties	District
Modifiability	No
Extensibility	Yes

    - HAS_PINCODE: Relation between CustRefID to Pincode

Properties	
Modifiability	No
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

a. Find all members of the family for a given person
b. Find all customers in a given cluster/area (city/taluk/geo-loc)/similar customers
c. Find relative (fathers/mothers…) of all the registered customer in a given cluster/area (city/taluk/geo-loc)/similar customers
d. Find all $1^{st}/2^{nd}$ degree connections for a given customer
e. Add further filters in above queries (age range, gender, education etc.)

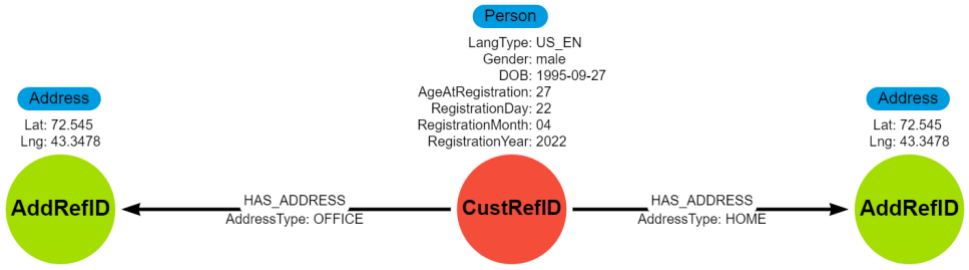
2. Address
    - AddRefID: Node

Nullable	No
Properties	Latitude and Longitude
Node Modifiability	No
Property Modifiability	Yes
Property Extensibility	Yes

There is a node for every AddRefID because a customer can have more than one address (such as HOME, OFFICE) and for every address we will have latitude and longitude associated with it.

- HAS_ADDRESS: Relation between CustRefID (Person) and AddRefID (Address Node)

Properties	AddressType
Modifiability	No
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

f. Find all addresses for a given customer. Or find an office address for a customer.
g. Find all customers for a given office address?
h. Find all customers whose office and home addresses are same
i. Find all customers whose office addresses are missing
j. Find all customer living in a given radius for a given pair of Lat, Lon

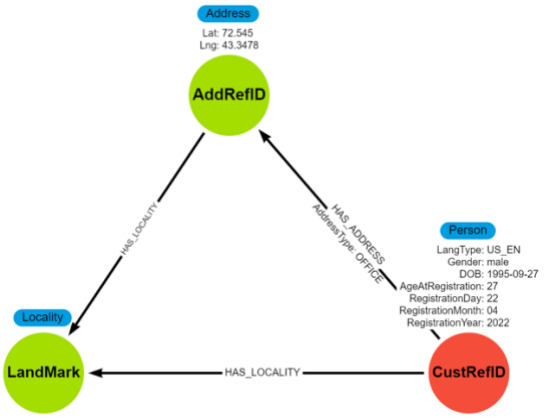
3. Locality

- Locality (Ex: Village name): Node

Nullable	Yes
Properties	
Node Modifiability	No
Property Modifiability	Yes
Property Extensibility	Yes

- HAS_LOCALITY: Relation between CustRefID and Locality

Properties	
Modifiability	Yes
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

k. Find all customers for a given Locality

l. Find all customers for whom Locality is not given

4. Pincode

- PinCode: Node

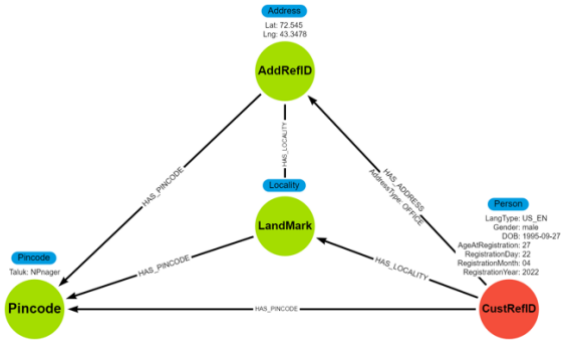
Nullable	Yes
Properties	Taluk
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- HAS_PINCODE: Relation from AddRefID and Locality to Pincode

Properties	
Modifiability	Yes
Extensibility	Yes

The "taluk" is used as a property of node Pincode and not as individual node due to following reasons

- One taluk can have more than one Pincode
- Not every customer provides taluk.
- The taluk is a local administration unit consisting of a few villages or a few wards (taluk's definition differs from state to state). So, using it as a property is better than as a node.



How to handle null pincode?

Pincode node will be created, and this will be linked with state. The node will have Taluk as property. Since the pincode was null, therefore we will use "Lat, Lon" to figure out the pincode. If "Lat, Lon" is also null, then we can use "taluk" info for the same. If "taluk" is also null, then we may use the central pin of the city/district. Since pincode is editable/updateable node, therefore this can be updated whenever we are able to get it or compute it.

Few important queries [detail queries are at the end of the document]

m. Find all customers for a given pincode
n. Find all customers within a X radius of given pincode
o. Find all customers who has purchased a product (or a given product) for a given pincode or within radius of X KM for a pincode
p. Find similar people (based on any criteria) for a neighborhood of a given pincode

5. State

- State: Node

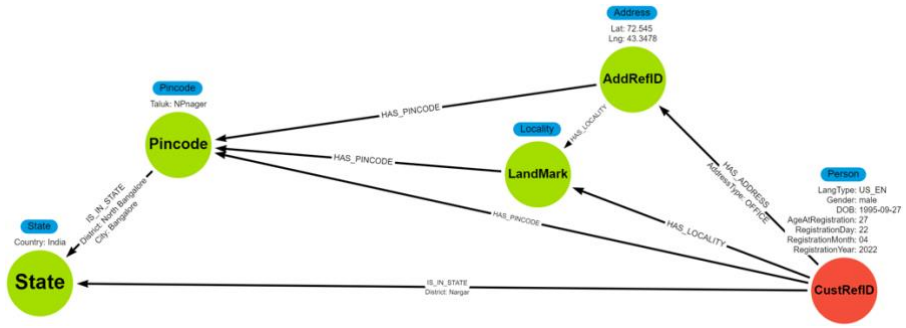
Nullable	No
Properties	Country
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- IS_IN_STATE: Relation from Pincode to State [Inversion]

Properties	District, City
Modifiability	Yes
Extensibility	Yes

- IS_IN_STATE: Relation from CustRefID to State

Properties	District
Modifiability	Yes
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

q. Find all active pin codes for a given state
r. Find all customers for a given state (who has or hasn't purchased any product)
s. Find all similar profile people (based on any criteria) who are in a given state

6. Profile

- CustProfile: Node

Nullable	No
Properties	Family Structure, Education, Profession, Marital Status, Children, SchoolGoingKids, NoOfSeniorCitizen, Dependent, Income, Business Category, Updated Date
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

CustProfile as a separate node to represent a customer's general information which we get from surveys (Question Response), and we know that not every customer takes surveys so it's better to have a node that represents a customer profile.

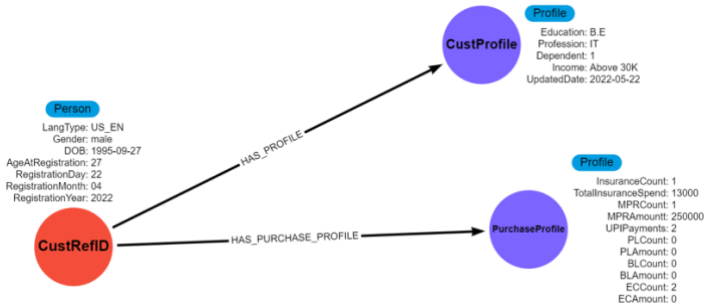
- PurchaseProfile: Node

Nullable	No
Properties	InsuranceCount, TotalInsuranceSpend, PersonalLoans, LoanAmount, UPIPayments, UPICount, UPIAmount
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- HAS_PROFILE: Relation between CustRefID node and CustProfile node

Properties	
Modifiability	Yes
Extensibility	Yes

- HAS_PURCHASE_PROFILE: Relation between CustRefID node and PruchaseProfile node



Few important queries [detail queries are at the end of the document]

t. Find all people similar to given profile (uses profile-based similarity)
u. Find all people who has their profile similar to a given person (same as above except it takes PersonID)
v. Add various filters (edu, prof, etc.)

7. DoctorConsultation

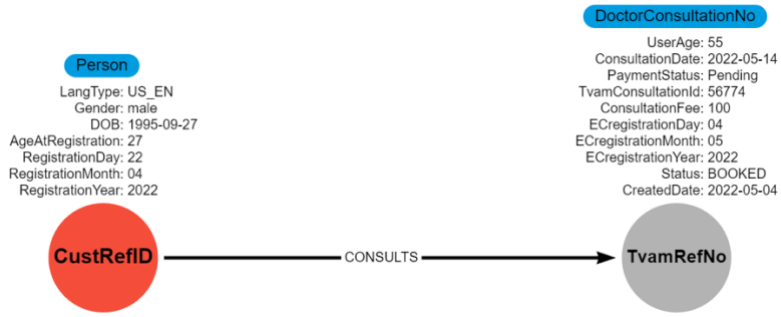
- TvamRefNo: Node

Nullable	No
Properties	AgeAtEC, ConsultationDate, PaymentStatus, TvamConsultationId, ConsultationFee, ECregistrationDay, ECregistrationMonth, ECregistrationYear, Status, CreatedDate
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

TvamRefNo is used as node instead of ConsultationId because there can be cases where ConsultationId is not present.

- CONSULTS: Relation between CustRefID and TvamRefID

Properties	
Modifiability	Yes
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

w. Find all persons who have consulted doctors (and num of times for each)
x. Find all persons who have consulted doctors (and num of times for each) in a given geo-loc (city, taluk or even pincode)
y. Find all persons who have booked the consultations but not yet consulted
z. Find all persons who have consulted Doctors but not yet purchased insurance
aa. Find persons similar to a given person who has consulted doctor but not purchased insurance yet
bb. Find different policies for insurances and their providers where people have consulted the Doctors most (top 3 or 5)
cc. Find all aging fathers (different relatives) of persons similar to a given person in an area who has taken insurance

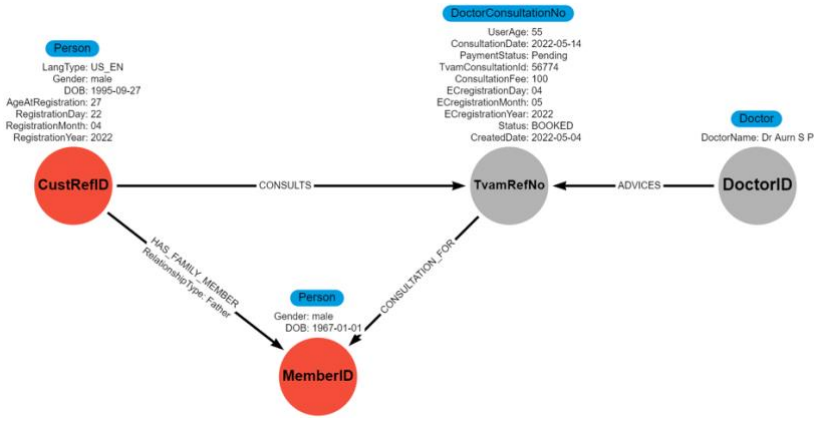
8. Doctor

- DoctorID: Node

Nullable	No
Properties	DoctorName
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- ADVICES: Relation between DoctorID and TvamRefNo

Properties	
Modifiability	Yes
Extensibility	Yes



9. PolicyVendor

- VendorID: Node

Nullable	No
Properties	VendorCode
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

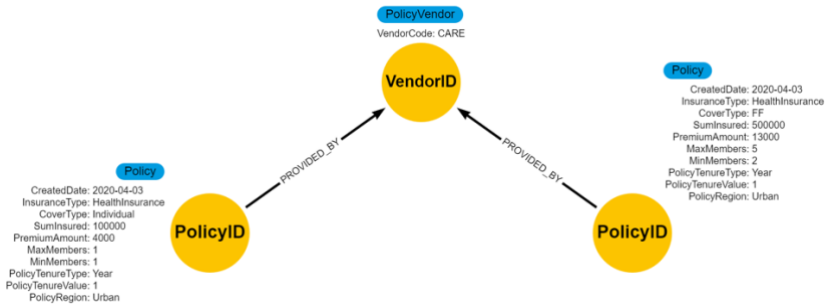
10. Policy

- PolicyID: Node

Nullable	No
Properties	CreatedDate, Insurance Type, Cover Type, Sum Insured, Premium Amount, MaxMembers, Policy Tenure Type, Policy Tenure Values, Policy Region, MinMembers, Allowed Relationships
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- PROVIDED_BY: Relation between PolicyID and VendorID

Properties	
Modifiability	Yes
Extensibility	Yes



11. Insurance

- InsuranceID: Node (for all Insurances)

Nullable	No
Properties	AgeAtInsurance, Policy Type, Policy Status, Policy Creation Month, Policy Creation Year, Policy Maturity Month, Policy Maturity Year
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

There is a node for every insurance purchased. This node is connected to the customer and the policy.

- COVERS: Relation between InsuranceID and MemberID

Properties	
Modifiability	Yes
Extensibility	Yes

- IS_FOR_POLICY: Relation between InsuranceID and PolicyID

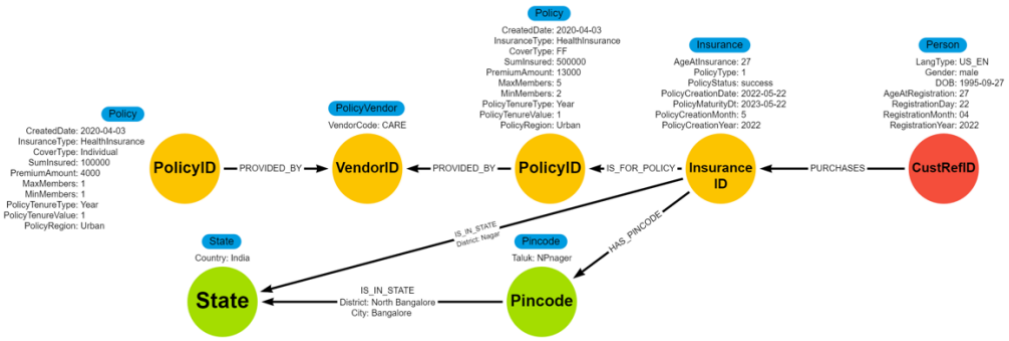
Properties	
Modifiability	Yes
Extensibility	Yes

- IS_IN_STATE: Relation between InsuranceID and State

Properties	District
Modifiability	Yes
Extensibility	Yes

- HAS_PINCODE: Relation between InsuranceId to Pincode

Properties	District
Modifiability	Yes
Extensibility	Yes



12. Loan

- LoanApplicationId: Node (For all Types of Loans)

Nullable	No
Properties	AgeAtLoanRegistration, DateOfBirth, Gender, Marital Status, Education Qualification, Purpose of Loan, related Loan details, Employment details, loan registration day, month, and year
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

We have 2 types of loans

- Personal and,
- Business Loan

There is a node for every loan application type. These nodes are updated when Loan application status changes.

Since the Tvam app customer and actual loan customer may be different therefore, we have Gender, Marital Status, and DOB as node properties.

- APPLIES_FOR: Relation between CustRefID and LoanApplicationID

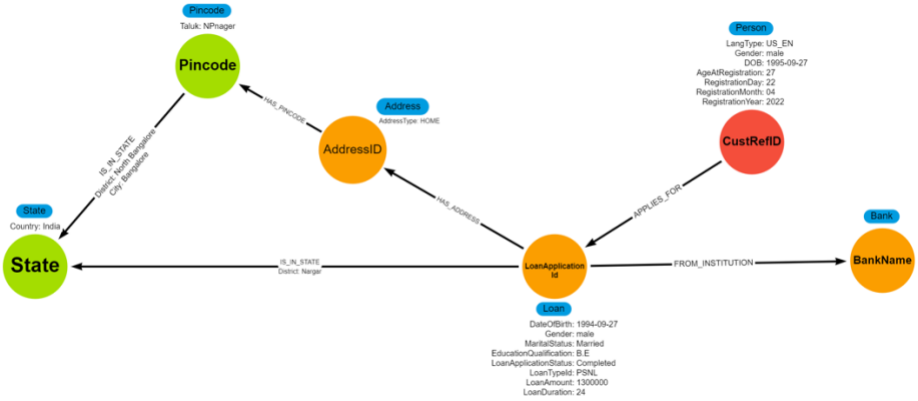
Properties	
Modifiability	Yes
Extensibility	Yes

- IS_IN_STATE: LoanApplicationId to State

Properties	District
Modifiability	Yes
Extensibility	Yes

- HAS_ADDRESS: LoanApplicationId to AddressId

Properties	
Modifiability	Yes
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

dd. Find all persons who have taken loans in a given area (pincode, geo-loc, city, taluk, state)

ee. Find all persons who similar to persons who have taken loan in a given area

ff. Find all ageing fathers (or any relative) of persons similar to a given person in an area who has taken loan (may add filters like loan-amount > X or duration < Y years etc.)

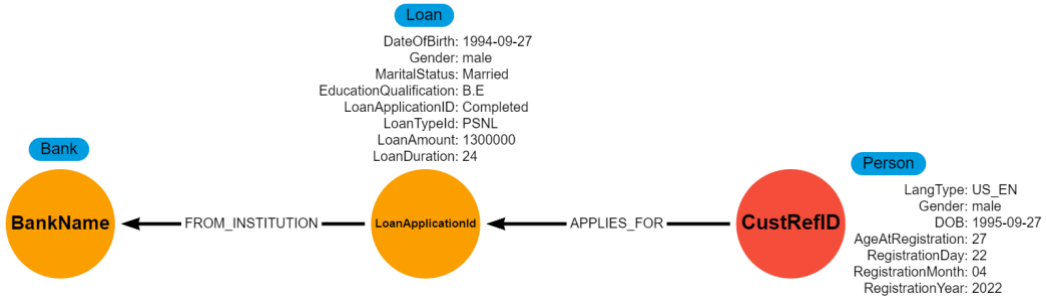
13. FinancialInstitute

- BankName: Node

Nullable	No
Properties	
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

- FROM_INSTITUTION: Relation between LoanApplicationId to BankName
- APPLIES_FOR: Relation between CustRefID and LoanApplicationId

Properties	
Modifiability	Yes
Extensibility	Yes



Few important queries [detail queries are at the end of the document]

gg. Find all customers who have applied for a loan (for any INSTITUTION or a given one) from a given geo-loc/area/Locality/pincode/geo-distance (further filter for customer profile like age, loan amount etc.)

hh. Top K banks who have offered loans to customers in a given area (further ass filter like, loan amount more than X etc.)

14. Transaction

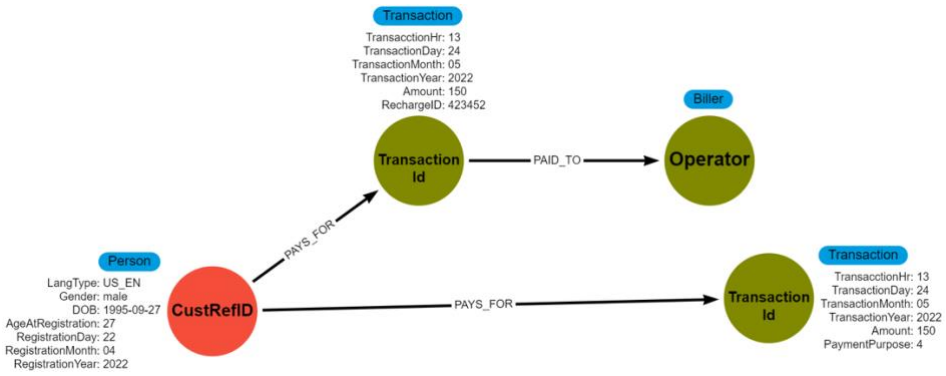
- TransactionId: Node (For MPR, UPI transfers)

Nullable	No
Properties	Amount, Payment Type, Transaction Type, PaymentTypeCategory, PayeePaymentAddress, MerchantCatCode, TransactionHr, TransactionDay, TranscationMonth, TransactionYear and TransactionDate
Node Modifiability	Yes
Property Modifiability	Yes
Property Extensibility	Yes

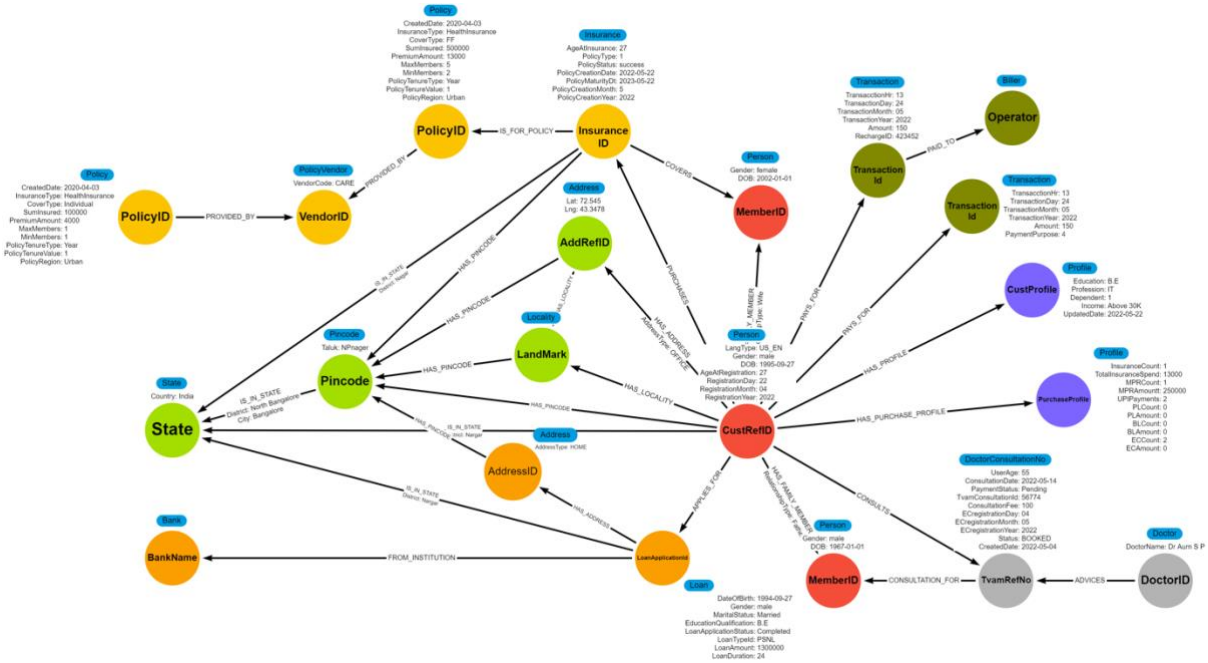
Note: Properties differ a little in the case of MPR (How?)

- PAYS_FOR: Relation between CustRefID and TransactionId
- PAYS_TO: Relation between TransactionId and Operator

Properties	
Modifiability	Yes
Extensibility	Yes



Graph Diagram



Queries

1. **For a Given user, Listing users from same village/Locality**

S1=>(@P1 Person:cf2b3d8d-f8f8-4e92-85cd-4511b21f7a15)-[@l1 HAS_LOCALITY]->(@L Locality:*)<-[@l2 HAS_LOCALITY]-(@P2 Person:*);RETURN P2.name AS CustRefID, P2.AgeAtRegistration AS Age, P2.Gender AS Gender, P2.LangType AS Language,P2.RegistrationYear AS Year, P2.RegistrationMonth AS Month, P2.RegistrationDay AS Day

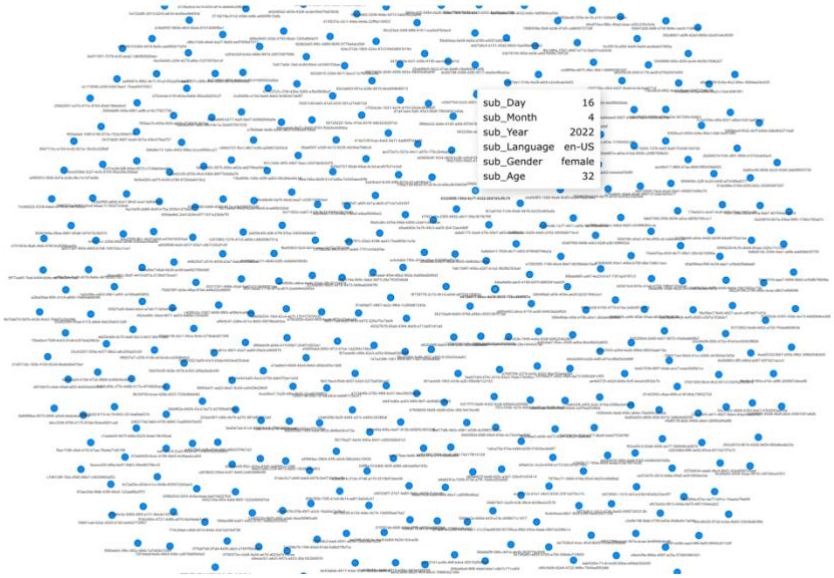
REST API Execution Time: 222 ms

DB Execution Time: < 2 ms

If we query on Ampere (BangDB FE), we will see the results like these and we can also create dashboards for the same for different logically related queries

CustRefID	Day	Month	Year	Language	Gender	Age
a780260e-42e9-480f-bedd-3f74a72aba25	5	4	2022	en-US	female	43
8fcc03e4-549f-4f66-8161-cca5e97b0ac4	28	4	2022	en-US	female	25
b578ea31-cb62-4fa9-aa73-556fc681cf00	11	4	2022	en-US	female	30
248317dd-fab5-4765-8d90-7ea89947bb55	15	4	2022	en-US	female	26
e3df8f9e-d975-4fbc-96b1-888f00951f2f	20	4	2022	en-US	female	31

Rows per page: 5 1–5 of 430



These graphs/charts can be zoomed in, moved, adjusted, property could be revealed etc. for further analysis

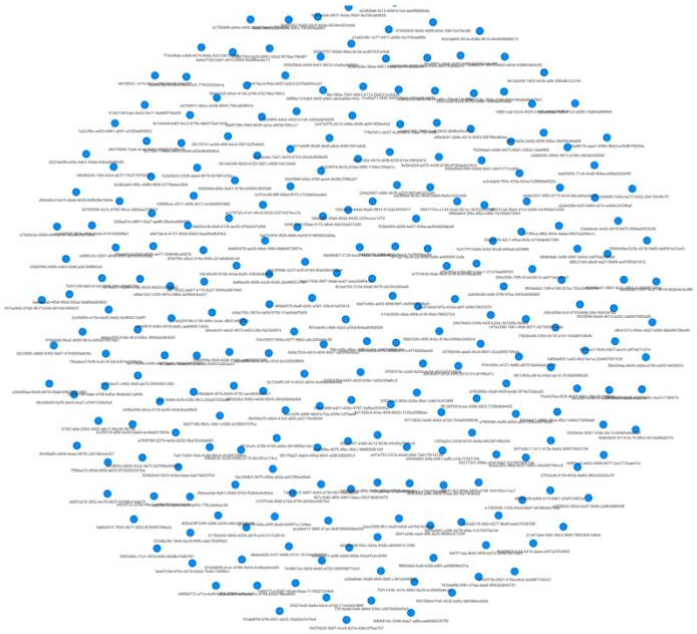
2. For a given user, Listing users from same village and same gender and AgeGroup

S1=>(@P1 Person:cf2b3d8d-f8f8-4e92-85cd-4511b21f7a15)-[@l1 HAS_LOCALITY]->(@L Locality:*)<-[@l2 HAS_LOCALITY]-(@P2 Person:* { Gender = "female" AND AgeAtRegistration > 26});RETURN P2.name AS CustRefID, P2.Gender AS Gender, P2.LangType AS Language,P2.RegistrationYear AS Year, P2.RegistrationMonth AS Month, P2.RegistrationDay AS Day

REST API Execution Time: 457 ms

sub	sub_Day	sub_Month	sub_Year	sub_Language	sub_Gender	sub_Age
a780260e-42e9-480f-bedd-3f74a72aba25	5	4	2022	en-US	female	43
b578ea31-cb62-4fa9-aa73-556fc681cf00	11	4	2022	en-US	female	30
e3df8f9e-d975-4fbc-96b1-888f00951f2f	20	4	2022	en-US	female	31
6f51380e-d81d-43b2-ae14-7036df389b20	14	4	2022	en-US	female	45
d8bc7cb6-d4e8-4a27-8b69-ee97856d1912	3	4	2022	en-US	female	29

Rows per page: 5 1–5 of 251



Note: we will not put charts and graphs for all, but user can do as he/she wishes

3. For a given user, List users from same Village and same cluster as the cluster of the given user

S1=>(@P Person:*)-[@k Cluster1 {_cluster_ = $$Person:a387386d-ac1e-4a90-82b0-d7217a9a53bd}]->(@P2 Person:*)-[@l HAS_LOCALITY]->(@L Locality:*);RETURN P2.name AS CustRefID, L.name AS Village WHERE Village = "Samastipur"REST API Execution Time: 220 ms

Village	CustRefID
Samastipur	a387386d-ac1e-4a90-82b0-d7217a9a53bd
Samastipur	b2252a93-d86a-48d1-b797-889fc732abf0
Samastipur	de8e92e1-367e-43db-abe9-02735910dd3a
Samastipur	30fb9e00-0600-44a1-8ad9-b3603d565da2
Samastipur	ebb694f3-ab45-4c9c-9c59-390882c37c55

Rows per page: 5 1–5 of 6

4. For a given Pincode, Counting Number of Villages, Takul and Users connected to the same Pincode

S1=>(@P Person:*)-[@l HAS_LOCALITY]->(@L Locality:*)-[@z HAS_PINCODE]->(@Z Pincode:713202);RETURN Z.name AS Pincode, COUNT(Z.Taluk) AS Number_Taluks, COUNT(L.name) AS Total_Villages, COUNT(P.name) AS Total_Customers

REST API Execution Time: 301 ms

Pincode	Total_Villages	Total_Customers	Number_Taluks
713202	4	4	0

Rows per page: 5 1–1 of 1

5. For State, Number of district, Number Pincode, Number of Taluk, Number of Villages and Number of Customers

S1=>(@P Person:*)-[@f HAS_LOCALITY]->(@A Locality:*)-[@z HAS_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Assam);RETURN S.name AS State, UCOUNT(s.District) AS Total_District, UCOUNT(Z.name) AS Total_Pincode, UCOUNT(Z.Taluk) AS Number_Taluks, UCOUNT(A.name) AS Total_Villages, UCOUNT(P.name) AS Total_Customer

REST API Execution Time: 4321 ms

State	Total_District	Total_Villages	Total_Customer	Total_Pincode	Number_Taluks
Assam	28	28	28	509	0

Rows per page: 5 1–1 of 1

6. Listing all Customer Profiles available in the database

S1=>(@P Person:*)-[@h HAS_PROFILE]->(@C Profile:*);RETURN P.name AS CustRefID, P.Gender AS Gender, P.DOB AS DOB,MATH_EXP("DATE((CURTIME-$DOB)/30806700000000)") AS Age, P.AccountAge AS Accountage,C.FamilySturcture AS Familystructure, C.MaritalStatus AS Marital_Status, C.Education AS Education, C.Profession AS Profession, C.Income AS Income, C.Dependent AS Dependent, C.NoOfSeniorCitizen, C.Children AS Children, C.BusinessCategory AS BusinessCategory

REST API Execution Time: 456 ms

CustRefID	BusinessCategory	Children	NoOfSeniorCitizen	Dependent	Income	Profession	Education	Marital_Status	Age	DOB	Gender	Accountage
07d2aa79-c5a9-4552-b41f-0365c1b1dccf	Koch Bihar	0	0	0	2022-05-19 09:43:34	Home	ef92b460-3aac-432a-9df6-affa0b847e2b	West Bengal	0			
87228cb3-10b0-43af-a12e-2ddfcfe50a12	Y.S.R.	0	0	0	2022-05-06 03:48:22	Home	844bd8a3-2119-4580-9264-24f4bc6819ee	Andhra Pradesh	0			
c22dba0d-75ec-4d40-b82e-f4424d8748e4	Kolkata	0	0	0	2022-05-19 12:48:36	Home	883de4d2-8ed3-472d-8fe7-2b69bc15483d	West Bengal	0			
4393486c-bbd3-4750-882f-ec23851d2f6b	Kheda	0	0	0	2022-05-31 11:51:35	Home	03b949c4-426a-4229-94b1-2a93e3866e53	Gujarat	0			
d79af77f-fad2-410d-84f2-0f87c54a34c3	Dakshina Kannada	0	0	0	2022-05-15 02:11:55	Home	50587907-5077-46ce-9f6d-98f37fee80a9	Karnataka	0			

Rows per page: 5 1–5 of 4085

7. Listing all Customers Profile for a village

S2=>[S1=>(@P Person:*)-[@h HAS_PROFILE]->(@C Profile:*)]-[@l HAS_LOCALITY]->(@L Locality:*);RETURN P.name AS CustRefID, P.Gender AS Gender, P.DOB AS DOB,MATH_EXP("DATE((CURTIME-$DOB)/30806700000000)") AS Age, P.AccountAge AS Accountage,C.FamilySturcture AS Familystructure, C.MaritalStatus AS Marital_Status, C.Education AS Education, C.Profession AS Profession, C.Income AS Income, C.Dependent AS Dependent, C.NoOfSeniorCitizen, C.Children AS Children, C.BusinessCategory AS BusinessCategory, L.name AS Village WHERE Village = "Baksa"

REST API execution time (ms) : 226 ms

Village	CustRefID	DOB	Gender	BusinessCategory	Children	NoOfSeniorCitizen	Dependent	Income	Profession	Education	Marital_Status	Age
Baksa	9c288812-c2f9-463e-988f-eaa9f7bd818c	1987-12-14	male	Pub Ltd Co.	4	1	1	Greater than 30K	Student	SSC	Divorced	35.42704447382368

Rows per page: 5 1–1 of 1

8. List State Wise – all registered Customers and their Member count

S1=>[S2=>(@P1 Person:*)-[@f HAS_FAMILY_MEMBER]->(@P2 Person:*)]-[@a HAS_ADDRESS]->(@A Address:*)-[@z WITH_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:*);RETURN S.name AS State, s.District AS District, COUNT(P1.name) AS Customers, COUNT(P2.name) AS Registered_Members

REST API execution time (ms) : 3300ms

District	State	Customers	Registered_Members
Sant Ravidas Nagar (Bhadohi)	Uttar Pradesh	10	10
Supaul	Bihar	72	72
Nizamabad	Andhra Pradesh	60	60
Jamnagar	Gujarat	4	4
Visakhapatnam	Telangana	28	28

Rows per page: 5 1–5 of 478

9. List State and Pincode Wise – Insurance Policy count

S2=>[S1=>(@I Insurance:*)-[@o IS_FOR_POLICY]->(@P Policy:*)]-[@f HAS_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Assam);RETURN S.name AS State, Z.name AS Pincode,P.name AS PolicyId, COUNT(I.name) AS PoliciesIssued, SUM(P.PremiumAmount) AS TolatPremium

REST API execution time (ms) : 2073 ms

PolicyId	Pincode	State	TolatPremium	PoliciesIssued
CH929014	784507	Assam	379288	28
CH544526	785619	Assam	44100	12
CH544526	788723	Assam	29400	8
CH824054	784529	Assam	207900	20
CH540849	782120	Assam	112140	12

Rows per page: 5 1–5 of 275

For a given pincode, example = 784507, same query

S2=>[S1=>(@I Insurance:*)-[@o IS_FOR_POLICY]->(@P Policy:*)]-[@f HAS_PINCODE]->(@Z Pincode:784507)-[@s IS_IN_STATE]->(@S State:Assam);RETURN S.name AS State, Z.name AS Pincode,P.name AS PolicyId, COUNT(I.name) AS PoliciesIssued, SUM(P.PremiumAmount) AS TolatPremium

REST API execution time (ms) : 295 ms

10. Customer Purchase Profile for a village and Pincode

S2=>[S1=>(@P Person:*)-[@r1 HAS_PURCHASE_PROFILE]->(@F Profile:*)]-[@h HAS_LOCALITY]->(@L Locality:*)-[@w HAS_PINCODE]->(@Z Pincode:802202);RETURN P.name AS CustRefID, F.ECCount AS Doctor_Appointments,F.ECAmount AS EC_AmountSpend,F.InsuranceCount AS PolicyPurchased,F.InsuranceAmount AS PolicyAmountSpend,F.MPRCount AS PrepaidRecharges,F.MPRAmount AS AmountSpendMPR,F.UPICount AS UPIPayments,F.UPIAmount AS UPIAmountSpend,F.PLCount AS PL_applied,F.PLAmount AS TotalPLAmount,F.BLCount AS BL_applied,F.BLAmount AS TotalBLAmount,MATH_EXP("(((((\$Doctor_Appointments + \$PolicyPurchased)+\$PrepaidRecharges)+\$UPIPayments)+\$PL_applied)+\$BL_applied)") AS Total_Usage,MATH_EXP("(((((\$EC_AmountSpend+\$PolicyAmountSpend)+\$AmountSpendMPR)+\$UPIAmountSpend)+\$TotalPLAmount)+\$TotalBLAmount)") AS Total_Spend, L.name AS village, Z.name AS Pincode

REST API execution time (ms) : 290 ms

village	Doctor_Appointments	TotalBLAmount	TotalPLAmount	UPIAmountSpend	UPIPayments	PrepaidRecharges	PolicyAmountSpend	PolicyPurchased	CustRefID	Pincode	Total_Spe
yisd	0	0	0	0	0	0	7246	1	e93beb5a-b30b-4272-ba39-d1fe469e14fb	802202	0
Bhojpur	0	0	0	0	0	0	7246	1	e93beb5a-b30b-4272-ba39-d1fe469e14fb	802202	0
Bhojpur	0	0	0	0	0	0	7246	1	e93beb5a-b30b-4272-ba39-d1fe469e14fb	802202	0
Bhojpur	0	0	0	0	0	0	7246	1	e93beb5a-b30b-4272-ba39-d1fe469e14fb	802202	0
Bhojpur	0	0	0	0	0	0	7246	1	e93beb5a-b30b-4272-ba39-d1fe469e14fb	802202	0

Rows per page: 5 1–5 of 5

11. List State Wise – Service performance

S2=>[S1=>(@P Person:*)-[@r1 HAS_PURCHASE_PROFILE]->(@F Profile:*)]-[@h HAS_ADDRESS]->(@A Address:*)-[@w WITH_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:*);RETURN S.name AS State, SUM(F.ECCount) AS Doctor_Appointments, SUM(F.ECAmount) AS EC_AmountSpend, SUM(F.InsuranceCount) AS PolicyPurchased, SUM(F.InsuranceAmount) AS PolicyAmountSpend, SUM(F.MPRCount) AS PrepaidRecharges, SUM(F.MPRAmount) AS AmountSpendMPR, SUM(F.UPICount) AS UPIPayments, SUM(F.UPIAmount) AS UPIAmountSpend, SUM(F.PLCount) AS PL_applied, SUM(F.PLAmount) AS TotalPLAmount, SUM(F.BLCount) AS BL_applied, SUM(F.BLAmount) AS TotalBLAmount, MATH_EXP("(((((($Doctor_Appointments + $PolicyPurchased)+$PrepaidRecharges)+$UPIPayments)+$PL_applied)+$BL_applied)") AS Total_Usage, MATH_EXP("(((((($EC_AmountSpend+$PolicyAmountSpend)+$AmountSpendMPR)+$UPIAmountSpend)+$TotalPLAmount)+$TotalBLAmount)") AS Total_Spend, COUNT(P.name) AS CustomerCount

REST API execution time (ms) : 240ms

12. List District wise – performance for a given state

S2=>[S1=>(@P Person:*)-[@r1 HAS_PURCHASE_PROFILE]->(@F Profile:*)]-[@h HAS_ADDRESS]->(@A Address:*)-[@w WITH_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE {District = "Koch Bihar"}]->(@S State:*);RETURN S.name AS State, s.District AS District,SUM(F.ECCount) AS Doctor_Appointments, SUM(F.ECAmount) AS EC_AmountSpend, SUM(F.InsuranceCount) AS PolicyPurchased, SUM(F.InsuranceAmount) AS PolicyAmountSpend, SUM(F.MPRCount) AS PrepaidRecharges, SUM(F.MPRAmount) AS AmountSpendMPR, SUM(F.UPICount) AS UPIPayments, SUM(F.UPIAmount) AS UPIAmountSpend, SUM(F.PLCount) AS PL_applied, SUM(F.PLAmount) AS TotalPLAmount, SUM(F.BLCount) AS BL_applied, SUM(F.BLAmount) AS TotalBLAmount, MATH_EXP("(((((($Doctor_Appointments + $PolicyPurchased)+$PrepaidRecharges)+$UPIPayments)+$PL_applied)+$BL_applied)") AS Total_Usage, MATH_EXP("(((((($EC_AmountSpend+$PolicyAmountSpend)+$AmountSpendMPR)+$UPIAmountSpend)+$TotalPLAmount)+$TotalBLAmount)") AS Total_Spend, COUNT(P.name) AS CustomerCount

REST API execution time (ms) : 229ms

13. List all transaction for all years and month

s1=>(@P Person:*)-[@t PAYS_FOR]->(@T Transaction:*); RETURN T.TransactionYear AS Year, T.TransactionMonth AS Month, T.TransactionDay AS Day, T.TransactionHR AS Hr , COUNT(T.name) AS TotalTransactions, UCOUNT(P.name) AS Users

REST API execution time (ms) : 281 ms

Year	Hr	Day	Month	Users	TotalTransactions
2021	10	10	4	6	7
2021	14	13	9	4	4
2021	15	10	4	3	3
2021	22	17	10	2	2
2021	14	21	9	2	2

Rows per page: 5 1–5 of 603

Similarly, same query for a given year and month

s1=>(@P Person:*)-[@t PAYS_FOR]->(@T Transaction:* {TransactionYear = "2021" AND TransactionMonth = "4" }); RETURN T.TransactionYear AS Year, T.TransactionMonth AS Month, T.TransactionDay AS Day, T.TransactionHR AS Hr , COUNT(T.name) AS TotalTransactions, UCOUNT(P.name) AS Users

OR same query with WHERE clause

s1=>(@P Person:*)-[@t PAYS_FOR]->(@T Transaction:*); RETURN T.TransactionYear AS Year, T.TransactionMonth AS Month, T.TransactionDay AS Day, T.TransactionHR AS Hr , COUNT(T.name) AS TotalTransactions, UCOUNT(P.name) AS Users WHERE Year = "2021" AND Month = "4"

Note: The filter for property is better than WHERE from performance reason (most of the time), WHERE should be used for filtering with computed and aggregated values generally

14. List all Insurance for a given year and month

s1=>(@P Person:*)-[@i PURCHASES]->(@I Insurance:*); RETURN I.PolicyCreationYear AS Year, I.PolicyCreationMonth AS Month, I.PolicyCreationDay AS Day , COUNT(I.name) AS TotalInsurances, COUNT(P.name) AS Users WHERE Year = "2022" AND Month = "5"

REST API execution time (ms) : 303 ms

Year	Day	Month	Users	TotalInsurances
2022	10	5	116	116
2022	15	5	73	73
2022	24	5	9	9
2022	26	5	1	1
2022	21	5	24	24

Rows per page: 5 1–5 of 27

15. List all Doctor_Appointmernt for a given year and month

s1=>(@P Person:*)-[@d CONSULTS]->(@D DoctorConsultationNo:* {ECregistrationYear = "2022" AND ECregistrationMonth = "1"}); RETURN D.ECregistrationYear AS Year, D.ECregistrationMonth AS Month, D.ECregistrationDay AS Day , COUNT(D.name) AS TotalAppointments, COUNT(P.name) AS Users

REST API execution time (ms) : 343 ms

Year	Day	Month	Users	TotalAppointments
2022	4	1	2	2
2022	16	1	4	4
2022	14	1	2	2
2022	7	1	2	2

Rows per page: 5 1–4 of 4

16. List all Loan Applications for a given year and month

s1=>(@P Person:*)-[@l APPLIES_FOR]->(@L Loan:*); RETURN L.LoanRegistrationYear AS Year, L.LoanRegistrationMonth AS Month , COUNT(L.name) AS TotalApplications, COUNT(P.name) AS Users WHERE Year = "2022" AND Month = "5"

REST API execution time (ms) : 268 ms

Year	Month	TotalApplications	Users
2022	5	362	362

Rows per page: 5 1–1 of 1

17. Listing all Cluster groups for a given cluster and the number of customers within each group

S1=>(@P1 Person:*)-[@k Cluster1]->(@P2 Person:*); RETURN k._cluster_ AS Cluster, COUNT(P1.name) AS Users, AVG(P2.Age) AS AvgAge

REST API execution time (ms) : 391 ms

Cluster	Users	AvgAge
0	1914	53
1	207	0
2	3742	53
3	160	0
4	4831	0

Rows per page: 5 1–5 of 5

18. Find all members of the family for a given person

S1=>(@P Person:*)-[@f HAS_FAMILY_MEMBER]->(@P2 Person:*);RETURN P.name AS Customer, f.RelationShipType AS Relation, P2.name AS MemberID, P2.Gender AS Member_Gender, P2.DOB AS Member_DOB, MATH_EXP("DATE((CURTIME-$Member_DOB)/30806700000000)") AS Member_Age WHERE Customer = "19e3e644-4fc1-4c79-a984-9202a1300544"

REST API execution time (ms) : 595 ms

Relation	Customer	MemberID	Member_DOB	Member_Gender	Member_Age
daughter	19e3e644-4fc1-4c79-a984-9202a1300544	dbc6e9cc-6409-4d4b-aabe-c5835267d02b	null	Female	53.81391935319145
daughter	19e3e644-4fc1-4c79-a984-9202a1300544	dbc6e9cc-6409-4d4b-aabe-c5835267d02b	null	Female	53.81391935319252
daughter	19e3e644-4fc1-4c79-a984-9202a1300544	ce28af5d-bc8d-460a-8a1c-91f64ba273e3	null	Female	53.81391935319359
son	19e3e644-4fc1-4c79-a984-9202a1300544	e74c10d3-3abc-4f2f-aa0b-c9ed5c1d4ded	null	Male	53.81391935319463
wife	19e3e644-4fc1-4c79-a984-9202a1300544	353a11b9-ce9d-46d5-9f43-931332173b3b	null	Female	53.81391935319564

Rows per page: 5 1–5 of 6

19. Find all customer in a given Village/Locality/State/ District or Pincode

S1=>(@P Person:*)-[@l HAS_LOCALITY]->(@L Locality:Kokrajhar)-[@z HAS_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Assam);RETURN P.name AS Customer,L.name AS Village,Z.name AS Pincode, Z.Taluk AS Taluk, s.District AS District, S.name AS State WHERE Pincode = "783373"

REST API execution time (ms) : ms

Village	District	Customer	Pincode	State
Kokrajhar	Kokrajhar	d2fe901a-36e1-4c49-913a-040847d4f39d	783373	Assam
Kokrajhar	Kokrajhar	d2fe901a-36e1-4c49-913a-040847d4f39d	783373	Assam
Kokrajhar	Chirang	d2fe901a-36e1-4c49-913a-040847d4f39d	783373	Assam
Kokrajhar	Kokrajhar	d2fe901a-36e1-4c49-913a-040847d4f39d	783373	Assam
Kokrajhar	Chirang	d2fe901a-36e1-4c49-913a-040847d4f39d	783373	Assam

Rows per page: 5 1–5 of 10

20. Find all Customer for a given cluster group within a cluster (list all the customers here)

S1=>(@P Person:*)-[@k Cluster1 {_cluster_ = 0}]->(@P2 Person:*); RETURN P.name AS Customer,k._cluster_ AS Cluster

REST API execution time (ms) : 506 ms

21. Find relatives for all registered customers in a given area

S2=>[S1=>(@P Person:*)-[@f HAS_FAMILY_MEMBER]->(@P2 Person:*)]-[@l HAS_LOCALITY]->(@L Locality:Thoothukkudi)-[@z HAS_PINCODE]->(@Z Pincode:*);RETURN P.name AS Customer, f.RelationShipType AS Relation, P2.name AS MemberId, L.name AS Village, Z.name AS Pincode WHERE Pincode = "628902"

REST API execution time (ms) : 361 ms

22. Find relatives for all registered customers in a given cluster group

S2=>[S1=>(@P1 Person:*)-[@k Cluster1 {_cluster_ = 0}]->(@P2 Person:*)]-[@f HAS_FAMILY_MEMBER]->(@P3 Person:*); RETURN P2.name AS Customer, f.RelationShipType AS Relation,P3.name AS MemberID

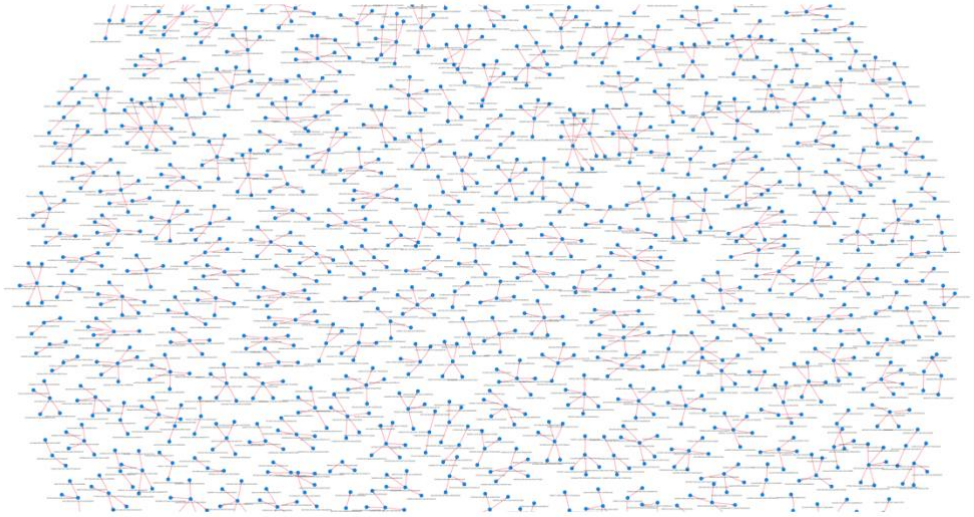
REST API execution time (ms) : 607 ms

The results are like following

Relation	MemberID	Customer
mother	5e2e47c4-d582-4246-bd49-a308f4dd8463	0877330b-0767-4303-ab1d-2b51f48c51ed
wife	c990a9da-d167-4d34-aabb-e6aa552d6d57	0877330b-0767-4303-ab1d-2b51f48c51ed
son	a972b334-2917-473e-852b-6963c5c1ecfc	0877330b-0767-4303-ab1d-2b51f48c51ed
daughter	2e878eec-055a-41b5-93e4-d8814ef23cdb	0877330b-0767-4303-ab1d-2b51f48c51ed
son	df0e71f4-30a6-4daa-9b98-446c7e92d3bf	b59a0528-f19d-4721-b735-158b186db80e

Rows per page: 5 1–5 of 1420

And if we visualize all the results, note this query returns all the customers registered and their relatives within the same cluster group. One can zoom in and out and see details etc.



23. Find all 1th degree connections for a given customer [note we can have nth degree]

S1=>(@P Person:cf2b3d8d-f8f8-4e92-85cd-a511b21f7a15)-[*]->(*)

REST API execution time (ms) : 220 ms

24. Find connection count, adding filter for age, gender and connection count

S1=>(@P Person:*)-[*]->(*); RETURN P.name AS Customer, COUNT(P.name) AS Connections, P.Gender AS Gender, P.DOB AS DOB, MATH_EXP("DATE((CURTIME-$DOB)/30806700000000)") AS Age WHERE Gender = "female" AND Age > 20 AND Connections > 1

REST API execution time (ms) : 219 ms

Customer	DOB	Gender	Connections	Age
be8f51e5-c5d1-4dd3-9a0c-bb9230a3a970	1995-09-20	female	2	27.47032048494973
b9c12964-4180-4ca6-8d3f-4c2dbfdc0bb2	1998-07-30	female	2	24.54233404044698
ccfdd4fc-4d2e-4b5a-86b8-d3439ffb3895	1962-01-30	female	2	61.92744831754265
ec08d855-e150-44ad-aa8f-d8d5bd13025d	1998-09-08	female	2	24.4301506517849
9f6af496-f6ee-475a-a2c8-c543913aee49	1995-08-01	female	2	27.61054972081966

Rows per page: 5 1–5 of 118

25. Find all addresses for a given customer

S1=>(@P Person:aba74d6b-9ce0-4974-9ad8-8aced33adc3d)-[@a HAS_ADDRESS]->(@A Address:)-[@z WITH_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Assam);RETURN P.name AS Customer, a.AddressType AS Address_Type, Z.Taluk AS Taluk, s.District AS District, S.name AS State, Z.name AS Pincode

REST API execution time (ms) : 223 ms

26. Find all customers for a given office address

S1=>(@P Person:)-[@a HAS_ADDRESS]->(@A Address:*)-[@z WITH_PINCODE]->(@Z Pincode:*)-[@s IN_STATE]->(@S State:*);RETURN P.name AS Customer, a.AddressType AS Address_Type, Z.Taluk AS Taluk, s.District AS District, S.name AS State, Z.name AS Pincode WHERE Pincode = "533296" AND Address_Type = “OFFICE”

REST API execution time (ms) : ms

27. Find all customers whose office and home addresses are same

S1=>(@P1 Person:*)-[@a1 HAS_ADDRESS {AddressType = "OFFICE"}]->(@A Address:*)<-[@a2 HAS_ADDRESS {AddressType = "OFFICE"}]-(@P2 Person:*);RETURN P2.name AS Customers, A.name AS AddRefID

REST API execution time (ms) : ms

28. Find all customer living in a 1KM radius for a given pair of Lat, Lon of a specific user

S=>(@p Person: aba74d6b-9ce0-4974-9ad8-8aced33adc3d)-[#RADIUS_OF 1000 {geo_distance = “lat.lon”}]->(@p2 Person:*)

REST API execution time (ms) : 320 ms

29. Find all customers who has purchased a given product for a given pincode

S2=>[S1=>(@I Insurance:*)-[@o IS_FOR_POLICY]->(@P Policy:CH673061)]-[@f HAS_PINCODE]->(@Z Pincode:*);RETURN Z.name AS Pincode, P.name AS Policy, COUNT(I.name) AS Total_Insurance WHERE Pincode = "743247"

REST API execution time (ms) : 232 ms

Policy	Pincode	Total_Insurance
CH673061	743247	1

Rows per page: 5 1–1 of 1

30. Find all similar profile people (based on any criteria) who are in a given state

S2=>[S1=>(@P Person:*)-[@k Cluster1 {_cluster_ = 0}]->(@P2 Person:*)]-[@a HAS_ADDRESS]->(@A Address:*)-[@z WITH_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Bihar); RETURN P.name AS Customer, Z.name AS Pincode,s.District AS District,S.name AS State

REST API execution time (ms) : 2001 ms

31. Find all people similar to given profile (uses profile-based similarity)

S2=>[S1=>(@P1 Person:*)-[@h HAS_PURCHASE_PROFILE]->(@C Profile:5c11fd50-79b6-4da6-87a6-a483f738cc63)]-[@p PurchaseSimilarity]->(@P2 Person:*);RETURN P1.name AS Customer,P2.name AS Similar_Customer,p.similarity AS Similarity WHERE Similarity > 0.85

REST API execution time (ms) : 281 ms

Customer	Similar_Customer	Similarity
5c11fd50-79b6-4da6-87a6-a483f738cc63	cf2b3d8d-f8f8-4e92-85cd-a511b21f7a15	0.999979

Rows per page: 5 1–1 of 1

32. Find all people who has their profile similar to a other person (same as above except it doesn't take PersonID) - find all pairs

S1=>(@P1 Person:*)-[@p PurchaseSimilarity]->(@P2 Person:*);RETURN P1.name AS Customer,P2.name AS Similar_Customer,p.similarity AS Similarity WHERE Similarity > 0.95

REST API execution time (ms) : 567 ms

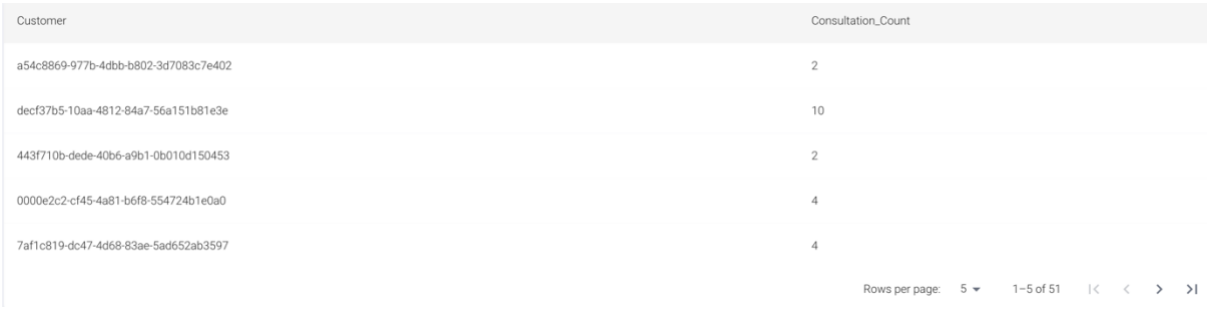
Customer	Similar_Customer	Similarity
5c11fd50-79b6-4da6-87a6-a483f738cc63	cf2b3d8d-f8f8-4e92-85cd-a511b21f7a15	0.999979
19e3e644-4fc1-4c79-a984-9202a1300544	5c11fd50-79b6-4da6-87a6-a483f738cc63	0.952518
19e3e644-4fc1-4c79-a984-9202a1300544	cf2b3d8d-f8f8-4e92-85cd-4511b21f7a15	0.982852
19e3e644-4fc1-4c79-a984-9202a1300544	cbc246ea-fc15-4a66-8a34-a8a0e7e91303	0.9578979999999999
19e3e644-4fc1-4c79-a984-9202a1300544	e93beb5a-b30b-4272-ba39-d1fe469e14fb	0.9882489999999999

Rows per page: 5 1–5 of 93

33. Find all persons who have consulted doctors (and number of times for each)

S1=>(@P Person:*)-[@d CONSULTS]->(@D DoctorConsultationNo:*);RETURN P.name AS Customer, COUNT(P.name) AS Consultation_Count

REST API execution time (ms) : 225 ms



Customer	Consultation_Count
a54c8869-977b-4dbb-b802-3d7083c7e402	2
decf37b5-10aa-4812-84a7-56a151b81e3e	10
443f710b-dede-40b6-a9b1-0b010d150453	2
0000e2c2-cf45-4a81-b6f8-554724b1e0a0	4
7af1c819-dc47-4d68-83ae-5ad652ab3597	4

Rows per page: 5 1–5 of 51

34. Find all persons who have consulted doctors (and number of times for each) in a given geo-loc (city, taluk or even pincode)

S2=>[S1=>(@P Person:*)-[@d TAKES_CONSULTATION]->(@D TvamRefNo:*)]-[@a LIVES_IN]->(@L Locality:*)-[@z HAS_PINCODE]->(@Z Pincode)-[@s IN_STATE]->(@S State:*);RETURN P.name AS Customer, L.name AS Village, Z.Taluk AS Taluk, Z.name AS Pincode, s.District AS District, S.name AS State WHERE Pincode = “786663”

REST API execution time (ms) : ms

35. Find all persons who have booked the consultations but not yet consulted

S1=>(@P Person:*)-[@d CONSULTS]->(@D DoctorConsultationNo:*);RETURN P.name AS Customer, D.Status AS Status WHERE Status = "BOOKED"

REST API execution time (ms) : 233 ms

36. Find all persons who have consulted Doctors but not yet purchased insurance

S1=>(@P Person)-[@p HAS_PURCHASE_PROFILE]->(@C Profile:*);RETURN P.name AS Customer, C.ECCount AS Doctor_Consultations, C.InsuranceCount AS Insurances WHERE Doctor_Consultations > 0 AND Insurances = 0

REST API execution time (ms) : 221 ms

37. Find persons similar to a given person who has consulted doctor but not purchased insurance yet

S2=>[S1=>(@P1 Person:*)-[@k Cluster1 {_cluster_ = $$Person:65dfb98c-e97e-473a-bdc5-03c21c6b2764}]->(@P2 Person:*)]-[@p HAS_PURCHASE_PROFILE]->(@C Profile:*);RETURN P2.name AS Customer, C.ECCount AS ECCount, C.InsuranceCount AS Insurance WHERE ECCount > 0 AND Insurance = 0

REST API execution time (ms) : 256 ms

38. Find different policies for insurances and their providers where people have consulted the Doctors most (top 3 or 5)

S2=>[S1=>(@P Person:*)-[@d CONSULTS]->(@D DoctorConsultationNo:*)]-[@p PURCHASES]->(@I Insurance:*)-[@o IS_FOR_POLICY]->(@X Policy:*)-[@v PROVIDED_BY]->(@V PolicyVendor:*);RETURN X.name AS Policy, V.name AS Policy_Vendor, COUNT(I.name) AS InsuranceCount SORT_DESC InsuranceCount LIMIT 5

REST API execution time (ms) : 227 ms

Policy	Policy_Vendor	InsuranceCount
CH544526	4A47D772-5D43-4FB1-A334-B11F57C0DC91	2
CH179990	9AB9A2EC-45EA-49A4-BEA2-05F0B4DC862E	2
CH540849	2B37F69F-89D1-40E6-B47A-0FD5E3B6DBC5	2

Rows per page: 5 1–3 of 3

39. Find all persons who have taken loans in a given area (pincode, geo-loc, city, taluk, state)

S1=>(@P Person:*)-[@a APPLIES_FOR]->(@L Loan:*)-[@f HAS_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Bihar);RETURN P.name AS Customer, Z.name AS Pincode, s.District AS District, S.name AS State WHERE Pincode = "847429"

REST API execution time (ms) : 269 ms

District	Customer	Pincode	State
Darbhanga	f54e5599-f0ad-4f91-a11d-27abab7235a2	847429	Bihar
Darbhanga	f54e5599-f0ad-4f91-a11d-27abab7235a2	847429	Bihar

Rows per page: 5 1–2 of 2

40. Find all persons who similar to persons who have taken loan in a given area

S1=>(@P1 Person:*)-[@k Cluster1 {_cluster_=$$Person:84c30b1a-3aba-4a31-825e-f5f9159fb2d9}]->(@P2 Person:*)-[@l APPLIES_FOR]->(@L Loan:*)-[@f HAS_PINCODE]->(@Z Pincode:*);RETURN P2.name AS Customer,Z.name AS Pincode WHERE Pincode = "741161"

REST API execution time (ms) : 238 ms

41. Top K banks who have offered loans to customers in a given area (further ass filter like, loan amount more than X etc.) For a given location

S2=>[S1=>(@L Loan)-[@f1 FROM_INSTITUTION]->(@B FinancialInstitution:*)]-[@f2 HAS_PINCODE]->(@Z Pincode:*)-[@s IS_IN_STATE]->(@S State:Assam);RETURN B.name AS BankName, COUNT(L.name) AS Loan_Applications, AVG(L.LoanAmount) AS LoanAmount, Z.name AS Pincode, s.District AS District, S.name AS State WHERE Pincode = "781014" LIMIT 1

REST API execution time (ms) : 485 ms

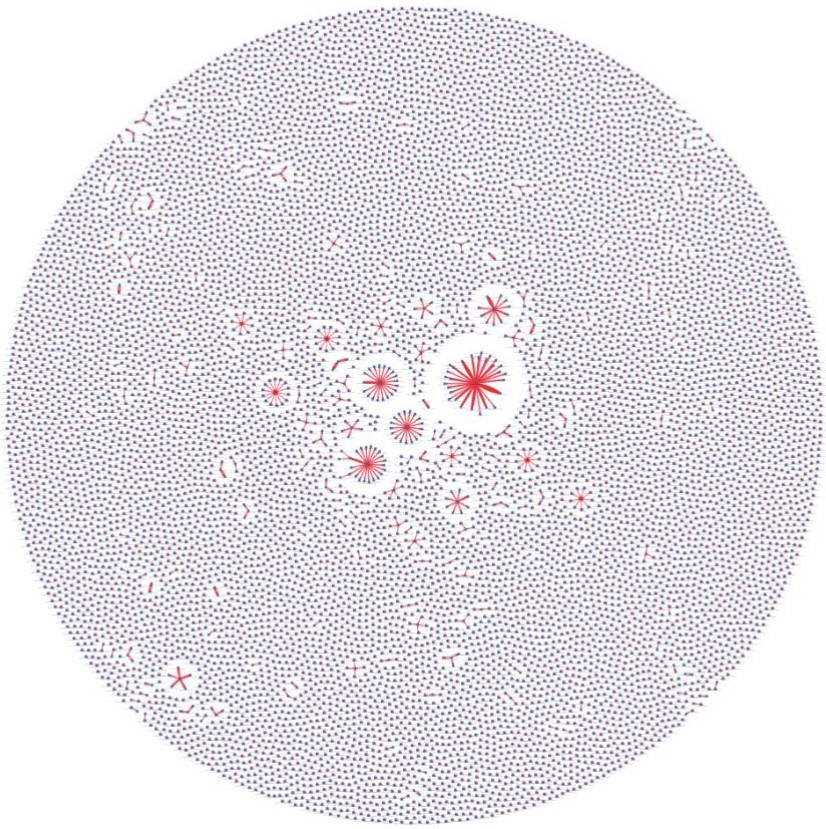
District	BankName	Pincode	State	Loan_Applications	LoanAmount
Kamrup Metropolitan	Fullerton India	781014	Assam	4	313400

Rows per page: 5 1–1 of 1

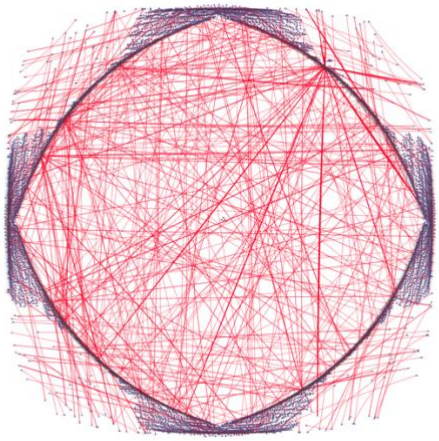
42. Look at how all persons are connected to all other nodes [full view]

CAUTION : Don’t run this query all the time, it’s fast but heavy to render. BangDB responds in 580 ms, but UI takes time to render everything

S1=>(@P Person:*)-[*]->(*)

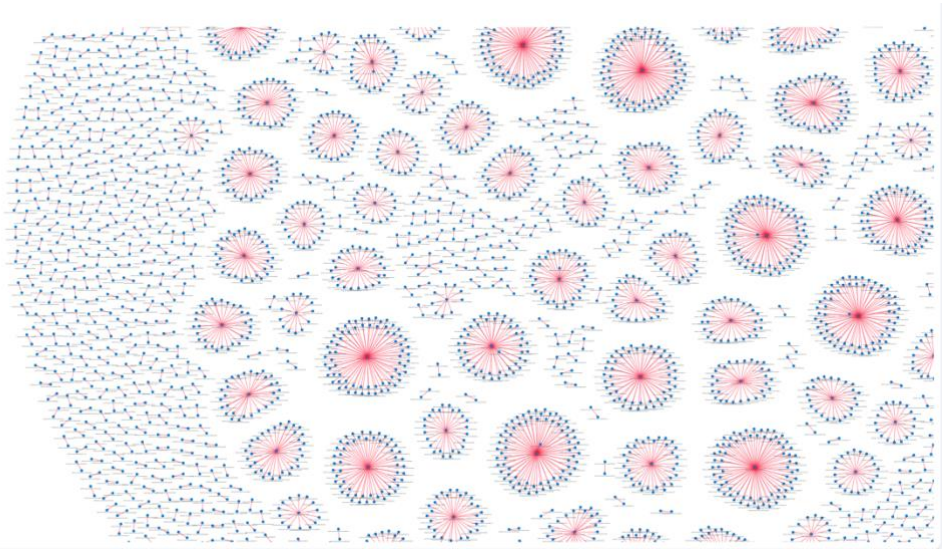


Zoom view of the inner small part



To see all transactions

S1=>(@P Person:*)-[PAYS_FOR]->(*)



These are many different varieties of queries. We can extend, add and edit many more as we require

Version history

V1.1

1. Person -> State, Person -> Pincode
2. Insurance -> State, Insurance -> Pincode [District as property for IS_IN_STATE from Insurance to State]
3. LoanApplicationID -> AddressID, LoanApplicationID -> State
4. HAS_LANKMARK to HAS_LOCALITY