Data Model Managment Handbook

Artur Mrozowski

January 8, 2019

Contents

1 Graph Data Model

1.1 The Choice of Graph Technology

1. **LPG-Labeled Property Graph** Neo4j had been chosen because it is a LPG, and as such, particularly appropriate for data modeling. It has also a large community, and we could benefit from it, by for example using APOC(Awesome Producers on Cypher) open source libraries. Community Edition is free and has able to accomodate our needs.

   With that said, LPGs receieved recently some traction, and it looks to become a defacto standard of graph technologies.

   In a recent post on Medium Looking Forward to 2019 in Graph Technologies author mentions the two major players in the arena. "Organizations like Neo4j and TigerGraph grew their marketshare in this area

and added a large number of new capabilities. New venture funding flowed into firms that are either on LPGs or quickly migrating to support LPGs. TigerGraph is noted for their native distributed graphs and Neo4j is noted for their innovation with products like Bloom."

TigerGraph is made for speed, and maybe not relevant for our data modeling use case. It may be well worth noting though that in their marketing campaign I quote:"TigerGraph offers the world's fastest graph analytics platform that tackles the toughest data challenges in real time, no matter how large or complex the data set. TigerGraph stores all data sources in a single, unified multiple-graph store that can scale up and out easily and efficiently to explore, discover and predict relationships. Unlike traditional graph databases, TigerGraph can scale for real-time multi-hop queries spanning trillions of relationships."

Above quoted post mentions also that: "Neo4j's Bloom product, although expensive on a per-user basis, is of interest because it is starting to blur the lines between graph visualization and natural language query processing. Many Bloom queries minimize the need for knowing Cypher and focus on allowing non-technical people to do complex analysis of graph databases." That could be potentially interesting even from data modeling point of view.

GQL, an initiative to create on standard property graph query language, is also interesting. Specially considering that TigerGrap and Neo4j are not the only players in the field.

```
Propety Graph vendors:
Amazon Neptune, Oracle PGX, Neo4j Server, SAP HANA Graph,
    AgensGraph (over PostgreSQL), Azure CosmosDB, Redis
   Graph, SQL Server 2017 Graph, Cypher for Apache Spark,
    Cypher for Gremlin, SQL Property Graph Querying,
   TigerGraph, Memgraph, JanusGraph, DSE Graph
```

2. **Synergy with AI** Google has recenlty released a paper called "Relational inductive biases, deep learning, and graph networks" with accompanying github repository. It will be interesting to follow that development.

1.2 Business and Solution Level

1. **3-level Data Architecture** Three level Data Architecture(conceptual, logical, physical) has been the standard for many years and made really good sense when designing application-agnostic SQL databases.

We will also in general follow this architecture, although one could argue if it really makes sense in NoSql schema design. NoSQL databases in general and GraphQL in particular are known to be application specific. One could argue that the logical step is not really necessary. See for example this discussion on DataVersity from Hackolade CEO Pascal Desmartes "Data Modeling is Dead... Long Live Schema Design!"

In our case the conceptual model is fixed, and comes from ACORD, in XMI format.So, the logical layer will mainly be one to one mapping, to the ACORD model. It could also be seen, as an operational subset of ACORD model, e.g our POC.

2. **Prerequisites and Software** ACORD is documented as UML in XMI format. We will be using Neo4j and external java libraries to load and work with the model.

   Install:

   ```
   Neo4j version 3.4.11
   APOC  apoc-3.4.0.4-all.jar.
   ```

   Neo4j installation is pretty straight forward. You need to place APOC jar file in the plugins directory of the Neo4j home and add these lines to the config file.

   ```
   dbms.security.procedures.unrestricted=apoc.*
   apoc.export.file.enabled=true
   apoc.import.file.enabled=true
   ```

   And uncomment this line

   ```
   dbms.directories.plugins=plugins
   ```

3. **Load XML** All you need to do to load XML files into the database is to call APOC procedure and provide url to the file:

   ```
   call apoc.xml.import('file:///tmp/ACORD.xml');
   ```

   Now we can see how the XMI document is structured.

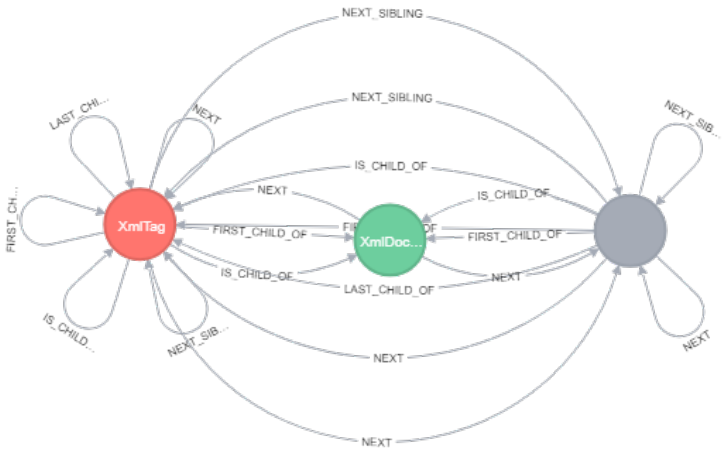

Figure 1: XML Structure

That document contains a lot of information, and of all it is captured in three basic nodes XmlTag, XmlDocument and XmlWord. But we are only interested in the business concepts. So, we need to extract business entities.

```
match(n) where (n.type = "uml:DataType" and n._name = "
    packagedElement" ) or
(n.type="uml:Class" and n._name = "packagedElement" )
set n:POC1Type return n;
```

Here we name business object types as POC1Type, and in reality we'd already here subset the number of objects to extract. Perhaps, by defining domain boundaries. That can be easily accomplished by adding a where clause. The blue nodes are entities and the red ones are their properties.

Figure 2: Business Entities

Next thing, is of course to see how these business objects relate to each other. UML model does contain relationships, but these are concerned with XMI structure. So, we need to extract and label business relationships. We are about to create our logical model.

1.3 Logical Model

1. **Creating Logical Model** Creating logical model consists of series of steps, depending on what we wish to include. Normally we'd like to include:

```
relations and their cardinality
properties
generalizations
subtype/supertype relations
data types
functional dependencies between properties and their
    identifying object types
lineage between XMI objects and logical objects.
```

Allthough the process is somewhat tedious, it needs to be done only once. Then it can be captured in a script/procedure and executed many times over.

Again, we are offered opportunity to subset the logical model or create several concurrent logical models. This command, will create logical model, with name Poc1LogiskModel, based on earlier defined POC1Type business object types.

```
MATCH
(o:POC1Type)
where o._name = "packagedElement"
and o.type <> "uml:Enumeration" and exists(o.name)
and o.id starts with "_fbb"
with o.name as ObjectName, o.id as FBB
merge (n:Poc1LogiskModel:POC1Type {ModelNavn: "LogModel 1
    ", Namn: ObjectName, FBB: FBB })
return count(ObjectName);
```

We won't be listing all the required steps here, refer to the Apendix for complete list. Once completed, it'll look like in Figure 3

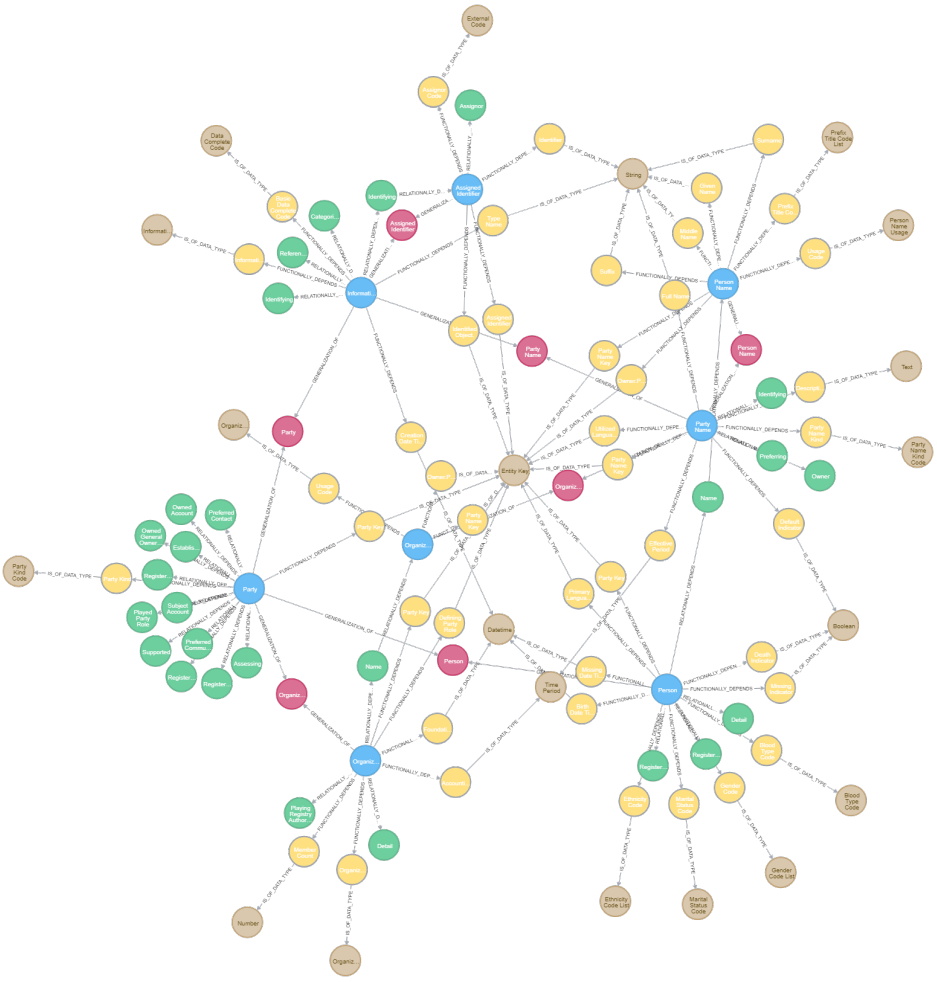

Figure 3: Logical Model

The blue nodes are our business entities, the green nodes are relation-

ships, the dark red nodes generalizations and yellow and light brown are properties and data types, respectively.

2. **Lineage** We can track the origin of an object by creating lineage. This can obviously be extended to capture lineage between all the layers and even data lineage.

```
match (g:POC1Type), (l:Poc1LogiskModel {ModelNavn: "
    LogModel 1"})
where g.id = l.FBB
with g.id as G_ID, l.FBB as L_ID
match (g:POC1Type {id: G_ID})
match (l:Poc1LogiskModel {ModelNavn: "LogModel 1", FBB:
    L_ID})
merge (g)-[:POC1_ORIGIN_OF]->(l);
```

And the result:

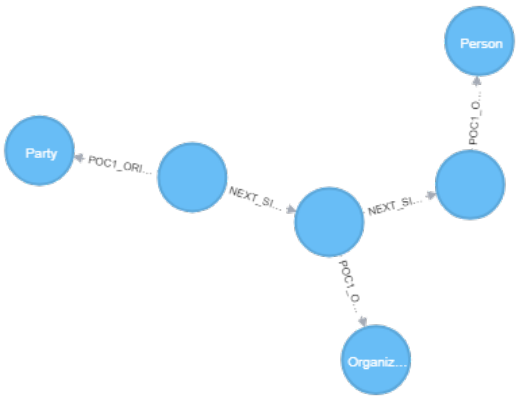

Figure 4: Lineage

We can see three XML tags of type UML Class and corresponding POC1NodeTypes.

1.4 Physical Level

1. **Creting physical model.** Same steps may be repeated, with some variations, in order to create the physical model. We can now iterate, create, and recreate physical model based on the logical model. We can also apply number of transformations(e.g. splitting, merging, versioning etc.)

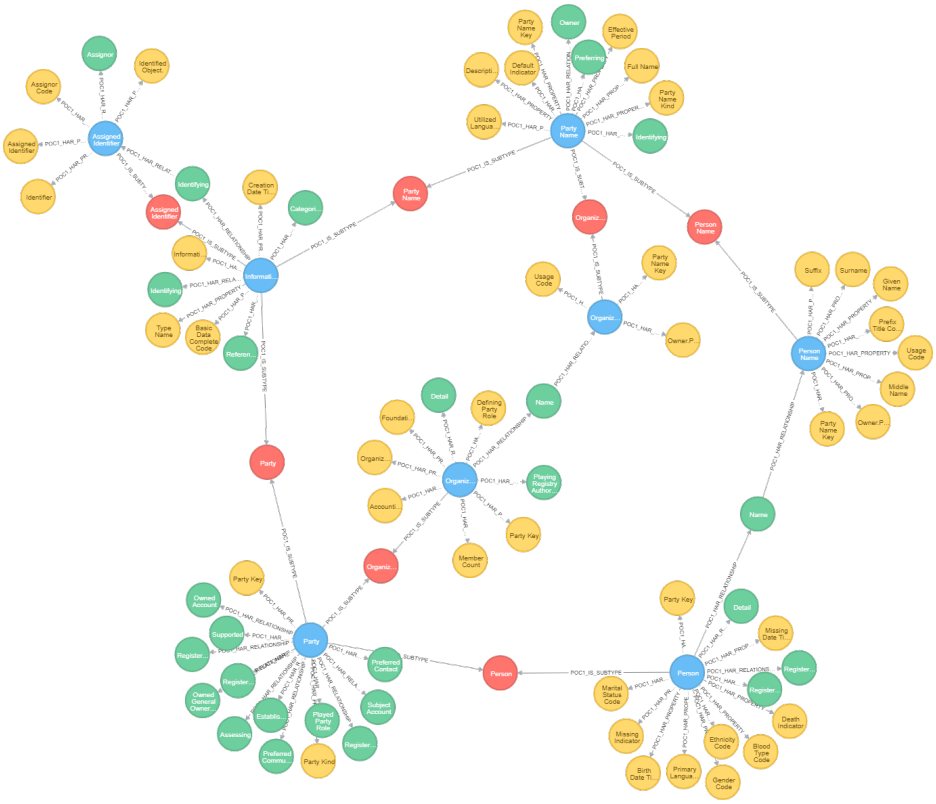

Figure 5: Physical Data Model

We can for example see that data types have been merged and became a properties of the relevant nodes.

2. **Transformations and refactoring.(Denormalization)** In this step we will apply a couple of transormations,for example denormalization, in the physical layer.

   Denormalization is the process of selectively violating normalization rules and reintroducing redundancy into the model and the database. In relational database it is done mainly due to performance. We can also denormalize to create a more user-friendly model. Let's see how this can be done in our graph data model.

   Rolling down. In this step we'd like to merge supertype node to its subtypes.

Figure 6: Party Name

From the Figure 6 we can see that both Person Name and Organization name are subtypes of Party Name. We want to roll down Party Name.

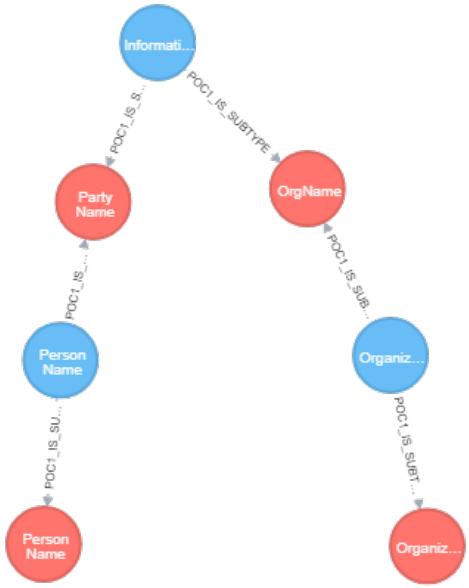

Figure 7: Party Name Rolling Down

We can see in Figure 7, that the business entity, Party Name, had been

merged into its subtypes. All the relations and properties of that node had been rolled down, see Figure 8.

Figure 8: Final result of Party Name roll down

Both Person Name and Organization Name have moved up in hierarchy, and both are now sub-types to the Information Model Object, the root object. Both entities have also inherited all properties and relations, creating now redundancy in the model.

In rolling down, we use APOC refactoring procedures, in this case mergeNodes. From the documentation, we can read that "These procedures help refactor the structure of your graph. This is helpful when you need to change your data model or for cleaning up data that was imported from an external source."

This is the code for that step , and as you can see we had to insert an additional generalization node, to avoid confusion, when running mergeNodes procedure.

```
MATCH (a2:Poc1NodeType)-[:POC1_IS_SUBTYPE]->(pX:
    Poc1FysiskGeneralization)
where
a2.Namn in ['OrgName','Party Name']
```

```
with collect(a2.Namn) as a2Namns
unwind a2Namns as x match(a3:Poc1NodeType)-[:
    POC1_IS_SUBTYPE]->(p1:Poc1FysiskGeneralization)<-[:
    POC1_IS_SUBTYPE]-(a4:Poc1NodeType)
where a3.Namn=x
and a4.Namn<>'Information Model Object'
with collect([a4,a3]) as nodes
unwind nodes as noder
CALL apoc.refactor.mergeNodes(noder,{properties:"
    override", mergeRels:true})
yield node MATCH (n)-[r:POC1_IS_SUBTYPE]->(c) return *
     ;
```

3. **Rolling up.** Rolling up, can be achieved using the script, we used for rolling down. If you are really lazy, you could just shuffle the a4 and a3 nodes like this $with collect([a3, a4]) as nodes$, and a roll up would then take place, instead. We need, of course, plug in suitable names of the entities .

2 GraphQL Schema

2.1 Object Graph Mapper(OGM)

1. **What is OGM?**

   An OGM (Object Graph Mapper) maps nodes and relationships in the graph to objects and references in a domain model. Object instances are mapped to nodes while object references are mapped using relationships, or serialized to properties (e.g. references to a Date). JVM primitives are mapped to node or relationship properties. An OGM abstracts the database and provides a convenient way to persist your domain model in the graph and query it without using low level drivers. It also provides the flexibility to the developer to supply custom queries where the queries generated by the OGM are insufficient.

   We will be using OGM to generate annotated GraphQL schema as a next step. Code in GitLab

2.2 GraphQL Schema Generation

1. **Annotated GraphQL Schema.**
   With help of OGM and simple Java program we will translate our graph

data model to annotated GraphQL schema. AWS Amplify(model tranform library) takes then that annotated GraphQL schema and transforms it to a real GraphQL schema.

Here is an example annotated shcema of Person Name and Organization Name, generated by our Java program.

```
type PersonName   @model
{

partyNameKind: String
partyNameKey: String
description: String
effectivePeriod: String
defaultIndicator: String
fullName: String
utilizedLanguageLanguageKey: String
ownerPartyKey: String
suffix: String
middleName: String
givenName: String
prefixTitleCode: String
surname: String
usageCode: String
person: Person @connection(name:"
    PersonPersonNameConnection")
}
type OrganizationName   @model
{

fullName: String
utilizedLanguageLanguageKey: String
effectivePeriod: String
defaultIndicator: String
partyNameKey: String
description: String
partyNameKind: String
organization: Organization @connection(name:"
    OrganizationOrganizationNameConnection")
}
```

What happens when you run it through the model transforms library (and deploy it via the AWS Amplify CLI) is that this is expanded. You get a CRUD API with paging and filtering built-in; input types are created for each of the mutations and subscriptions are set up on each mutation; the backing store in a DynamoDB database.

The @*model* directive stores the data in a table dedicated to the type within Amazon DynamoDB.

Generation of the API is described in the next section.

3 GraphQL API

3.1 Creating GraphQL API

Creating a functional GraphQL API is hard. You have to create a GraphQL schema, decide on authentication and database structures, implement the schema in a GraphQL service, wire up the authentication, hook up the database sources, ensure the whole thing is scalable, worry about logging and monitoring, and then write your app.

1. **AWS Amplify and AppSync. AppSync** helps you with everything except the GraphQL schema and the app. **Amplify** helps not only with the GraphQL schema but also wiht deployement of it. Technically, the schema is embedded in an Amazon CloudFormation template that can be deployed on AWS.

   AWS Amplify CLI will deploy all the resources necessary to implement the GraphQL schema on AWS AppSync and wire them up with appropriate VTL mapping templates to properly do the operations.

   It takes just a few commands to deploy our GraphQL schema.

   ```
   amplify init
   amplify add api
   amplify push
   ```

   The last *push* command is acutally deploying to AWS. Until then, the schema, resolvers, and template reside on the local machine. The detailed output from command line can be viewed in Appendix D.

   The entire API can be visualized using GraphQL Voyager.

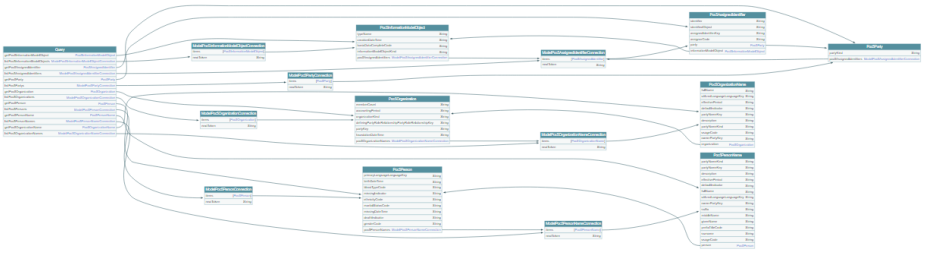

Figure 9: CRUD API in Graph Voyager

We won't dive into technical details of Amplify and AppSync, since there is some excelent documentation provided by AWS. Amplifyquic-start AppSync quickstart

A

Appendix A- Creating logical model.

1. **Load XML**

```
match(n) detach delete n;
call apoc.xml.import('file:///tmp/ACORD.xml');
match(n) where (n.type = "uml:DataType" and n.
    _name = "packagedElement" and n.name in ['
    Person','Person Name','Party','Organization','
    Organization Name','Assigned Identifier','
    Information Model Object','Party Name']) or
(n.type="uml:Class" and n._name = "
    packagedElement" and n.name in ['Person','
    Person Name','Party','Organization','
    Organization Name','Assigned Identifier','
    Information Model Object','Party Name'])
set n:POC1Type return n;
```

2. **Create logical model**

```
match(n:Poc1LogiskModel) detach delete n;
MATCH
(o:POC1Type)
where o._name = "packagedElement"
and o.type <> "uml:Enumeration" and exists(o.name
    )
```

```
and o.id starts with "_fbb"
with o.name as ObjectName, o.id as FBB
merge (n:Poc1LogiskModel:POC1Type {ModelNavn: "
    LogModel 1", Namn: ObjectName, FBB: FBB })
return count(ObjectName);
```

3. **Add properties**

```
MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)
where o._name = "ownedAttribute" and exists(e.
    name)
and o.type <> "uml:Enumeration" and exists(o.name
    )
and not exists(o.association)
and o.id starts with "_fbb"
with o.name as attribute, e.id as FBB_O, o.id as
    FBB, o.type as type
merge (n:Poc1LogiskModel:LogicalProperty {
    ModelNavn: "LogModel 1", Namn: attribute, FBB:
     FBB, FBB_O:FBB_O, type_id:type });
```

4. **Add data types.**

```
        //frame#frame#####frameDataTypesAsRelations

match (e:LogicalProperty)
match (o:XmlTag)
where (e.type_id=o.id)
with o.name as typeName, o.type as umlType, o.id
    as type_id
merge (n:Poc1LogiskModel:LogicalDataType {
    ModelNavn: "LogModel 1", Namn: typeName, FBB:
    type_id, umlType:umlType});

match (e:LogicalProperty)
match(o:LogicalDataType)
where (e.type_id=o.FBB)
with e.Namn as PropertyNamn, o.Namn as
    DataTypeNamn, e.FBB as FBB_P, o.FBB as FBB_T
match(e:LogicalProperty {ModelNavn: "LogModel 1",
     Namn: PropertyNamn, FBB: FBB_P})
match(o:LogicalDataType {ModelNavn: "LogModel 1",
     Namn: DataTypeNamn, FBB: FBB_T})
merge (e)-[:IS_OF_DATA_TYPE]->(o);
```

5. **Add relations, cardinality and genralizations.**

```
//******with associations + cardinality


MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)<-[r1:
    LAST_CHILD_OF]-(t:XmlTag)
where o._name = "ownedAttribute" and exists(e.
    name)
and o.type <> "uml:Enumeration" and exists(o.name
    )
and  exists(o.association)
and o.id starts with "_fbb"
and t._name="upperValue"
with o.name as attribute, e.id as FBB_O, o.id as
    FBB, o.association as association, o.type as
    type, t.value as value_t
//match(o:LogicalProperty {ModelNavn: "LogModel 1
    ", Namn: attribute, FBB: FBB})
merge (n:Poc1LogiskModel:LogicalRelation {
    ModelNavn: "LogModel 1", Namn: attribute, FBB:
     FBB, FBB_O:FBB_O, type: type, association:
    association, cardinality: value_t });


//***************with generalizations


MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)
where o._name= "generalization" and exists(e.name
    )
and o.type="uml:Generalization" and exists(o.
    general)
and o.id starts with "_fbb"
with e.name as attribute, o.id as FBB, e.id as
    FBB_O, o.general as genOf, o.type as type
merge(n:Poc1LogiskModel:LogicalGeneralization {
    ModelNavn: "LogModel 1", Namn: attribute, FBB:
     FBB, FBB_O:FBB_O, general:genOf, type:type })
    ;




MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)
where o._name = "ownedAttribute" and exists(e.
    name)
and o.type <> "uml:Enumeration" and exists(o.name
```

```
    )
and not exists(o.association)
and o.id starts with "_fbb"
with o.name as PropertyName, o.id as FBB_P, e.
    name as ObjectName, e.id as FBB_O
match (o:LogicalProperty {ModelNavn: "LogModel 1"
    , Namn: PropertyName, FBB: FBB_P})
match (e:POC1Type {ModelNavn: "LogModel 1", Namn:
     ObjectName, FBB: FBB_O})

merge (e)-[:FUNCTIONALLY_DEPENDS]->(o);



//ER relationships + cardinality


MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)
where o._name = "ownedAttribute" and exists(e.
    name)
and o.type <> "uml:Enumeration" and exists(o.name
    )
and  exists(o.association)
and o.id starts with "_fbb"
with o.name as PropertyName, o.id as FBB_P, e.
    name as ObjectName, e.id as FBB_O
match (o:LogicalRelation {ModelNavn: "LogModel 1"
    , Namn: PropertyName, FBB: FBB_P})
match (e:POC1Type {ModelNavn: "LogModel 1", Namn:
     ObjectName, FBB: FBB_O})
merge (e)-[:RELATIONALLY_DEPENDS]->(o);



match (o:LogicalRelation)
match (p:POC1Type)
where (o.type=p.FBB)
merge (o)-[:RELATIONALLY_DEPENDS]->(p);


// Subtype supertype relationships
MATCH
(e:POC1Type)<-[r:IS_CHILD_OF]-(o:XmlTag)
where o._name= "generalization" and exists(e.name
    )
and o.type="uml:Generalization" and exists(o.
    general)
```

```
and o.id starts with "_fbb"
with o.id as FBB_P, e.name as ObjectName, e.id as
    FBB_O
match (o:LogicalGeneralization {ModelNavn: "
   LogModel 1", FBB: FBB_P})
match (e:POC1Type {ModelNavn: "LogModel 1", Namn:
    ObjectName, FBB: FBB_O})
merge (e)-[:GENERALIZATION_OF]->(o);


match(o:LogicalGeneralization)
match(p:POC1Type)
where o.general=p.FBB
merge (o)<-[:GENERALIZATION_OF]-(p);
```

B Appendix B- Creating physical model.

Text of Appendix B is Here

C Appendix C- GraphQL Schema.

```
type Mutation {
createPoc5InformationModelObject(input:
   CreatePoc5InformationModelObjectInput!):
   Poc5InformationModelObject
updatePoc5InformationModelObject(input:
   UpdatePoc5InformationModelObjectInput!):
   Poc5InformationModelObject
deletePoc5InformationModelObject(input:
   DeletePoc5InformationModelObjectInput!):
   Poc5InformationModelObject
createPoc5AssignedIdentifier(input:
   CreatePoc5AssignedIdentifierInput!):
   Poc5AssignedIdentifier
updatePoc5AssignedIdentifier(input:
   UpdatePoc5AssignedIdentifierInput!):
   Poc5AssignedIdentifier
deletePoc5AssignedIdentifier(input:
   DeletePoc5AssignedIdentifierInput!):
   Poc5AssignedIdentifier
createPoc5Party(input: CreatePoc5PartyInput!): Poc5Party
updatePoc5Party(input: UpdatePoc5PartyInput!): Poc5Party
deletePoc5Party(input: DeletePoc5PartyInput!): Poc5Party
```

```
createPoc5Organization(input: CreatePoc5OrganizationInput!)
    : Poc5Organization
updatePoc5Organization(input: UpdatePoc5OrganizationInput!)
    : Poc5Organization
deletePoc5Organization(input: DeletePoc5OrganizationInput!)
    : Poc5Organization
createPoc5Person(input: CreatePoc5PersonInput!): Poc5Person
updatePoc5Person(input: UpdatePoc5PersonInput!): Poc5Person
deletePoc5Person(input: DeletePoc5PersonInput!): Poc5Person
createPoc5PersonName(input: CreatePoc5PersonNameInput!):
    Poc5PersonName
updatePoc5PersonName(input: UpdatePoc5PersonNameInput!):
    Poc5PersonName
deletePoc5PersonName(input: DeletePoc5PersonNameInput!):
    Poc5PersonName
createPoc5OrganizationName(input:
    CreatePoc5OrganizationNameInput!): Poc5OrganizationName
updatePoc5OrganizationName(input:
    UpdatePoc5OrganizationNameInput!): Poc5OrganizationName
deletePoc5OrganizationName(input:
    DeletePoc5OrganizationNameInput!): Poc5OrganizationName
}

type Query {
getPoc5InformationModelObject(id: ID!):
    Poc5InformationModelObject
listPoc5InformationModelObjects(filter:
    ModelPoc5InformationModelObjectFilterInput, limit: Int,
     nextToken: String):
    ModelPoc5InformationModelObjectConnection
getPoc5AssignedIdentifier(id: ID!): Poc5AssignedIdentifier
listPoc5AssignedIdentifiers(filter:
    ModelPoc5AssignedIdentifierFilterInput, limit: Int,
    nextToken: String):
    ModelPoc5AssignedIdentifierConnection
getPoc5Party(id: ID!): Poc5Party
listPoc5Partys(filter: ModelPoc5PartyFilterInput, limit:
    Int, nextToken: String): ModelPoc5PartyConnection
getPoc5Organization(id: ID!): Poc5Organization
listPoc5Organizations(filter:
    ModelPoc5OrganizationFilterInput, limit: Int, nextToken
    : String): ModelPoc5OrganizationConnection
getPoc5Person(id: ID!): Poc5Person
listPoc5Persons(filter: ModelPoc5PersonFilterInput, limit:
    Int, nextToken: String): ModelPoc5PersonConnection
getPoc5PersonName(id: ID!): Poc5PersonName
listPoc5PersonNames(filter: ModelPoc5PersonNameFilterInput,
     limit: Int, nextToken: String):
    ModelPoc5PersonNameConnection
```

```
getPoc5OrganizationName(id: ID!): Poc5OrganizationName
listPoc5OrganizationNames(filter:
    ModelPoc5OrganizationNameFilterInput, limit: Int,
    nextToken: String): ModelPoc5OrganizationNameConnection
}

type Subscription {
onCreatePoc5InformationModelObject:
    Poc5InformationModelObject @aws_subscribe(mutations: ["
    createPoc5InformationModelObject"])
onUpdatePoc5InformationModelObject:
    Poc5InformationModelObject @aws_subscribe(mutations: ["
    updatePoc5InformationModelObject"])
onDeletePoc5InformationModelObject:
    Poc5InformationModelObject @aws_subscribe(mutations: ["
    deletePoc5InformationModelObject"])
onCreatePoc5AssignedIdentifier: Poc5AssignedIdentifier
    @aws_subscribe(mutations: ["
    createPoc5AssignedIdentifier"])
onUpdatePoc5AssignedIdentifier: Poc5AssignedIdentifier
    @aws_subscribe(mutations: ["
    updatePoc5AssignedIdentifier"])
onDeletePoc5AssignedIdentifier: Poc5AssignedIdentifier
    @aws_subscribe(mutations: ["
    deletePoc5AssignedIdentifier"])
onCreatePoc5Party: Poc5Party @aws_subscribe(mutations: ["
    createPoc5Party"])
onUpdatePoc5Party: Poc5Party @aws_subscribe(mutations: ["
    updatePoc5Party"])
onDeletePoc5Party: Poc5Party @aws_subscribe(mutations: ["
    deletePoc5Party"])
onCreatePoc5Organization: Poc5Organization @aws_subscribe(
    mutations: ["createPoc5Organization"])
onUpdatePoc5Organization: Poc5Organization @aws_subscribe(
    mutations: ["updatePoc5Organization"])
onDeletePoc5Organization: Poc5Organization @aws_subscribe(
    mutations: ["deletePoc5Organization"])
onCreatePoc5Person: Poc5Person @aws_subscribe(mutations: ["
    createPoc5Person"])
onUpdatePoc5Person: Poc5Person @aws_subscribe(mutations: ["
    updatePoc5Person"])
onDeletePoc5Person: Poc5Person @aws_subscribe(mutations: ["
    deletePoc5Person"])
onCreatePoc5PersonName: Poc5PersonName @aws_subscribe(
    mutations: ["createPoc5PersonName"])
onUpdatePoc5PersonName: Poc5PersonName @aws_subscribe(
    mutations: ["updatePoc5PersonName"])
onDeletePoc5PersonName: Poc5PersonName @aws_subscribe(
    mutations: ["deletePoc5PersonName"])
```

```
onCreatePoc5OrganizationName: Poc5OrganizationName
    @aws_subscribe(mutations: ["createPoc5OrganizationName"
    ])
onUpdatePoc5OrganizationName: Poc5OrganizationName
    @aws_subscribe(mutations: ["updatePoc5OrganizationName"
    ])
onDeletePoc5OrganizationName: Poc5OrganizationName
    @aws_subscribe(mutations: ["deletePoc5OrganizationName"
    ])
}

type Poc5InformationModelObject {
typeName: String
creationDateTime: String
basicDataCompleteCode: String
informationModelObjectKind: String
poc5AssignedIdentifiers(filter:
    ModelPoc5AssignedIdentifierFilterInput, sortDirection:
    ModelSortDirection, limit: Int, nextToken: String):
    ModelPoc5AssignedIdentifierConnection
}

enum ModelSortDirection {
ASC
DESC
}

type ModelPoc5InformationModelObjectConnection {
items: [Poc5InformationModelObject]
nextToken: String
}

input ModelStringFilterInput {
ne: String
eq: String
le: String
lt: String
ge: String
gt: String
contains: String
notContains: String
between: [String]
beginsWith: String
}

input ModelIDFilterInput {
ne: ID
eq: ID
le: ID
```

```
lt: ID
ge: ID
gt: ID
contains: ID
notContains: ID
between: [ID]
beginsWith: ID
}

input ModelIntFilterInput {
ne: Int
eq: Int
le: Int
lt: Int
ge: Int
gt: Int
contains: Int
notContains: Int
between: [Int]
}

input ModelFloatFilterInput {
ne: Float
eq: Float
le: Float
lt: Float
ge: Float
gt: Float
contains: Float
notContains: Float
between: [Float]
}

input ModelBooleanFilterInput {
ne: Boolean
eq: Boolean
}

input ModelPoc5InformationModelObjectFilterInput {
typeName: ModelStringFilterInput
creationDateTime: ModelStringFilterInput
basicDataCompleteCode: ModelStringFilterInput
informationModelObjectKind: ModelStringFilterInput
and: [ModelPoc5InformationModelObjectFilterInput]
or: [ModelPoc5InformationModelObjectFilterInput]
not: ModelPoc5InformationModelObjectFilterInput
}

input CreatePoc5InformationModelObjectInput {
```

```
typeName: String
creationDateTime: String
basicDataCompleteCode: String
informationModelObjectKind: String
}

input UpdatePoc5InformationModelObjectInput {
typeName: String
creationDateTime: String
basicDataCompleteCode: String
informationModelObjectKind: String
}

input DeletePoc5InformationModelObjectInput {
id: ID
}

type Poc5AssignedIdentifier {
identifier: String
identifiedObject: String
assignedIdentifierKey: String
assignorCode: String
party: Poc5Party
informationModelObject: Poc5InformationModelObject
}

type ModelPoc5AssignedIdentifierConnection {
items: [Poc5AssignedIdentifier]
nextToken: String
}

input ModelPoc5AssignedIdentifierFilterInput {
identifier: ModelStringFilterInput
identifiedObject: ModelStringFilterInput
assignedIdentifierKey: ModelStringFilterInput
assignorCode: ModelStringFilterInput
and: [ModelPoc5AssignedIdentifierFilterInput]
or: [ModelPoc5AssignedIdentifierFilterInput]
not: ModelPoc5AssignedIdentifierFilterInput
}

input CreatePoc5AssignedIdentifierInput {
identifier: String
identifiedObject: String
assignedIdentifierKey: String
assignorCode: String
poc5AssignedIdentifierPartyId: ID
poc5AssignedIdentifierInformationModelObjectId: ID
}
```

```
input UpdatePoc5AssignedIdentifierInput {
identifier: String
identifiedObject: String
assignedIdentifierKey: String
assignorCode: String
poc5AssignedIdentifierPartyId: ID
poc5AssignedIdentifierInformationModelObjectId: ID
}

input DeletePoc5AssignedIdentifierInput {
id: ID
}

type Poc5Party {
partyKind: String
poc5AssignedIdentifiers(filter:
    ModelPoc5AssignedIdentifierFilterInput, sortDirection:
    ModelSortDirection, limit: Int, nextToken: String):
    ModelPoc5AssignedIdentifierConnection
}

type ModelPoc5PartyConnection {
items: [Poc5Party]
nextToken: String
}

input ModelPoc5PartyFilterInput {
partyKind: ModelStringFilterInput
and: [ModelPoc5PartyFilterInput]
or: [ModelPoc5PartyFilterInput]
not: ModelPoc5PartyFilterInput
}

input CreatePoc5PartyInput {
partyKind: String
}

input UpdatePoc5PartyInput {
partyKind: String
}

input DeletePoc5PartyInput {
id: ID
}

type Poc5Organization {
memberCount: String
accountingPeriod: String
```

```
organizationKind: String
definingPartyRoleRelationshipPartyRoleRelationshipKey:
   String
partyKey: String
foundationDateTime: String
poc5OrganizationNames(filter:
   ModelPoc5OrganizationNameFilterInput, sortDirection:
   ModelSortDirection, limit: Int, nextToken: String):
   ModelPoc5OrganizationNameConnection
}

type ModelPoc5OrganizationConnection {
items: [Poc5Organization]
nextToken: String
}

input ModelPoc5OrganizationFilterInput {
memberCount: ModelStringFilterInput
accountingPeriod: ModelStringFilterInput
organizationKind: ModelStringFilterInput
definingPartyRoleRelationshipPartyRoleRelationshipKey:
   ModelStringFilterInput
partyKey: ModelStringFilterInput
foundationDateTime: ModelStringFilterInput
and: [ModelPoc5OrganizationFilterInput]
or: [ModelPoc5OrganizationFilterInput]
not: ModelPoc5OrganizationFilterInput
}

input CreatePoc5OrganizationInput {
memberCount: String
accountingPeriod: String
organizationKind: String
definingPartyRoleRelationshipPartyRoleRelationshipKey:
   String
partyKey: String
foundationDateTime: String
}

input UpdatePoc5OrganizationInput {
memberCount: String
accountingPeriod: String
organizationKind: String
definingPartyRoleRelationshipPartyRoleRelationshipKey:
   String
partyKey: String
foundationDateTime: String
}
```

```
input DeletePoc5OrganizationInput {
id: ID
}

type Poc5Person {
primaryLanguageLanguageKey: String
birthDateTime: String
bloodTypeCode: String
missingIndicator: String
ethnicityCode: String
maritalStatusCode: String
missingDateTime: String
deathIndicator: String
genderCode: String
poc5PersonNames(filter: ModelPoc5PersonNameFilterInput,
    sortDirection: ModelSortDirection, limit: Int,
    nextToken: String): ModelPoc5PersonNameConnection
}

type ModelPoc5PersonConnection {
items: [Poc5Person]
nextToken: String
}

input ModelPoc5PersonFilterInput {
primaryLanguageLanguageKey: ModelStringFilterInput
birthDateTime: ModelStringFilterInput
bloodTypeCode: ModelStringFilterInput
missingIndicator: ModelStringFilterInput
ethnicityCode: ModelStringFilterInput
maritalStatusCode: ModelStringFilterInput
missingDateTime: ModelStringFilterInput
deathIndicator: ModelStringFilterInput
genderCode: ModelStringFilterInput
and: [ModelPoc5PersonFilterInput]
or: [ModelPoc5PersonFilterInput]
not: ModelPoc5PersonFilterInput
}

input CreatePoc5PersonInput {
primaryLanguageLanguageKey: String
birthDateTime: String
bloodTypeCode: String
missingIndicator: String
ethnicityCode: String
maritalStatusCode: String
missingDateTime: String
deathIndicator: String
genderCode: String
```

```
}

input UpdatePoc5PersonInput {
primaryLanguageLanguageKey: String
birthDateTime: String
bloodTypeCode: String
missingIndicator: String
ethnicityCode: String
maritalStatusCode: String
missingDateTime: String
deathIndicator: String
genderCode: String
}

input DeletePoc5PersonInput {
id: ID
}

type Poc5PersonName {
partyNameKind: String
partyNameKey: String
description: String
effectivePeriod: String
defaultIndicator: String
fullName: String
utilizedLanguageLanguageKey: String
ownerPartyKey: String
suffix: String
middleName: String
givenName: String
prefixTitleCode: String
surname: String
usageCode: String
person: Poc5Person
}

type ModelPoc5PersonNameConnection {
items: [Poc5PersonName]
nextToken: String
}

input ModelPoc5PersonNameFilterInput {
partyNameKind: ModelStringFilterInput
partyNameKey: ModelStringFilterInput
description: ModelStringFilterInput
effectivePeriod: ModelStringFilterInput
defaultIndicator: ModelStringFilterInput
fullName: ModelStringFilterInput
utilizedLanguageLanguageKey: ModelStringFilterInput
```

```
ownerPartyKey: ModelStringFilterInput
suffix: ModelStringFilterInput
middleName: ModelStringFilterInput
givenName: ModelStringFilterInput
prefixTitleCode: ModelStringFilterInput
surname: ModelStringFilterInput
usageCode: ModelStringFilterInput
and: [ModelPoc5PersonNameFilterInput]
or: [ModelPoc5PersonNameFilterInput]
not: ModelPoc5PersonNameFilterInput
}

input CreatePoc5PersonNameInput {
partyNameKind: String
partyNameKey: String
description: String
effectivePeriod: String
defaultIndicator: String
fullName: String
utilizedLanguageLanguageKey: String
ownerPartyKey: String
suffix: String
middleName: String
givenName: String
prefixTitleCode: String
surname: String
usageCode: String
poc5PersonNamePersonId: ID
}

input UpdatePoc5PersonNameInput {
partyNameKind: String
partyNameKey: String
description: String
effectivePeriod: String
defaultIndicator: String
fullName: String
utilizedLanguageLanguageKey: String
ownerPartyKey: String
suffix: String
middleName: String
givenName: String
prefixTitleCode: String
surname: String
usageCode: String
poc5PersonNamePersonId: ID
}

input DeletePoc5PersonNameInput {
```

```
id: ID
}

type Poc5OrganizationName {
fullName: String
utilizedLanguageLanguageKey: String
effectivePeriod: String
defaultIndicator: String
partyNameKey: String
description: String
partyNameKind: String
usageCode: String
ownerPartyKey: String
organization: Poc5Organization
}

type ModelPoc5OrganizationNameConnection {
items: [Poc5OrganizationName]
nextToken: String
}

input ModelPoc5OrganizationNameFilterInput {
fullName: ModelStringFilterInput
utilizedLanguageLanguageKey: ModelStringFilterInput
effectivePeriod: ModelStringFilterInput
defaultIndicator: ModelStringFilterInput
partyNameKey: ModelStringFilterInput
description: ModelStringFilterInput
partyNameKind: ModelStringFilterInput
usageCode: ModelStringFilterInput
ownerPartyKey: ModelStringFilterInput
and: [ModelPoc5OrganizationNameFilterInput]
or: [ModelPoc5OrganizationNameFilterInput]
not: ModelPoc5OrganizationNameFilterInput
}

input CreatePoc5OrganizationNameInput {
fullName: String
utilizedLanguageLanguageKey: String
effectivePeriod: String
defaultIndicator: String
partyNameKey: String
description: String
partyNameKind: String
usageCode: String
ownerPartyKey: String
poc5OrganizationNameOrganizationId: ID
}
```

```
input UpdatePoc5OrganizationNameInput {
fullName: String
utilizedLanguageLanguageKey: String
effectivePeriod: String
defaultIndicator: String
partyNameKey: String
description: String
partyNameKind: String
usageCode: String
ownerPartyKey: String
poc5OrganizationNameOrganizationId: ID
}

input DeletePoc5OrganizationNameInput {
id: ID
}
```

D Apendix D - Amplify schema deployment.

```
C:\awsAmplify\bonus5>amplify init
Note: It is recommended to run this command from the root
   of your app directory
? Choose your default editor: Visual Studio Code
? Choose the type of app that you're building javascript
Please tell us about your project
? What javascript framework are you using none
? Source Directory Path:  src
? Distribution Directory Path: dist
? Build Command:  npm.cmd run-script build
? Start Command: npm.cmd run-script start
Using default provider awscloudformation

For more information on AWS Profiles, see:
https://docs.aws.amazon.com/cli/latest/userguide/cli-
   multiple-profiles.html

? Do you want to use an AWS profile? Yes
? Please choose the profile you want to use default
/ Initializing project in the cloud...

CREATE_IN_PROGRESS AuthRole              AWS::IAM::Role
               Tue Jan 08 2019 10:54:47 GMT+0100 (W.
   Europe Standard Time)
CREATE_IN_PROGRESS bonus5-20190108105433 AWS::
   CloudFormation::Stack Tue Jan 08 2019 10:54:37 GMT+0100
    (W. Europe Standard Time) User Initiated
```

```
CREATE_IN_PROGRESS DeploymentBucket      AWS::S3::Bucket
              Tue Jan 08 2019 10:54:40 GMT+0100 (W. Europe
    Standard Time)
CREATE_IN_PROGRESS DeploymentBucket      AWS::S3::Bucket
              Tue Jan 08 2019 10:54:41 GMT+0100 (W. Europe
    Standard Time) Resource creation Initiated
CREATE_IN_PROGRESS UnauthRole            AWS::IAM::Role
               Tue Jan 08 2019 10:54:42 GMT+0100 (W.
   Europe Standard Time)
CREATE_IN_PROGRESS UnauthRole            AWS::IAM::Role
               Tue Jan 08 2019 10:54:42 GMT+0100 (W.
   Europe Standard Time) Resource creation Initiated
CREATE_IN_PROGRESS AuthRole              AWS::IAM::Role
               Tue Jan 08 2019 10:54:48 GMT+0100 (W.
   Europe Standard Time) Resource creation Initiated
CREATE_COMPLETE    UnauthRole            AWS::IAM::Role
               Tue Jan 08 2019 10:54:56 GMT+0100 (W.
   Europe Standard Time)
CREATE_COMPLETE    DeploymentBucket      AWS::S3::Bucket
              Tue Jan 08 2019 10:55:01 GMT+0100 (W. Europe
    Standard Time)
CREATE_COMPLETE    AuthRole              AWS::IAM::Role
               Tue Jan 08 2019 10:55:02 GMT+0100 (W.
   Europe Standard Time)
CREATE_COMPLETE    bonus5-20190108105433 AWS::
   CloudFormation::Stack Tue Jan 08 2019 10:55:09 GMT+0100
    (W. Europe Standard Time)
 Successfully created initial AWS cloud resources for
    deployments.

Your project has been successfully initialized and
   connected to the cloud!

--------------------------------------------------------------------------------


C:\awsAmplify\bonus5>\amplify add api
? Please select from one of the below mentioned services
   GraphQL
? Provide API name: bonus5
? Choose an authorization type for the API API key
? Do you have an annotated GraphQL schema? Yes
? Provide your schema file path: schema.txt

GraphQL schema compiled successfully. Edit your schema at C
   :\awsAmplify\bonus5\amplify\backend/api/bonus5/schema.
   graphql
Successfully added resource bonus5 locally
```

```
Some next steps:
"amplify push" will build all your local backend resources
   and provision it in the cloud
"amplify publish" will build all your local backend and
   frontend resources (if you have hosting category added)
    and provision it in the cloud

-------------------------------------------------------------------------------


C:\awsAmplify\bonus5>amplify push
| Category | Resource name | Operation | Provider plugin
   |
| -------- | ------------- | --------- | -----------------
   |
| Api      | bonus5        | Create    | awscloudformation
   |
? Are you sure you want to continue? true

GraphQL schema compiled successfully. Edit your schema at C
   :\awsAmplify\bonus5\amplify\backend\api\bonus5/schema.
   graphql
? Do you want to generate code for your newly created
   GraphQL API (Y/n) y
? Do you want to generate code for your newly created
   GraphQL API Yes
? Enter the file name pattern of graphql queries, mutations
    and subscriptions (src\graphql\**\*.js)
? Enter the file name pattern of graphql queries, mutations
    and subscriptions src\graphql\**\*.js
? Choose the code generation language target (Use arrow
   keys)
? Choose the code generation language target typescript
? Enter the file name for the generated code (src\API.ts)
? Enter the file name for the generated code src\API.ts
? Do you want to generate/update all possible GraphQL
   operations - queries, mutations and subscriptions (Y/n)
    Y
? Do you want to generate/update all possible GraphQL
   operations - queries, mutations and subscriptions Yes
- Updating resources in the cloud. This may take a few
   minutes...

UPDATE_IN_PROGRESS bonus5-20190108105433 AWS::
   CloudFormation::Stack Tue Jan 08 2019 11:17:30 GMT+0100
    (W. Europe Standard Time) User Initiated
CREATE_IN_PROGRESS apibonus5              AWS::
   CloudFormation::Stack Tue Jan 08 2019 11:17:35 GMT+0100
    (W. Europe Standard Time)
```

```
CREATE_IN_PROGRESS apibonus5              AWS::
    CloudFormation::Stack Tue Jan 08 2019 11:17:35 GMT+0100
     (W. Europe Standard Time) Resource creation Initiated
- Updating resources in the cloud. This may take a few
    minutes...

CREATE_IN_PROGRESS bonus5-20190108105433-apibonus5-
    RJNJ6HQI268K AWS::CloudFormation::Stack Tue Jan 08 2019
     11:17:35 GMT+0100 (W. Europe Standard Time) User
    Initiated
- Updating resources in the cloud. This may take a few
    minutes...

CREATE_IN_PROGRESS GraphQLAPIKey                    AWS::
    AppSync::ApiKey        Tue Jan 08 2019 11:17:49 GMT
    +0100 (W. Europe Standard Time)
CREATE_IN_PROGRESS GraphQLAPI
...
...
...
CREATE_COMPLETE apibonus5 AWS::CloudFormation::Stack Tue
    Jan 08 2019 11:19:51 GMT+0100 (W. Europe Standard Time)
\ Updating resources in the cloud. This may take a few
    minutes...

UPDATE_COMPLETE_CLEANUP_IN_PROGRESS bonus5-20190108105433
    AWS::CloudFormation::Stack Tue Jan 08 2019 11:19:53 GMT
    +0100 (W. Europe Standard Time)
UPDATE_COMPLETE                       bonus5-20190108105433
    AWS::CloudFormation::Stack Tue Jan 08 2019 11:19:53 GMT
    +0100 (W. Europe Standard Time)
 Code generated successfully and saved in file src\API.ts
 Generated GraphQL operations successfully and saved at src
     \graphql
 All resources are updated in the cloud

GraphQL endpoint: https://5w4datknrnblxolovon62sqvtu.
    appsync-api.us-east-1.amazonaws.com/graphql
GraphQL API KEY: da2-njkwca5en5do7bjeuuuh45l34u
```