Query CASTEMO knowledge graphs

Now, time to get knowledge out of the knowledge graphs. This chapter categorizes some useful queries to validate your data, and start digging information from the CASTEMO knowledge graphs stored in RethinkDB.

- Finding inconsistent and invalid data
- Querying CASTEMO knowledge graphs in Neo4j
- Querying with relations
- Get data of trial events and their participants (DISSINET-specific)
- Querying with counting

Finding inconsistent and invalid data

This is the collapsible text.

For various reasons, such as data import or bugs of some version of the interface, a CASTEMO knowledge graph can contain **inconsistent data**. It is thus important to identify this data and correct the inconsistencies either manually or from a script.

Get all entities involved in more than one synonym cloud

In the CASTEMO data model, one entity can be only involved in one synonym cloud.

ReQL

```
r.db("inkvisitor")
  .table("relations")
  .filter({ type: "SYN" })
  .getField("entityIds")
  .reduce(function (acc, val) {
    return acc.setUnion(val);
  })
  .default([])
  .map(function (ide) {
    return {
      id: ide,
      count: r
        .db("inkvisitor")
        .table("relations")
        .filter({ type: "SYN" })
        .getField("entityIds")
```

```
        .filter(function (syn) {
          return syn.contains(ide);
        })
        .count(),
    };
  })
  .filter(function (a) {
    return a("count").gt(1);
  });
```

R

Empty language: get labels of all concepts with empty language

ReQL

```
r.db("inkvisitor")
  .table("entities")
  .filter({ language: "", class: "C" })
  .getField("label");
```

Querying CASTEMO knowledge graphs in Neo4j

Querying with relations

Get data of trial events and their participants (DISSINET-specific)

How to match the person physically at trial (in a deposition, deponent)?

In DISSINET data, there are several concepts with different degree of generalization, which point to the person at trial. Generally, use this cascade of props under a trial event, preferring the first proptype match:

1. `bed73d79-d3e7-4036-b357-42ec0da37c43` (person at trial hearing - i edited the label today, but anyway the UUID is the holder of identity and should be used in all cases, not label)
2. `4e4b5f63-8eac-41ec-96d4-45cbd19ea8f6` (deponent)
3. `06a4e252-d4fb-4956-9a9a-2614efb70351` (witness)
4. `4e2bffec-ead1-499e-8bd2-81ba45602c9e` (testis)
5. `9e17df82-83fc-440d-8353-0a9c31a3766c` (participant in a trial)
6. `c5f3108d-d6ba-4ad7-afcc-2776ec9407ce` (defendant)
7. `36082ea7-61aa-43b1-a6f3-9f07be17f59a` (reus)

That is, if any "person at trial hearing" (`bed73d79-d3e7-4036-b357-42ec0da37c43`) is matched, don't continue and match this Person. Do not consider persons from different levels of the cascade, but only the first which is matched as proptype of a trial event, starting from no. 1 and continuing further. Because the point is not to match both deponent and defendant (the two can be different, e.g. if a witness gives testimony about another defendant in the trial of that defendant).

Querying with counting

Find Actions used in more than 20 statements

```
MATCH (s:Statement)-[:HAS_ACTION]-(a:Action)
WITH a, COUNT(DISTINCT s) AS st_count
WHERE st_count > 20
RETURN a.uuid as action_id, a.name as action_label, st_count as statement_count
```