# 

# FIBO Data Alignment Exercises

In this exercise, we will be exploring three datasets.

1. GLEIF levels 1 and 2 (entity data and ownership data)
2. PermID (Thomson Reuters reference data)
3. Swap data in a spreadsheet

We will be working through a number of use cases that require the use of multiple data sets.

These data sets show many of the concepts in the lectures, to wit:

1. The GLEIF data has already been aligned to FIBO, and can be queried using FIBO terms.
2. The PermID data has not been aligned to FIBO, so we will show how to use it with FIBO data
3. The Swap data is not in RDF, not to mention not aligned with FIBO. We will import it and align it.

The use cases include:

1. Determining the ownership hierarchies; parents and ultimate parents of a registered entity.
2. Rolling up information about Swaps along these hierarchies (“what are the total amounts in swaps by a particularly ultimate parent organization?”)
3. Transitive exposure. Who are the parties I swap with, and who do they swap with, etc.?
4. Filter these answers by business area, jurisdiction, etc.

## Exploring GLEIF data

Let’s start by finding out what graphs exist in the database (query *whatGraphsExist*):

SELECT DISTINCT ?g

WHERE {

GRAPH ?g {?s ?p ?o}

}

The output from this query is:

urn:GLEIF

urn:PermID

Graph GLEIF contains FIBO information about legal entities (companies). PermID is a Thomson-Reuters database that contains analogous information about companies. The Work graph is where we will be putting the results of any CONSTRUCT queries that we perform in this exercise.

Let’s start by exploring GLEIF. The following query tells us what classes have instances in this graph (query *whatGleifClassesExist*):

SELECT DISTINCT ?class

WHERE {

GRAPH <urn:GLEIF> {

?s a ?class

}

}

The output from this query is:

owl:Ontology

fibo-fbc-fct-breg:RegistrationAddress

fibo-fbc-fct-breg:RegisteredAddress

fibo-fbc-fct-ra:RegistryIdentifier

fibo-be-le-lp:LegalEntity

fibo-be-le-lei:LegalEntityIdentifier

l2ont:UltimateOwnership

l2ont:DirectOwnership

l2ont:PrimaryForeignBranchOwnership

lcc-cr:Country

owl:Class

l2ont:AccountingSource

owl:NamedIndividual

owl:ObjectProperty

owl:DatatypeProperty

owl:Restriction

We are not particularly interested in the OWL classes since they appear in any OWL ontology. Of the remaining classes, the top-level “thing” is

fibo-be-le-lp:LegalEntity

So let’s see what properties an instance of LegalEntity can have in this database (query *whatGleifPropertiesOnLegalEntityExist*):

SELECT DISTINCT ?p

WHERE {

GRAPH <urn:GLEIF> {

?s a fibo-be-le-lp:LegalEntity .

?s ?p ?o .

}

}

The results are:

rdf:type

fibo-fnd-rel-rel:hasLegalName

fibo-be-le-lp:isOrganizedIn

fibo-be-le-lei:hasAddressOfLegalFormation

fibo-fbc-fct-breg:hasHeadquartersAddress

Again, we’re not particularly interested in the rdf:type property since it is ubiquitous. So, let’s find a few (say, 5) Legal Entities along with the values of these properties (query *propertyValuesOfLegalEntity*):

SELECT DISTINCT ?le ?legalName ?organizedIn ?legAddr ?hqAddr

WHERE {

GRAPH <urn:GLEIF> {

?le a fibo-be-le-lp:LegalEntity .

optional {?le fibo-fnd-rel-rel:hasLegalName ?legalName }

optional {?le fibo-be-le-lp:isOrganizedIn ?organizedIn}

optional {?le fibo-be-le-lei:hasAddressOfLegalFormation ?legAddr}

optional {?le fibo-fbc-fct-breg:hasHeadquartersAddress ?hqAddr}

}

}

ORDER BY ?le ?legalName ?organizedIn ?legAddr ?hqAddr

LIMIT 5

Here is what we get:

**le:**

l1ont:LE-001GPB6A9XPE8XJICC14

**legalName:**

Fidelity Advisor Series I - Fidelity Advisor Leveraged Company Stock Fund

**organizedIn:**

http://www.omg.org/spec/LCC/Countries/ISO3166-1-CountryCodes-Add/CTRY-US

**legAddr:**

l1ont:Leg-001GPB6A9XPE8XJICC14

**hqAddr:**

l1ont:HQ-001GPB6A9XPE8XJICC14

**le:**

l1ont:LE-004L5FPTUREIWK9T2N63

**legalName:**

Hutchin Hill Capital, LP

**organizedIn:**

http://www.omg.org/spec/LCC/Countries/ISO3166-1-CountryCodes-Add/CTRY-US

**legAddr:**

l1ont:Leg-004L5FPTUREIWK9T2N63

**hqAddr:**

l1ont:HQ-004L5FPTUREIWK9T2N63

**le:**

l1ont:LE-00EHHQ2ZHDCFXJCPCL46

**legalName:**

Vanguard Russell 1000 Growth Index Trust

**organizedIn:**

http://www.omg.org/spec/LCC/Countries/ISO3166-1-CountryCodes-Add/CTRY-US

**legAddr:**

l1ont:Leg-00EHHQ2ZHDCFXJCPCL46

**hqAddr:**

l1ont:HQ-00EHHQ2ZHDCFXJCPCL46

**le:**

l1ont:LE-00GBW0Z2GYIER7DHDS71

**legalName:**

Aristeia Capital, L.L.C.

**organizedIn:**

http://www.omg.org/spec/LCC/Countries/ISO3166-1-CountryCodes-Add/CTRY-US

**legAddr:**

l1ont:Leg-00GBW0Z2GYIER7DHDS71

**hqAddr:**

l1ont:HQ-00GBW0Z2GYIER7DHDS71

**le:**

l1ont:LE-00KLB2PFTM3060S2N216

**legalName:**

Harris Associates Investment Trust - Oakmark International Fund

**organizedIn:**

http://www.omg.org/spec/LCC/Countries/ISO3166-1-CountryCodes-Add/CTRY-US

**legAddr:**

l1ont:Leg-00KLB2PFTM3060S2N216

**hqAddr:**

l1ont:HQ-00KLB2PFTM3060S2N216

Most of this is not immediately meaningful, but we can understand the legal names, as well as the fact that all these companies are organized in the US. The final part of the “le” value is a Legal Entity Identifier (LEI). For example, the LEI of *Fidelity Advisor Series I - Fidelity Advisor Leveraged Company Stock Fund* is 001GPB6A9XPE8XJICC14.

The other property values refer to other objects in the database, and to interpret them we need to look at the properties of *those* object. For example, let’s look at what type of thing the address-of-legal-formation *http://www.adaptive.com/ontology/L1/Leg-001GPB6A9XPE8XJICC14* is, and what its properties are (query traversePropertyValuesOfLegalEntity):

SELECT ?property ?value

WHERE {

GRAPH <urn:GLEIF> {

l1ont:Leg-001GPB6A9XPE8XJICC14 ?property ?value

}

}

The results are:

rdf:type

fibo-fbc-fct-breg:RegistrationAddress

fibo-fbc-fct-breg:hasAddressLine1

C/O Eqiuty Portfolio Growth

fibo-fbc-fct-breg:hasCity

Boston

fibo-fbc-fct-breg:hasSubdivision

subdivCode:REG-US-MA

bureg:hasCountry

countryCode:CTRY-US

bureg:hasPostalCode

02109

So we see that it is a RegistrationAddress, and its properties are typical components of an address. Now let’s look at the headquarters address *http://www.adaptive.com/ontology/L1/HQ-001GPB6A9XPE8XJICC14* (query traversePropertyValuesOfLegalEntity2):

SELECT ?property ?value

WHERE {

GRAPH <urn:GLEIF> {

l1ont:HQ-001GPB6A9XPE8XJICC14 ?property ?value

}

}

The results are:

**rdf:type**

bureg:RegisteredAddress

**bureg:hasAddressLine1**

245 Summer Street

**bureg:hasCity**

Boston

**bureg:hasSubdivision**

subdivCode:REG-US-MA

**bureg:hasCountry**

countryCode:CTRY-US

**bureg:hasPostalCode**

02210

We see that this is an instance of RegisteredAddress (rather than RegistrationAddress, which was the type of the address-of-legal-formation). Again, its properties are typical components of an address.

Going back to the properties of any LegalEntity (query *whatGleifPropertiesOnLegalEntityExist*), we see that there is no property indicating when one company is a subsidiary of another, or, in the reverse direction, when one company owns another. There is, however, a class DirectOwnership (query *whatGleifClassesExist*), as well as one called UltimateOwnership. Let’s use this to create some a new property that express the ownership relation between a company and a subsidiary (query definitionOfParentOf):

CONSTRUCT {

?parent fw:parentOf ?child .

}

WHERE {

GRAPH <urn:GLEIF> {

?o a l2ont:DirectOwnership .

?o l2ont:hasOwner ?parent .

?o l2ont:hasSubsidiary ?child .

}

}

You won’t see the resulting triples in the lower panel. Instead, click on Export Results, choose trig for the type of file, and place them in a file called parents.trig. Open the file to see the resulting triples. For example:

<http://www.adaptive.com/ontology/L1/LE-3157002JBFAI478MD587> fw:parentOf <http://www.adaptive.com/ontology/L1/LE-097900BEK60000004860> , <http://www.adaptive.com/ontology/L1/LE-097900BEK60000005151> , <http://www.adaptive.com/ontology/L1/LE-097900BEK60000005248> , <http://www.adaptive.com/ontology/L1/LE-097900BEK50000004521> , <http://www.adaptive.com/ontology/L1/LE-097900BEK60000004957> .

This tells us that the company with LEI 3157002JBFAI478MD587 has five subsidiaries, with respective LEIs 097900BEK60000004860, 097900BEK60000005151, etc.

We can issue new queries to get the names of these companies, but a simpler approach would be to modify the query to construct this information right away (query parentChildNames):

SELECT ?parentName ?childName

WHERE {

GRAPH <urn:GLEIF> {

?o a l2ont:DirectOwnership .

?o l2ont:hasOwner ?parent .

?o l2ont:hasSubsidiary ?child .

?parent fibo-fnd-rel-rel:hasLegalName ?parentName .

?child fibo-fnd-rel-rel:hasLegalName ?childName .

}

}

For example, from the result (which we are transposing here from horizontal to vertical):

parent: l1ont:LE-3157002JBFAI478MD587

parentName: Tatra banka, a.s.

child: l1ont:LE-097900BEK60000004860

childName: Tatra-Leasing, s.r.o.

we see that the company with LEI 3157002JBFAI478MD587 is named *Tatra banka, a.s.*, and its subsidiary with LEI 097900BEK60000004860 is named *Tatra-Leasing, s.r.o.*

A CONSTRUCT query returns a set of triples, but it does not add them to the database. To add them, we can use INSERT rather than CONSTRUCT, with everything else the same (query insertParentOfTriples):

INSERT {

GRAPH <urn:work> {?parent fw:parentOf ?child}

}

WHERE {

GRAPH <urn:GLEIF> {

?o a l2ont:DirectOwnership .

?o l2ont:hasOwner ?parent .

?o l2ont:hasSubsidiary ?child .

}

}

Notice that we are adding the triples to the work graph rather than the GLEIF graph, which for purposes of this workshop we want to leave unchanged.

We can check what is in the work graph as follows (query contentsOfWorkGraph):

SELECT ?s ?p ?o

WHERE {

GRAPH <urn:work> {?s ?p ?o}

}

The results are a list of rows like this:

l1ont:LE-029200268F8M5YI5I629 fw:parentOf l1ont:LE-213800HXR6XZKC2G7U74

One of the classes in the GLEIF graph is UltimateOwnership, but we can use a query to generate a relation ultimateParentOf (query definitionOfUltimateParentOf):

INSERT {GRAPH <urn:work> {?anc fw:ultimateParentOf ?desc}}

#SELECT ?anc ?desc

WHERE {

GRAPH <urn:work> {

?anc fw:parentOf+ ?desc .

MINUS {?grandAnc fw:parentOf ?anc}

}

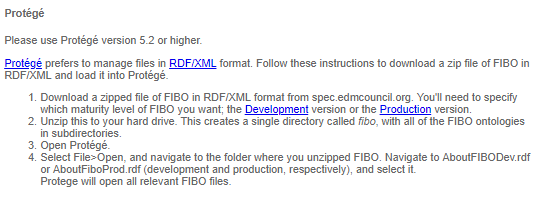
}

This defines the ultimate parent as something (?anc) that has a parentOf path to a descendant (?desc) but that has no parent itself. After executing this query, run the query contentsOfWorkGraph again to see the ultimateParentOf triples that were added.

## Importing spreadsheet data

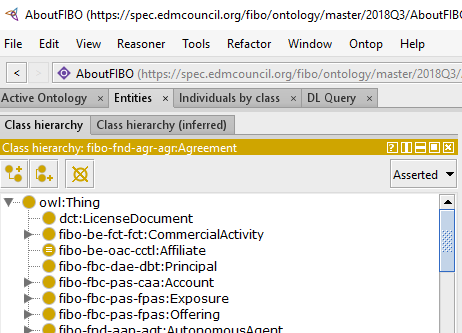
We are going to import some rather elaborate data about interest rate swaps. This data is based on an example from a real regulator who governs swaps in the US. We’re going to start by examining what FIBO has to say about Interest Rate Swaps.

1. Start by downloading the most recent release of FIBO from <https://spec.edmcouncil.org/static/ontology/> . We’re going to be using Protégé to examine FIBO, so follow the instructions for downloading FIBO for Protégé as shown here:



We’ll use the Production version of FIBO.

1. Unzip the archive .
2. Open Protégé. File>Open, navigate to where you unzipped the archive, and load AboutFIBOProd.rdf. This will load all of the current Production release of FIBO. When it is loaded, click on the “Entities” tab, and you should see something like this:



1. Click “Search” in the upper right hand corner of Protégé, and search for InterestRateSwap. Click on FixedFloatInterestRateSwap.

The instructor will guide you through the structure of a FixedFloatInterestRateSwap in FIBO.

We will import a spreadsheet as described in the lectures. The conceptual aspect of the mapping was described there. Here, we’ll see how to make this happen.

We’ll be using Stardog to import from the spreadsheet. Stardog provides a templating mechanism for importing data from a structured source. In brief, suppose a spreadsheet has column headers Fixed\_Rate\_Payer, Floating\_Rate\_Payer, Effective\_Date, and Termination\_Date. Then Stardog will expand expressions of the form

fibo-fnd-agr-ctr:hasEffectiveDate ex:ExplicitDate{Effective\_Date} ;

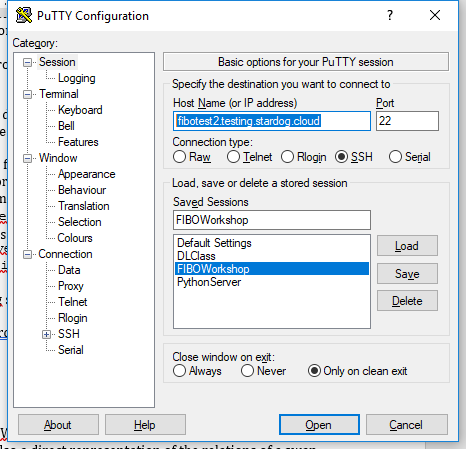
by filling in the value of Effective\_Date for each row in the spreadsheet.

Use putty to log in to the stardog server and we will examine the mapping file.

Host Name: fibotest2.testing.stardog.cloud

User Name: ubuntu

Password: fibo1



At the command prompt, type

cd /var/opt/stardog/Workshop/

less mappings.ttl

The instructor will guide you through the mappings file.

You can import the spreadsheet, using that mapping file, into stardog with the script provided:

Let’s create a relation ex:tradingWith between LEIs of the parties to a swap (query definitionOfTradingWith), and also a direct representation of the relations of a swap to its fixed rate payer and its floating rate payer and its notional amount:

INSERT

{

GRAPH <import:Swaps> {

?fixedCompanyU ex:tradingWith ?floatCompanyU .

?floatCompanyU ex:tradingWith ?fixedCompanyU .

?swap ex:hasFixedPayer ?fixedCompanyU .

?swap ex:hasFloatingPayer ?floatCompanyU .

?swap ex:hasSharedNotionalAmount ?amount

}

}

#SELECT distinct ?fixedCompanyU ?floatCompanyU

WHERE

{

GRAPH <import:Swaps> {

?swap a fibo-der-rtd-irswp:FixedFloatInterestRateSwap .

?swap fibo-der-drc-swp:hasLeg ?fixedLeg .

?fixedLeg a fibo-der-rtd-irswp:FixedInterestRateLeg .

?swap fibo-der-drc-swp:hasLeg ?floatLeg .

?floatLeg a fibo-der-rtd-irswp:FloatingInterestRateLeg .

?fixedLeg fibo-der-drc-swp:hasPayingParty ?fixedPayer .

?floatLeg fibo-der-drc-swp:hasPayingParty ?floatPayer .

?fixedLeg fibo-fnd-acc-cur:hasNotionalAmount ?fixedNotAmt .

?floatLeg fibo-fnd-acc-cur:hasNotionalAmount ?floatNotAmt .

?fixedNotAmt fibo-fnd-acc-cur:hasAmount ?amount .

?floatNotAmt fibo-fnd-acc-cur:hasAmount ?amount .

?fixedPayer fibo-fnd-rel-rel:hasIdentity ?fixedIdentity .

?fixedIdentity fibo-fnd-aap-agt:isIdentifiedBy ?fixedLEI .

?fixedLEI fibo-fnd-rel-rel:hasTag ?fixedCompany .

?floatPayer fibo-fnd-rel-rel:hasIdentity ?floatIdentity .

?floatIdentity fibo-fnd-aap-agt:isIdentifiedBy ?floatLEI .

?floatLEI fibo-fnd-rel-rel:hasTag ?floatCompany .

}

GRAPH <urn:GLEIF> {

?leidx fibo-fnd-rel-rel:hasTag ?fixedCompany ;

fibo-fnd-aap-agt:identifies ?fixedCompanyU .

?leidl fibo-fnd-rel-rel:hasTag ?floatCompany ;

fibo-fnd-aap-agt:identifies ?floatCompanyU .

}

}

Note that we have a commented-out SELECT, which we used prior to the INSERT, in order to verify that we were getting good results from the query. We can now verify the inserted triples by executing query whoIsTradingWithWhom:

SELECT ?a ?b

WHERE {

?a ex:tradingWith ?b

}

Verify the swap to payer relation through the query SwapPayers:

SELECT ?s ?fi ?fl ?amount

WHERE { GRAPH <import:Swaps> {

?s ex:hasFixedPayer ?fi .

?s ex:hasFloatingPayer ?fl .

?s ex:hasSharedNotionalAmount ?amount

}

}

Now let’s define “ultimate fixed rate payer” and “ultimate floating rate payer” by combining the payer information with information about ultimate parents (query definitionOfUltimatePayer):

INSERT {

GRAPH <import:Swaps> {

?swap ex:hasUltimateFixedPayer ?ultimateFixedPayer .

?swap ex:hasUltimateFloatingPayer ?ultimateFloatingPayer

}

}

#SELECT ?swap ?ultimateFixedPayer ?ultimateFloatingPayer

USING <urn:work>

USING <import:Swaps>

WHERE {

?swap ex:hasFixedPayer ?fixedPayer .

?swap ex:hasFloatingPayer ?floatingPayer .

OPTIONAL {?ultimateParentOfFixedPayer fw:ultimateParentOf ?fixedPayer}

BIND (COALESCE (?ultimateParentOfFixedPayer, ?fixedPayer)

AS ?ultimateFixedPayer) .

OPTIONAL {?ultimateParentOfFloatingPayer fw:ultimateParentOf ?floatingPayer}

BIND (COALESCE (?ultimateParentOfFloatingPayer, ?floatingPayer)

AS ?ultimateFloatingPayer) .

}

The reason we have to coalesce the ultimate parent of the fixed/floating payer with the direct fixed/floating payer is that the latter may not have a parent company, in which case it serves as the ultimate payer itself.

Verify the results through the query whoAreTheUltimatePayers:

SELECT ?s ?fi ?fl

WHERE { GRAPH <import:Swaps> {

?s ex:hasUltimateFixedPayer ?fi .

?s ex:hasUltimateFloatingPayer ?fl

}

}

Now let’s find the rollup amount for each ultimate payer over all the swaps that its subsidiaries participate in (query positionRollups):

SELECT ?ultimatePayer (SUM(?amounti) AS ?rollup)

FROM <import:Swaps>

WHERE

{

{

?swap ex:hasUltimateFixedPayer ?ultimatePayer

}

UNION

{

?swap ex:hasUltimateFloatingPayer ?ultimatePayer .

}

?swap ex:hasSharedNotionalAmount ?amount .

BIND (xsd:decimal(?amount) AS ?amounti) .

}

GROUP BY ?ultimatePayer

## Transitive Exposure

There are basically two kinds of risk in a swap: Market Risk, which is the risk that the market might go in a direction that is not favorable to your position, and counterparty risk, that your counterparty might default on their payments. In the case of Counterparty risk, it is possible that this failure could put you in such a compromised position that you are forcded to default on your own payments in other instruments, which is of course a risk for your other counterparties. Therefore, it is interesting to ask the question – to what other parties am I connected indirectly, one swap to the next? This is transitive exposure. It is interesting to roll this up to ultimate parents; that is, among the ultimate parents, who am I connected to through transitive swap exposures?

This is relatively easy to compute in SPARQL.

SELECT distinct ?ultimateParent1 ?ultimateParent2

FROM <import:Swaps>

FROM <urn:work>

WHERE {

?company1 ex:tradingWith+ ?company2 .

?ultimateParent1 fw:ultimateParentOf ?company1 .

?ultimateParent2 fw:ultimateParentOf ?company2 .

}

Challenge: What are the names of these parties?

Challenge: can we use an INSERT versions of positionRollups and then use that information in a modified transitiveExposures query to present, along with ultimateParent1 and ultimateParent2, the rolled up exposure of each of these ultimate parents?

## Using “Crosswalks”

A “crosswalk” is a name given to a dataset that connects equivalent concepts from two different (meta-)data sets. In this exercise, we have a dataset from Thomson-Reuters that provides more information about legal entities. How can we use this data in the research we have been doing so far with the GLEIF and our swaps?

Let’s start by having a look at the Thomson-Reuters data. We have loaded it in a graph called <urn:PermID>. Let’s look at all the classes that are used in <urn:PermID>

SELECT DISTINCT ?c

WHERE {

GRAPH <urn:PermID>

{?s a ?c}

}

The results of this query include tr-org:Organization and trbc:BusinessSector. We will examine members of these two classes.

Let’s start by finding an organization and what we know about it

SELECT DISTINCT ?p

FROM <urn:PermID>

WHERE {?org a tr-org:Organization ; ?p ?o }

This will find all the properties that apply to members of the class Organization. The result includes tr-org:hasPrimaryBusinessSector. What do those business sectors look like?

SELECT DISTINCT ?bs

FROM <urn:PermID>

WHERE {

?org tr-org:hasPrimaryBusinessSector ?bsuri .

?bsuri rdfs:label ?bs

}

Now, how can we connect these business sectors to things we already know about? Another property in the list of known properties was tr-org:hasLEI . Let’s have a look at that.

SELECT DISTINCT ?bs ?lei

FROM <urn:PermID>

WHERE {

?org tr-org:hasPrimaryBusinessSector ?bsuri ;

tr-org:hasLEI ?lei .

?bsuri rdfs:label ?bs

} LIMIT 100

The data from the GLEIF also uses LEIs. Let’s connect those together.

SELECT ?gname ?trname ?bs

WHERE {GRAPH <urn:PermID> {

?trorg tr-org:hasPrimaryBusinessSector ?bsuri ;

vcard:organization-name ?trname ;

tr-org:hasLEI ?lei .

?bsuri rdfs:label ?bs .

}

GRAPH <urn:GLEIF> {

?leiuri fibo-fnd-aap-agt:identifies ?org .

?leiuri a fibo-be-le-lei:LegalEntityIdentifier ;

fibo-fnd-rel-rel:hasTag ?lei .

?org fibo-fnd-rel-rel:hasLegalName ?gname .

}}

LIMIT 100

We see that the crosswalk seems to work pretty well; we have the same name for the organizations on each side. We can also see the name of the business sector for each organization.

Now let’s combine that with things we learned from the GLEIF level 2 data. You might expect that if one organization is the parent of another, that they would be in the same business sector. Or maybe not. Let’s find out.

First, let’s remember the business sectors we computed for the entities in the GLEIF:

INSERT {GRAPH <urn:work> {?glorg fw:inSector ?bs}}

WHERE { GRAPH <urn:PermID> {

?trorg tr-org:hasPrimaryBusinessSector ?bsuri ;

vcard:organization-name ?trname ;

tr-org:hasLEI ?lei .

?bsuri rdfs:label ?bs .

}

GRAPH <urn:GLEIF> {

?leiuri fibo-fnd-aap-agt:identifies ?glorg .

?leiuri a fibo-be-le-lei:LegalEntityIdentifier ;

fibo-fnd-rel-rel:hasTag ?lei .

?glorg fibo-fnd-rel-rel:hasLegalName ?gname .

}}

Now, it is a simple matter to find the parent/child relationships that don’t match:

SELECT ?ParentBS ?ChildBS

FROM <urn:work>

WHERE {?parent fw:parentOf ?child .

?parent fw:inSector ?ParentBS .

?child fw:inSector ?ChildBS .

FILTER (?ChildBS != ?ParentBS)

}

We can do the same in the swaps; what are the business sectors of the counterparties in a swap?

SELECT ?Partner1BS ?Partner2BS

FROM <urn:work>

FROM <import:Swaps>

WHERE {?partner1 ex:tradingWith ?partner2 .

?partner1 fw:inSector ?Partner1BS .

?partner2 fw:inSector ?Partner2BS .

FILTER (?Partner1BS != ?Partner2BS)

}

Or, to bring all the datasets together, we can look at the business sectors of the ultimate payers:

SELECT DISTINCT ?fixedSector ?floatSector

FROM <urn:work>

FROM <import:Swaps>

WHERE {

?swap ex:hasUltimateFixedPayer ?ultimateFixedPayer .

?swap ex:hasUltimateFloatingPayer ?ultimateFloatingPayer .

?ultimateFixedPayer fw:inSector ?fixedSector .

?ultimateFloatPayer fw:inSector ?floatSector .

FILTER (?fixedSector != ?floatSector)

} LIMIT 10