DRAFT v1

Graph Generation of FHIR Resource

This Draft talks about an analytical prospective of FHIR resource using Graph Database. We think that a Property Graph could be a good solution to perform analytics because of its hierarchal nature and connection between each entities in a graph we call node. We believe if we could make a complete Graph Network out out the Raw JSON of FHIR then we can perform complex queries that will give use more information beyond the resource itself.

We choose a Graph Database to manage all the complexity of creating and managing the nodes and relationship between them. We found that Neo4J is a perfect fit for this choice and its being used by many corporate analytics companies including many Health-Care system providers.

So now we needed a way or proper tool-set to create Property Graph based on the FHIR Resource (JSON). Hence we are on an ongoing development to create a libraary that does the job , we call it "*JYPHER*".

JYPHER

JYPHER stands for JSON to Cypher conversion . Cypher is a well known Graph Query Language that is simplistic in nature and used in Neo4J Graph Database . JYPHER is in ongoing development with Golang. It is a knowledge oriented JSON to cypher query generator. It takes Raw JSON data as input then it uses a query processor for JSON data and produces a Cypher query based on the domain knowledge that generates nodes and vertex for the graph database.

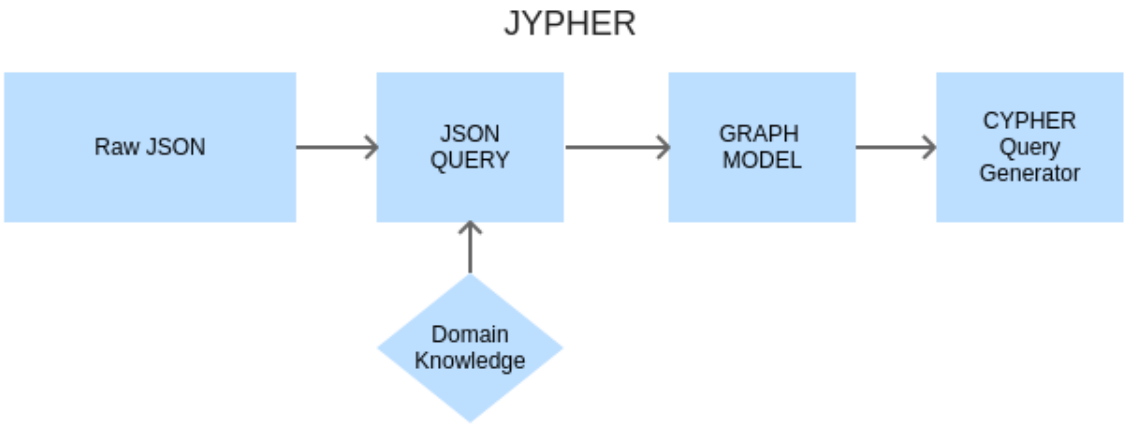

Fig 1 : Internal Structure of JYPHER

JYPHER library has 4 major components

1. JSON Query Parser

During the development of JYPHER we needed a way to perform complex queries on Raw JSON data. For this we have used JSON PATH (http://goessner.net/articles/JsonPath/index.html#e2) library to extract data out of FHIR Resource.

For instance, let's assume we need to know what is the name of the state that the person uses as official or his marital status etc from a complex JSON data like below.

Ultimately we need the value of multiple fields in a JSON data that could be nested to any level .

A syntax example of query would be, like $.*maritalStatus.text* in a JSON file means there exists a parent field or object called *maritalStatus* and it has child field or property called *text* . Look at the table below for a quick idea .

Below is a sample of FHIR Patient Resource in JSON


```
{
   "resourceType":"Patient",
   "id":"HJU345ER",
   "identifier":[
      {
         "use":"usual",
         "system":"https://neuron.care/system/identifier",
         "value":"HJU345ER"
      }
   ],
   "active":true,
   "name":[
      {
         "use":"usual",
         "family":"Babu",
         "given":[
            "Amin",
            "Rahman"
         ],
         "suffix":[
            "MSc"
         ]
      }
   ],
   "telecom":[
      {
         "system":"phone",
         "value":"0648352638",
         "use":"mobile"
      },
      {
         "system":"email",
         "value":"p.patient@gmail.com",
         "use":"home"
      }
   ],
   "gender":"male",
   "birthDate":"10-10-1993",
   "deceasedBoolean":false,
   "address":[
      {
         "line":[
```

```
            "123 Flat3 Hourse no 334"
         ],
         "city":"Gulshan",
         "state":"Dhaka",
         "postalCode":"1210",
         "country":"Bangladesh",
         "use":"official"
      }
   ],
   "maritalStatus":{
      "coding":[
         {
            "system":"http://hl7.org/fhir/v3/MaritalStatus",
            "code":"U",
            "display":"Single"
         }
      ],
      "text":"Never Married"
   },
   "multipleBirthBoolean":true,
   "communication":[
      {
         "language":{
            "coding":[
               {
                  "system" : "hl7.org/fhir/ValueSet/languages",
                  "code":"bn",
                  "display":"Bangla"
               },
               {
                  "system" : "hl7.org/fhir/ValueSet/languages",
                  "code":"en",
                  "display":"English"
               }
            ]
         },
         "preferred":true
      }
   ],
   "generalPractitioner": [
               {
                        "reference" : "Practitioner/ADH345AR"
               }
   ]
   "managingOrganization":{
      "reference":"Organization/4567894"
   }
}
```

Fig 2 : Patient-resource.json

Example Query Syntax And Their Results

Query	Result
$.address[*]	[Full Address Array]
$.address[*]?(@.use == \"official\").line	123 Flat3 Hourse no 334
$.maritalStatus.text	Never Married
$.communication[*].language.coding[*].code	bn en

2. Domain Knowledge

A Domain knowledge gives JYPHER the flexibility for defining what properties from a JSON data could be a graph node or property because we don't want to hard code the domain knowledge inside the Cypher library.

We introduced a domain knowledge in plain simple JSON format so that it could be changed at anytime without touching the code from inside. Its simple format is easily readable user will be able to understand what kind of nodes and relationship will be created if this particular domain knowledge is applied to JYPHER. Fig 3 shows a sample JSON file that is just a part of larger Domain Knowledge. This JSON data is used as a model to query Raw JSON file and then it generates a Graph Model based as shown in the JYPHER architecture in Fig 1.

```
"graph" : [
                {
                        "type" : "resource",
                        "id": "$.identifier[*]?(@.use == 'official').id@id",
                        "node": "patient",
                        "timestamp" : true,
                        "properties": [
                                "$.name[*]?(@.use == 'official').given@name",
                                "$.identifier[*]?(@.use ==
'usual').system@system",
                                "$.gender",
                                "$.birthDate",
                                "$.deceasedBoolean"
                        ]
                },
                {
                        "type": "address",
                        "id": "%internal.address.city.id",
                        "node": "city",
                        "properties": [
                                "$.address[*]?(@.country='US').city@name"
                        ],
                        "relation": [
                                {
                                        "towards" : "patient",
                                        "name" : "LIVES_IN",
                                        "direction" : "l"
```

```
                    }
                ]
            }
]
```

Fig 3 : Domain Knowledge of patient-resource.json called patient-domain.json

If we apply this domain knowledge to the JSON data that we will get this Graph below

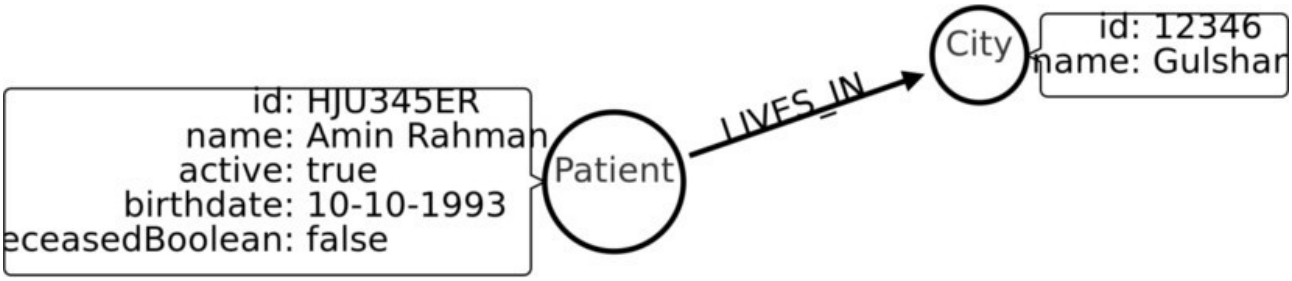

Fig 4 : Graph nodes based on Domain Knowledge

3. Graph Model Generation

Based on the Domain Knowledge JYPHER generates a common structure that contains information like how many nodes the query has to create with node name and its properties along with relationship information. This model is used as base to generate Cypher query.

4. Cypher Query Generation

This is the last step of JYPHER . The graph model is converted into a Cypher query and the Query then get executed in Neo4J Graph Database where it creates a Graph (Fig 5)

```
MERGE (patient:Patient {id:"HJU345ER"}) ON CREATE SET patient.system = "https://neuron.care/system/identifier" , patient.name = "Adam", patient.gender = "male" , patient.birthDate = "10-10-1993" MERGE (city:City {id:"12346"}) ON CREATE SET city.name = "LA" CREATE UNIQUE (city)<-[:LOCATED_IN]-(patient) MERGE (state:State {id:"52369"}) ON CREATE SET state.name = "NY" CREATE UNIQUE (state)<-[:SITUATED_IN]-(city) MERGE (organization:Organization {id:"4567894"}) CREATE UNIQUE (organization)<-[:ADMITTED_IN]-(patient)
```

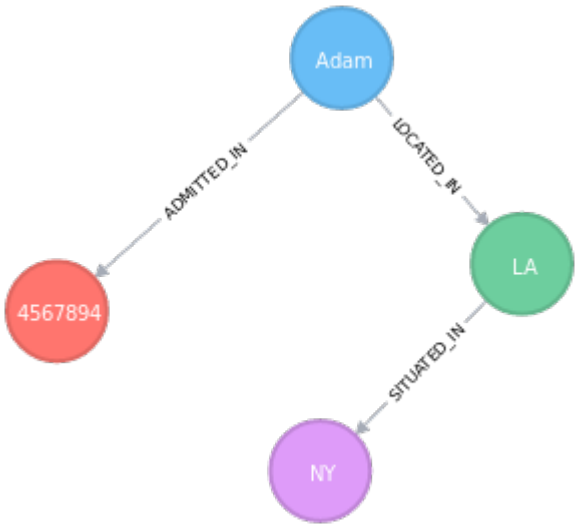

Fig 5 : Property Graph, based on the Sample Query Above

Complete Property Graph of Patient Resource

A Patient Resource has enough information like speaking language, address , relative/contract , gender information , the doctor that he visited and in which hospital he is admitted into. This information will give a Property Graph like below if proper domain Knowledge is applied.

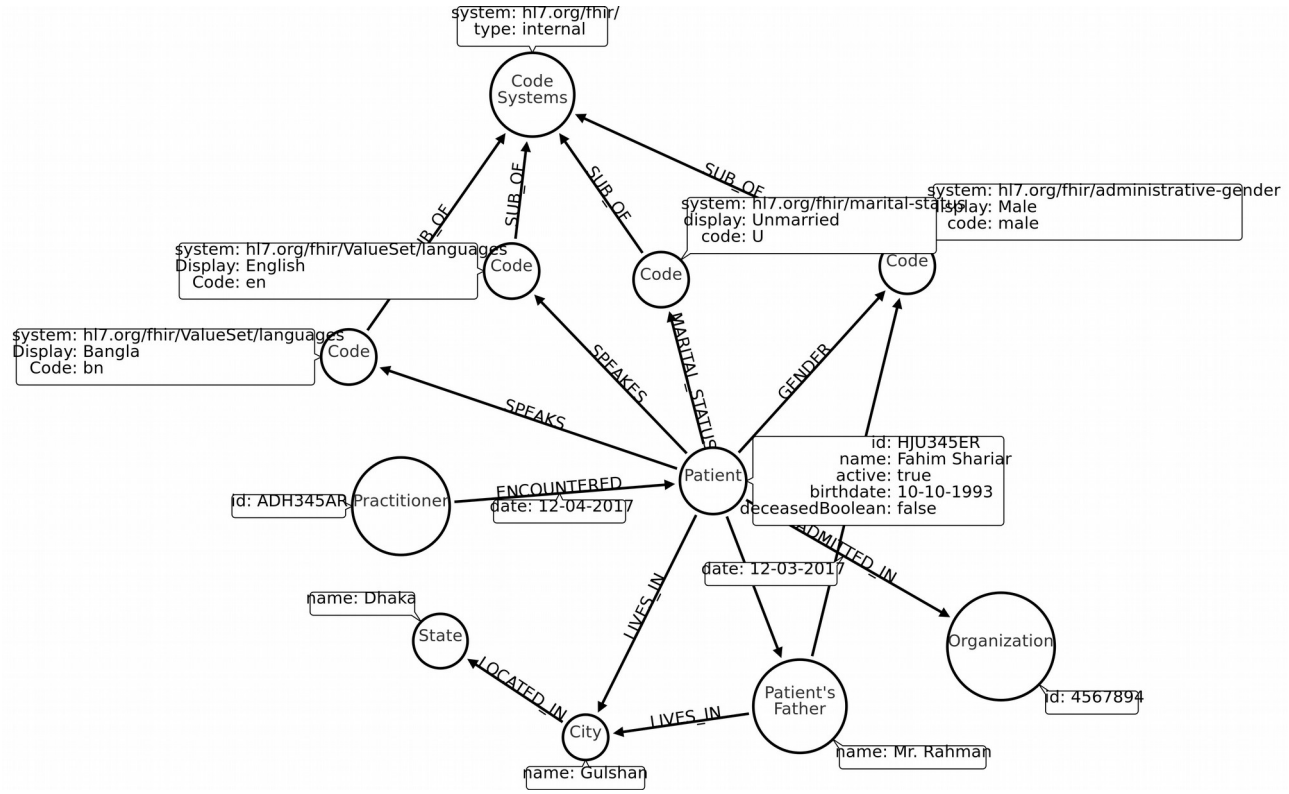

Fig 6 : Property Graph of Patient Resource

Extraction of Information

Our aim was to be able to do complex queries but for that we need huge amount of patient data and proper domain knowledge . However we can now traverse through the graph using Cypher query to get information. For instance , "*show me all the patient who is admitted into an organization whose id is 4567894 also show me the name of the cities they live in.*" . Corresponding Cypher Query will be

```
MATCH (o:Organization {id:'4567894'})<-[:ADMITTED_IN]-(n)
MATCH (n)-[:LOCATED_IN]->(c)
RETURN o, n.name ,c.name
```

Fig 7 : Cypher Query