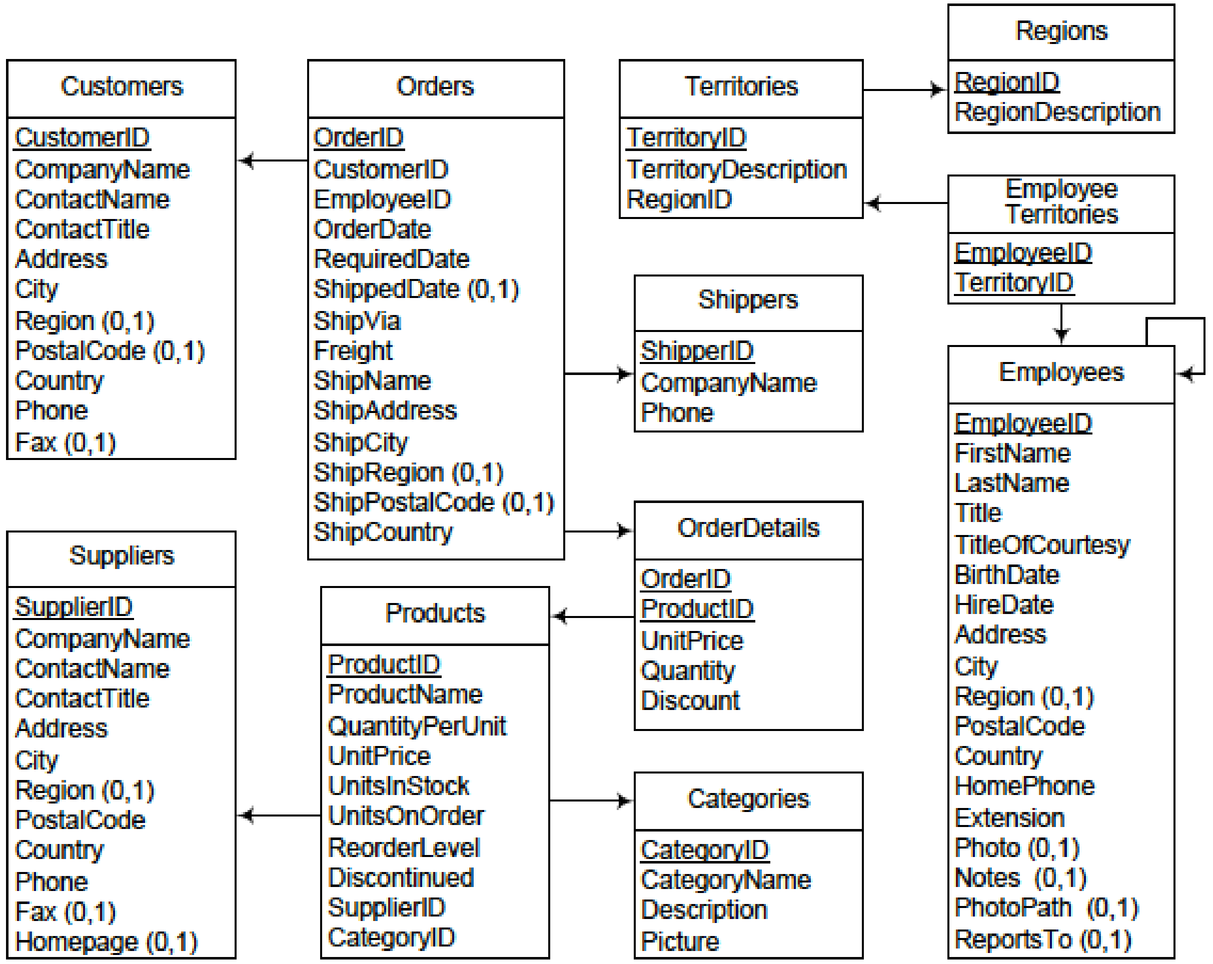
Introduction to Graph Databases

Neo4j

Alejandro Vaisman
avaisman@itba.edu.ar

Neo4j - Practice

Neo4j Practice – The Northwind Database



Neo4j Practice

1. Using the LOAD CVS statement

```
LOAD CSV WITH HEADERS FROM "file:///territories.csv" AS row
CREATE (:Territory {territoryID: row.territoryid,
name: row.territorydescription});

============
LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS row
CREATE (:Employee{employeeID: row.employeeid,
lastName: row.lastname,firstName: row.firstname, city:row.city,region:row.region,country:row.country});

=============

LOAD CSV WITH HEADERS FROM "file:///employeeterritories.csv" AS row
MATCH (t:Territory {territoryID: row.territoryid})
MATCH (e:Employee {employeeID: row.employeeid})
MERGE (e)-[:AssignedTo]->(t)
```

Neo4j Practice

2. Connecting to a Postgres DB

- Driver copied in the “Plugins” folder
- APOC library must also be copied in the “Plugins” folder

```
WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

%% NorthwindOLTP:  your database in the PostgreSQL instance
%% url: to be used in the procedure call
CALL apoc.load.jdbc(url,"select * from categories") YIELD row
% the query string can also mention just a table
% row: a “row variable” just as before
RETURN row.description,row.categoryname
```

This lists the table “categories” in Neo4j.
We can use this also for loading data into Neo4j.

Neo4j Practice

```
WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

CALL apoc.load.jdbc(url,"select * from products") YIELD row

CREATE (:Product {productID: row.productid,productName:row.productname, supplier: row.supplierid, category:row.categoryid,
qtyperunit:row.quantityperunit})

====================================


WITH "jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres" as url

CALL apoc.load.jdbc(url,"select * from suppliers") YIELD row

CREATE (:Supplier {supplierID: row.supplierid, supplierName:row.companyname, city:row.city, region:row.region, country:row.country})
```

Neo4j Practice

3. With Cypher

```
MATCH(s:Supplier)
MATCH(p:Product) where p.supplier=s.supplierID
MERGE (s)-[:Supplies]->(p)
```

Neo4j Practice - Creating the NW Graph

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/city.csv" AS row
CREATE (:City {cityID:row.citykey,cityName: row.cityname});

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/territories.csv" AS row
CREATE (:Territory {territoryID: row.territoryID, name: row.territoryDescription});

...

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/employee-territories.csv" AS row
MATCH (territory:Territory{territoryID: row.territoryID})
MATCH (employee:Employee {employeeID: row.employeeID})
MERGE (employee)-[:AssignedTo]->(territory);
```

Instituto Tecnológico de Buenos Aires

Neo4j Practice

-- To create the join of orders with order details.

CREATE VIEW order1 AS (SELECT o.orderid AS orderID,o.orderdate AS
 orderDate,o.shippeddate AS shippedDate,o.shipname AS shipName, sum(quantity)
AS totqty,sum(unitprice*quantity) AS totAmount FROM orders o,orderdetails o1
WHERE o.orderid=o1.orderid
group by o.orderid,o.orderdate,o.shippeddate,o.shipname
order by orderid asc)
SELECT * INTO ordershg FROM order1

COPY ordershg to 'C:\tmp\ordershg.csv' delimiter ',' CSV header USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:/NWdata/ordershg.csv" AS row
CREATE (:**Order** {orderID: row.orderid, orderDate: row.orderdate,
ShippedDate: row.shippeddate,shipName:row.shipname,totalQty:row.totqty, totalAmount:row.totamount});

You can also connect directly to a PostgreSQL database

CALL apoc.load.jdbc('jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres','select * from ordershg') YIELD row
CREATE (:Order {orderID: row.orderid, orderDate: row.orderdate, ShippedDate: row.shippeddate,shipName:row.shipname,totalQty:row.totqty, totalAmount:row.totamount});

Neo4j Practice

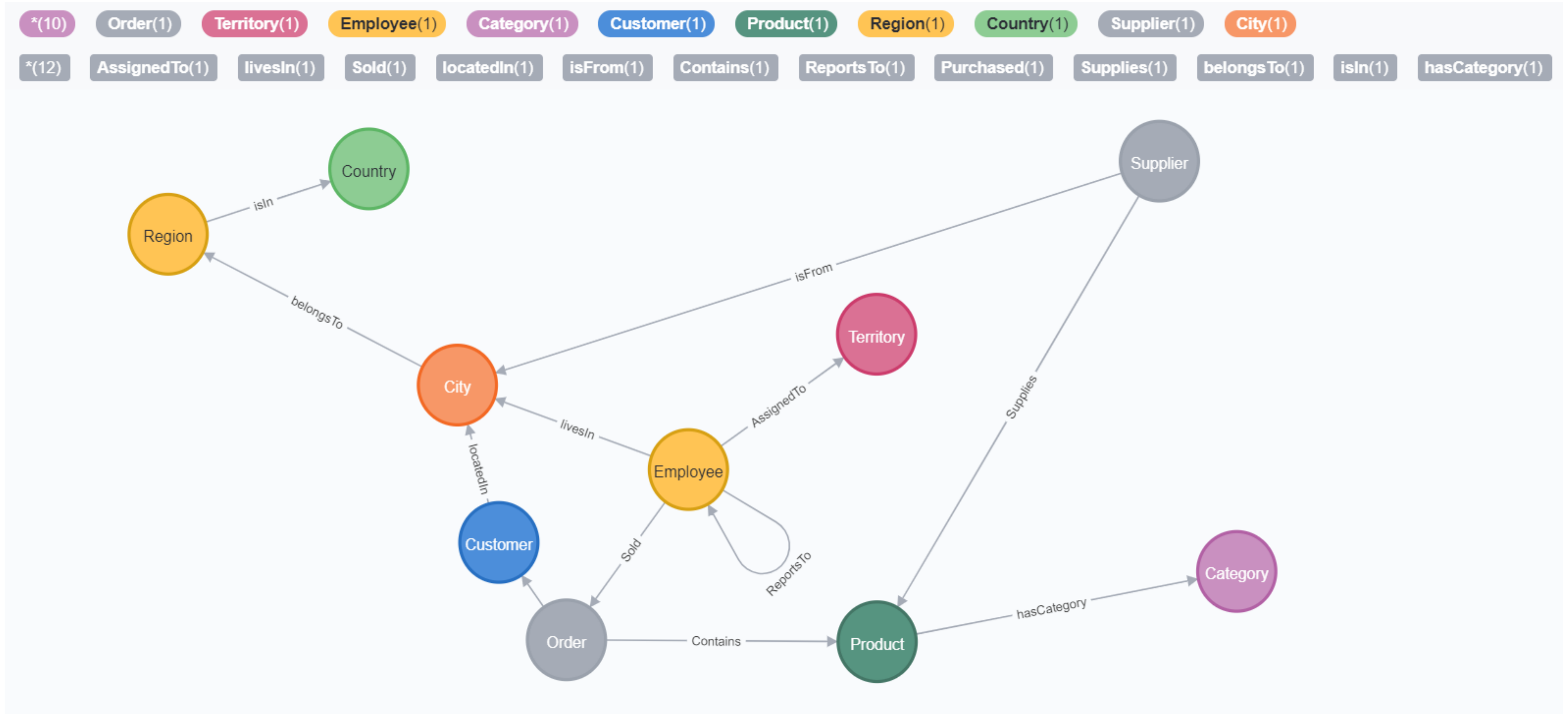
```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/orders.csv" AS row
MATCH (order:Order {orderID: row.orderID})
MATCH (employee:Employee {employeeID: row.employeeID})
MERGE (employee)-[:Sold]->(order);

LOAD CSV WITH HEADERS FROM "file:/NWdata/orderdetails.csv" AS row
MATCH (order:Order {orderID: row.orderID})
MATCH (product:Product {productID: row.productID})
MERGE (order)-[:Contains{unitPrice:row.unitPrice,quantity:row.quantity,  discount:row.discount}]->(product);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/NWdata/products.csv" AS row
MATCH (product:Product {productID: row.productID})
MATCH (supplier:Supplier {supplierID: row.supplierID})
MERGE (supplier)-[:Supplies]->(product);

CALL apoc.load.jdbc('jdbc:postgresql://localhost:5433/NorthwindOLTP?user=postgres&password=postgres','select * from employees') YIELD row
MATCH (employee:Employee {employeeID: row.employeeid})
MATCH (employee1:Employee {employeeID: row.reportsto})
MERGE (employee)-[:ReportsTo]->(employee1);
```

Schema: Northwindhg database



Problem 1. Northwindhg database

- Query 1. List products and their unit price.

```
MATCH (p:Product)
RETURN p.productName, p.unitPrice
ORDER BY p.unitPrice DESC
```

- Query 2. List information about products 'Chocolade' & 'Pavlova'.

```
MATCH (p:Product)
WHERE p.productName IN ['Chocolade','Pavlova']
RETURN p
```

- Query 3. List information about products with names starting with a "C", whose unit price is greater than 50.

```
MATCH (p:Product)
WHERE p.productName STARTS WITH "C" AND tofloat(p.unitPrice) > 50
RETURN p.productName, p.unitPrice;
```

- Query 4. Same as 3, but considering the sales price, not the product's price.

```
MATCH (p:Product) <- [c:Contains] - (o:Order)
WHERE p.productName STARTS WITH "C" AND tofloat(c.unitPrice) > 50
RETURN distinct p.productName, p.unitPrice,c.unitPrice;
```

Problem 1. Northwindhg database

- Query 5. Total purchased by customer and product.

```
MATCH (c:Customer)
OPTIONAL MATCH (p:Product)<-[pu:Contains]-(:Order)-[:Purchased]-
>(c) RETURN c.customerName,p.productName, tofloat(sum(tofloat(pu.unitPrice) * toInteger (pu.quantity))) AS volume
ORDER BY volume DESC;
```

- Query 6. Top 10 employees, considering the number of orders sold.

```
MATCH (:Order)<-[:Sold]-(e:Employee)
RETURN e.firstName,e.lastName, count(*) AS Ordenes
ORDER BY Ordenes DESC LIMIT 10
```

- Query 7. For each employee, list the assigned territories.

```
 MATCH (t:Territory)<-[:AssignedTo]-(e:Employee)
 RETURN e.lastName, COLLECT(t.name);
```

- Query 8. For each city, list the companies settled in that city.

```
MATCH (c:City)<-[:locatedIn]-(c1:Customer)
RETURN c.cityname, COLLECT(c1.customerName);
```

Problem 1. Northwindhg database

- Query 10. How many persons an employee reports to, either directly or transitively?

```
MATCH (report:Employee)
OPTIONAL MATCH (e)<-[rel:ReportsTo*]-(report)
RETURN report.lastName AS e1, COUNT(rel) AS reports
```

- Query 11. Whom do persons called “Robert” report to?

```
MATCH (e:Employee)<-[:ReportsTo*]-(sub:Employee)
WHERE sub.firstName = 'Robert'
RETURN e.firstName,e.lastName,sub.lastName
```

- Query 12. Who does not report to anybody?

```
MATCH (e:Employee)
WHERE NOT (e)-[:ReportsTo]->()
RETURN e.firstName  as TopBossFirst, e.lastName as TopBossLast
```

- Query 13. Suppliers, number of categories they supply, and a list of such categories

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
WITH s.supplierName as Supplier, collect(distinct c.categoryName) as Categories
WITH Supplier, Categories, size(Categories) AS Cantidad ORDER BY Cantidad DESC
RETURN Supplier, Cantidad, Categories;
```

Problem 1. Northwindhg database

- Query 14. Suppliers who supply beverages

```
MATCH (c:Category {categoryName:"Beverages"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.supplierName as ProduceSuppliers;
```

- Query 15. Customer who purchases the largest amount of beverages

```
MATCH (cust:Customer)<-[:Purchased]-(:Order)-[o:Contains]->(p:Product),
(p)-[:hasCategory]->(c:Category {categoryName:"Beverages"})
RETURN DISTINCT cust.customerName as CustomerName, SUM(toInteger(o.quantity)) as cant ORDER by  cant DESC LIMIT 1
```

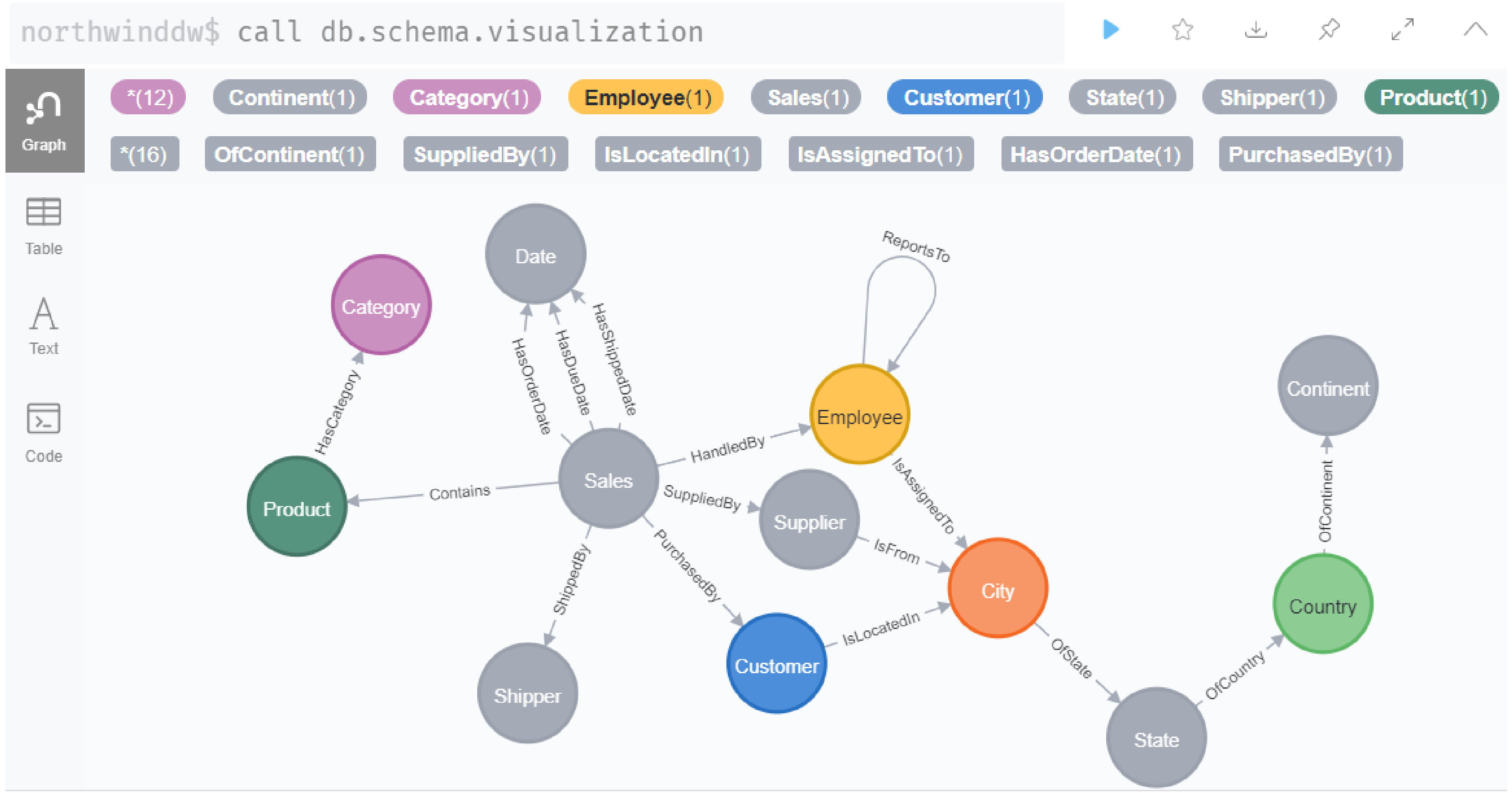
- Query 16. List the 5 most popular products (considering the number of orders)

```
MATCH (c:Customer)<-[:Purchased]-(o:Order)-[o1:Contains]->(p:Product)
return c.customerName, p.productName, count(o1) as orders
order by orders desc  LIMIT 5
```

- Query 17. Products ordered by customers from the same country than their suppliers

```
MATCH (c:Customer)-[r:locatedIn]->(cy:City) -[:belongsTo]-(:Region)-[:isIn]->(co:Country)
WITH co,c MATCH (s:Supplier) WHERE co.countryname=s.country
WITH s,co,c MATCH (s)-[su:Supplies]->(p:Product) <-[:Contains]-(o:Order)-[:Purchased]->(c)
RETURN c.customerName,s.supplierName,co.countryname,p.productName
```

Problem 2 – NorthwindDW database



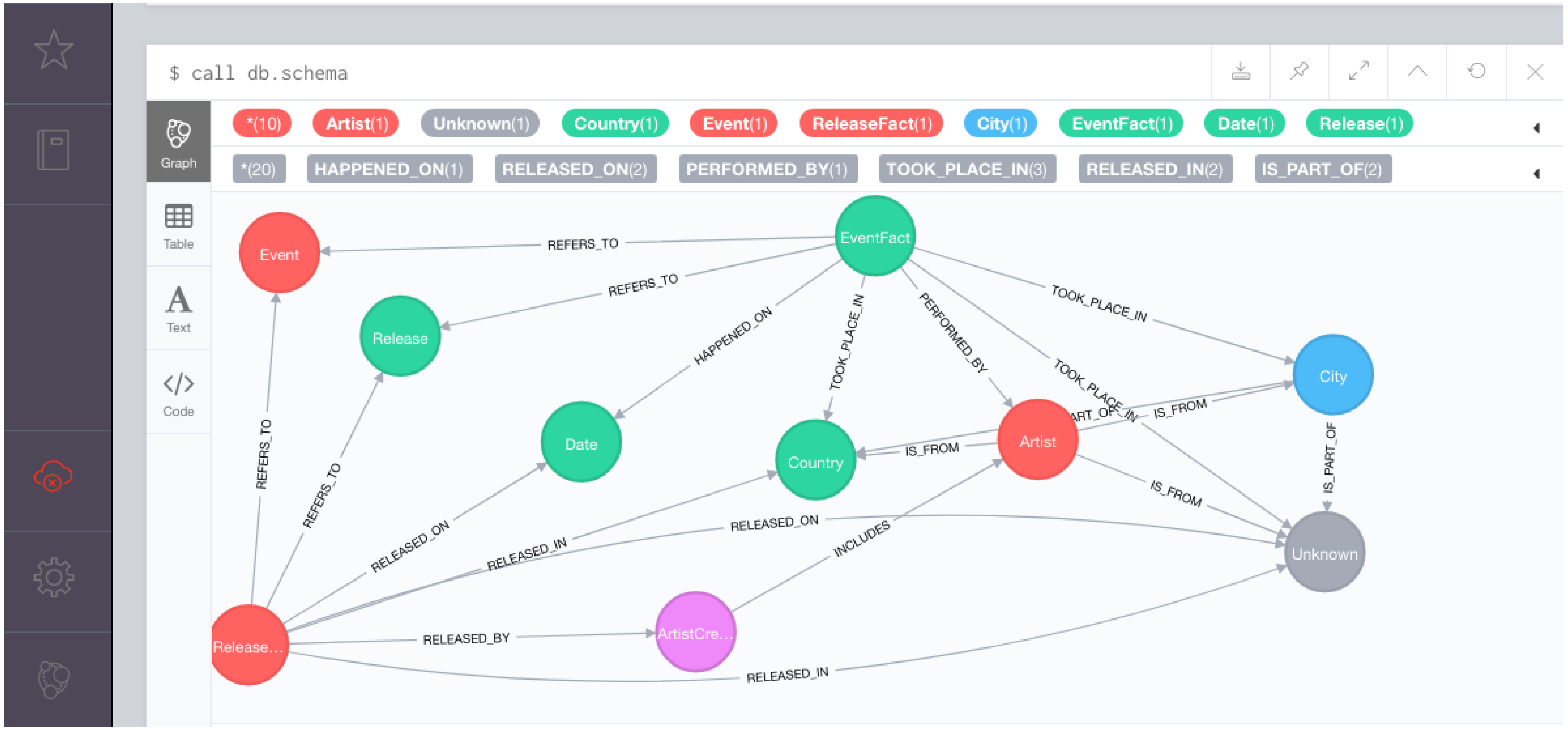
```
northwinddw$ call db.schema.visualization
```

Problem 2 – NorthwindDW database

Query 1. Total sales amount per customer, year, and product category.

```
MATCH (c:Category)<-[hc:HasCategory]-(p:Product)<-[pu:Contains]- (s:Sales)-[:PurchasedBy]->(cu:Customer)
MATCH (s)-[:HasOrderDate]->(d:Date)
RETURN cu.CompanyName AS Customer,c.CategoryName as Category,d.year,
sum(tofloat(s.SalesAmount)) AS Volume
ORDER BY Customer DESC;
```

Problem 3 – MusicBrainz database



```
$ call db.schema
```

Problem 3 – MusicBrainz database

Compute the triples of artists, and the number of times they have performed together in an event, if this number is at least 3.

```
MATCH (a1:Artist)<-[]-(e:EventFact)-[]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1,a2,COLLECT(e) AS events WHERE SIZE(events) > 2
MATCH (a1:Artist)<-[]-(e1:EventFact)-[]->(a2:Artist)
MATCH (a3:Artist)<-[]-(e1) WHERE a2.id < a3.id
WITH a1.name as name1, a2.name as name2,a3.name as name3, COUNT(e1.idEvent) as nbrTimes WHERE nbrTimes > 2
RETURN name1,name2,name3, nbrTimes ORDER BY nbrTimes DESC
```

Compare against this solution:

```
MATCH (a1:Artist)<-[]-(e:EventFact)-[]->(a2:Artist)
MATCH (a3:Artist)<-[]-(e) WHERE a1.id < a2.id and a2.id < a3.id
WITH a1.name as name1, a2.name as name2,a3.name as name3, COUNT(e.idEvent) as nbrTimes WHERE nbrTimes > 2
RETURN name1,name2,name3, nbrTimes ORDER BY nbrTimes DESC
```

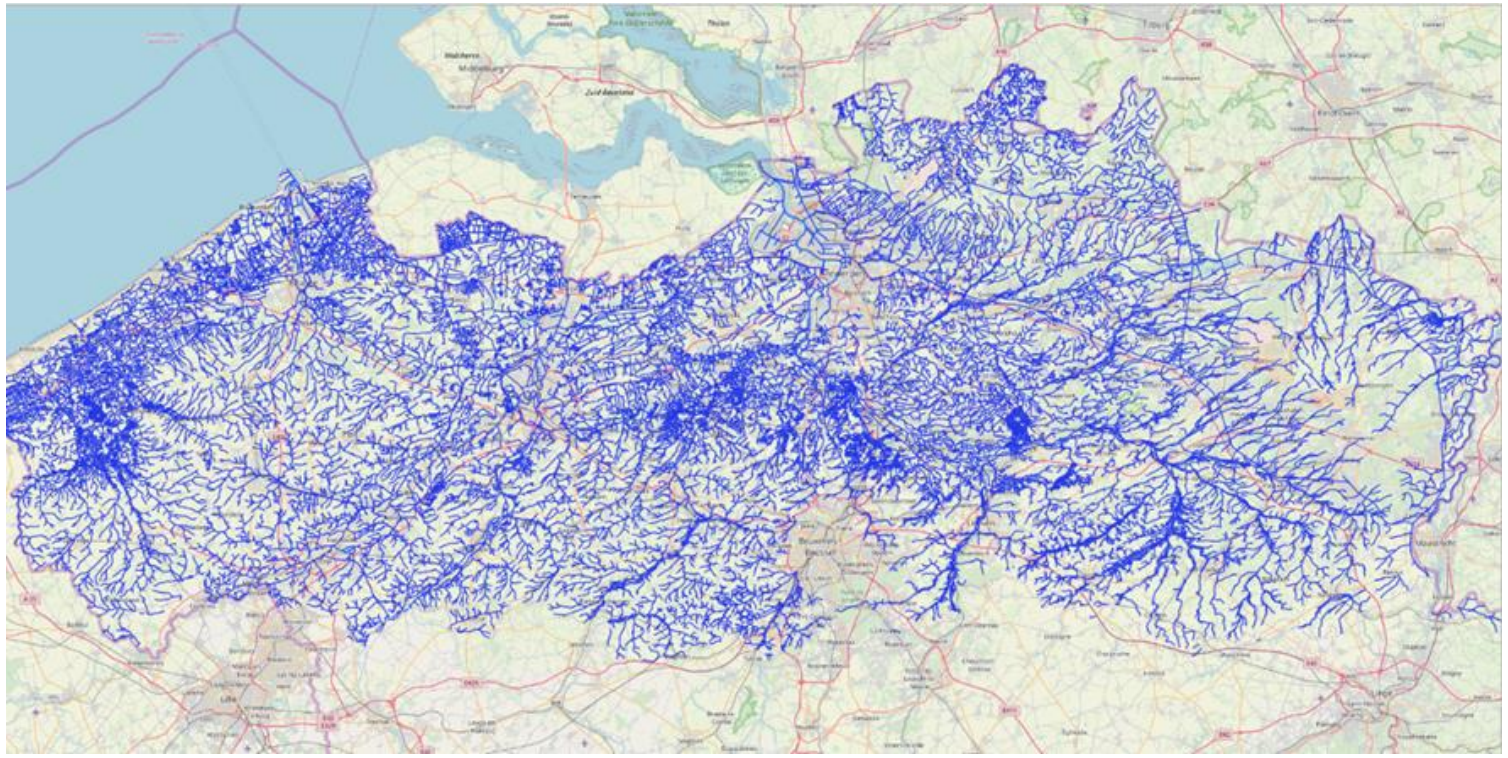
Problem 3 – MusicBrainz database

Compute the quadruples of artists, and the number of times they have performed together in an event, if this number is at least 3.

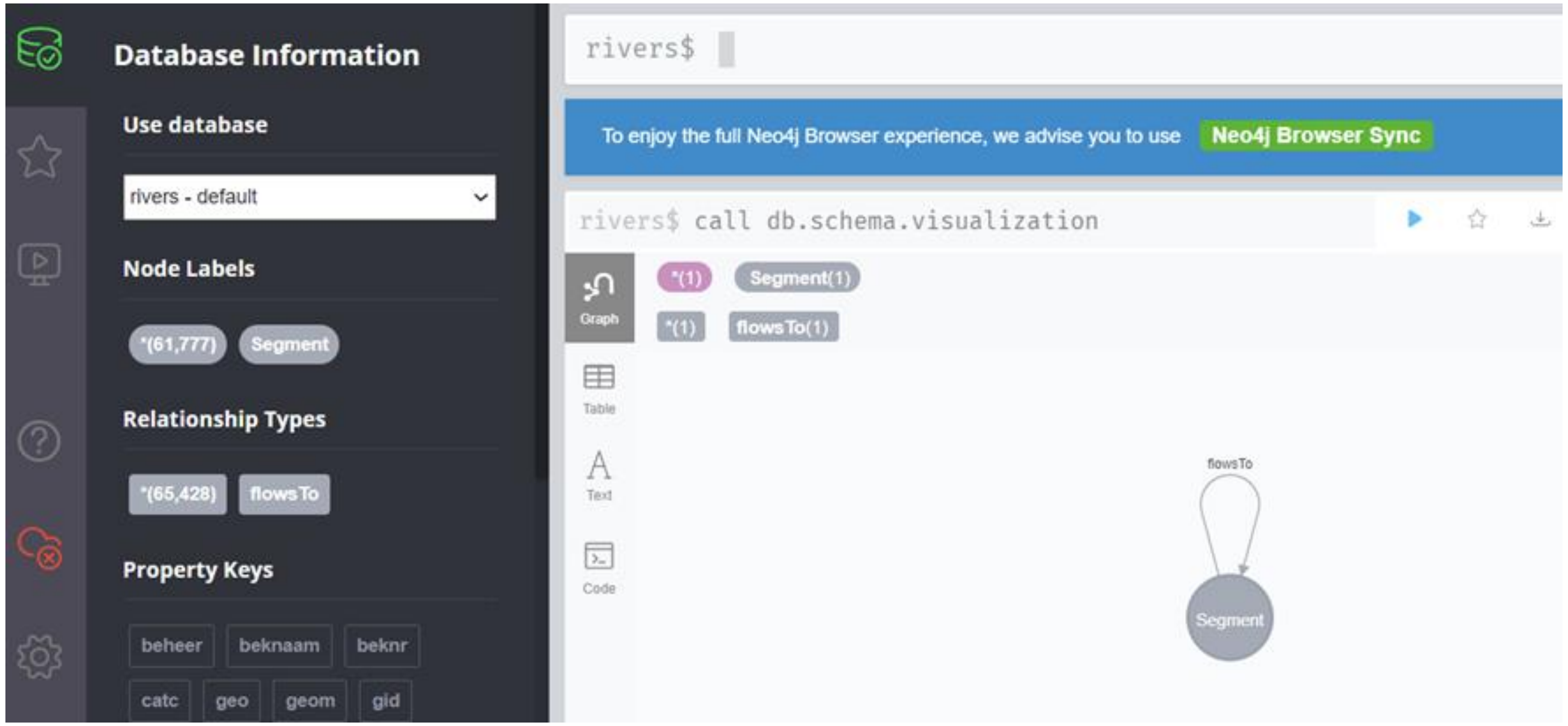
This is not the best solution:

```
MATCH (a1:Artist)<-[]-(e:EventFact)-[]->(a2:Artist)
WHERE a1.id < a2.id
MATCH (a3:Artist)<-[]-(e) WHERE a2.id < a3.id
MATCH (a4:Artist)<-[]-(e:EventFact) WHERE a3.id < a4.id
WITH a1.name as name1, a2.name as name2, a3.name as name3,
a4.name as name4, COUNT(e.idEvent) as nbrTimes WHERE nbrTimes > 2
RETURN name1,name2,name3,name4,nbrTimes ORDER BY nbrTimes DESC
```

Problem 4 – Rivers



Problem 4 – Rivers



Database Information

Use database

rivers - default

Node Labels

*(61,777) Segment

Relationship Types

*(65,428) flowsTo

Property Keys

beheer beknaam beknr catc geo geom gid

```
rivers$
```

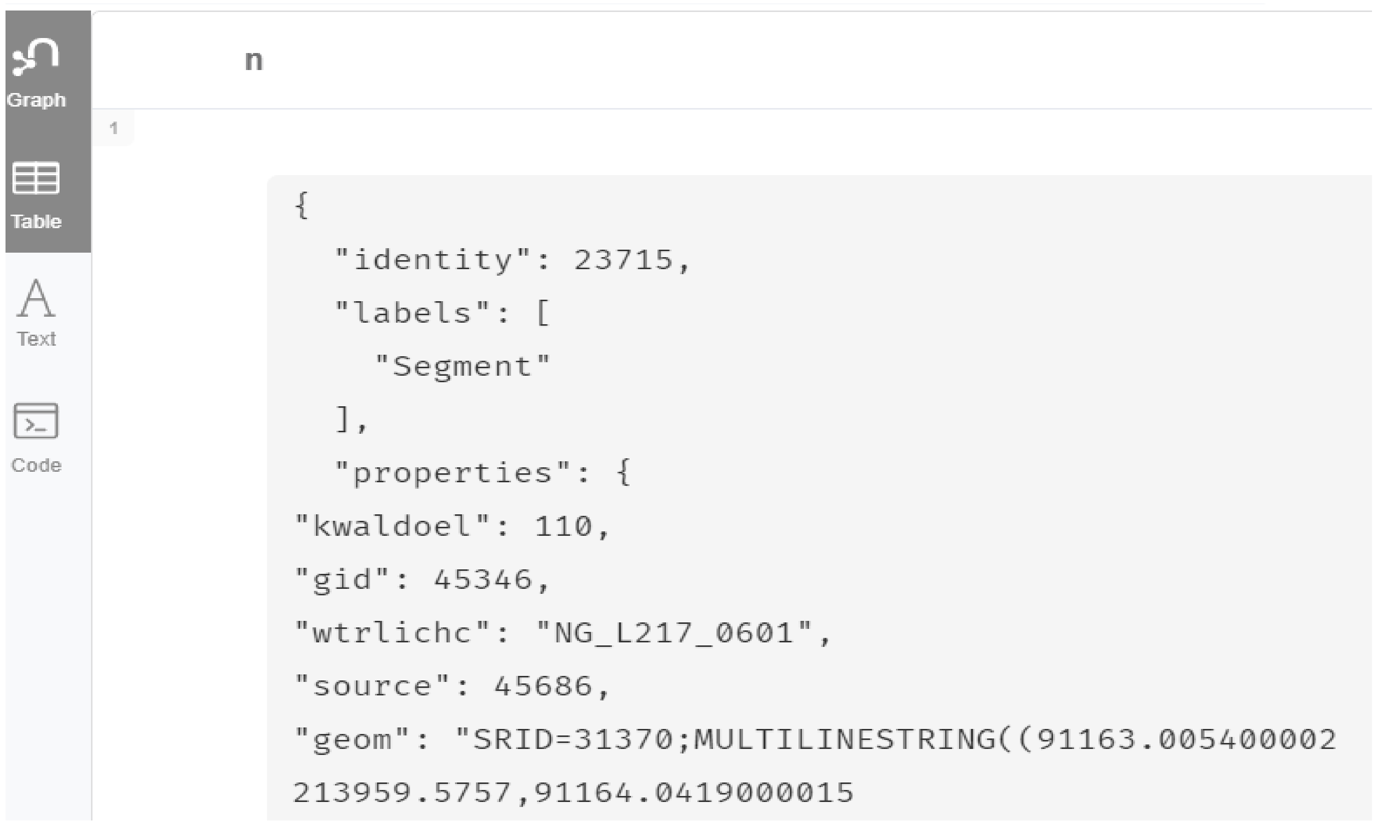
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync

```
rivers$ call db.schema.visualization
```

*(1) Segment(1)

*(1) flowsTo(1)

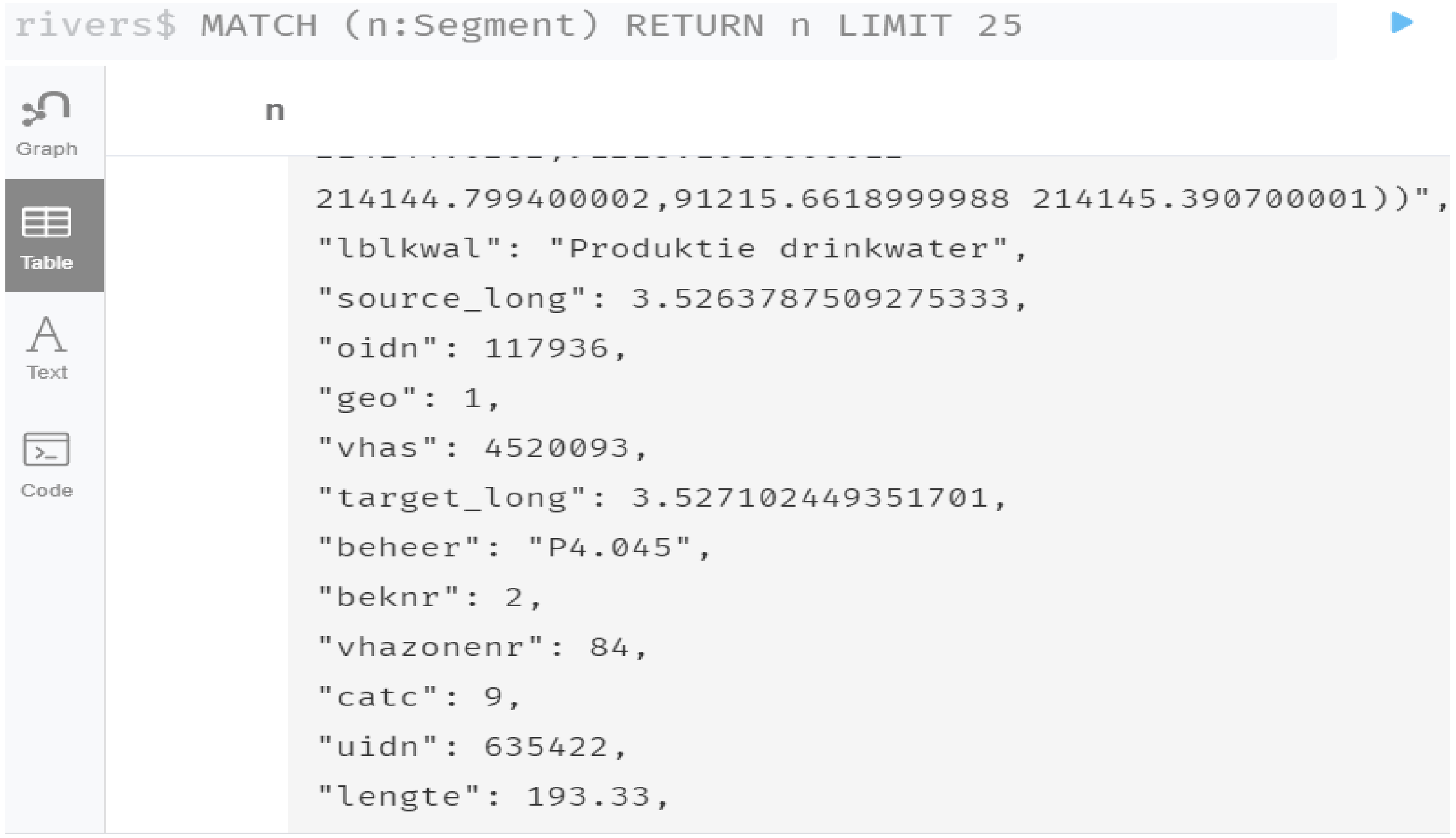
Problem 4 – Rivers



n

```
{
  "identity": 23715,
  "labels": [
    "Segment"
  ],
  "properties": {
"kwaldoel": 110,
"gid": 45346,
"wtrlichc": "NG_L217_0601",
"source": 45686,
"geom": "SRID=31370;MULTILINESTRING((91163.005400002
213959.5757,91164.0419000015
```

Problem 4 – Rivers



```
rivers$ MATCH (n:Segment) RETURN n LIMIT 25
```

```
n
214144.799400002,91215.6618999988 214145.390700001))",
"lblkwal": "Produktie drinkwater",
"source_long": 3.5263787509275333,
"oidn": 117936,
"geo": 1,
"vhas": 4520093,
"target_long": 3.527102449351701,
"beheer": "P4.045",
"beknr": 2,
"vhazonenr": 84,
"catc": 9,
"uidn": 635422,
"lengte": 193.33,
```

Problem 4 – Rivers

Find the number of splits in the downstream path of segment 6020612

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path AS pp
UNWIND NODES(pp) as p
MATCH (p)-[:flowsTo]->(r:Segment)
WITH p, count(DISTINCT r) as co WHERE co > 1
RETURN count(p)
```

The **spanningTree** function from the APOC library is used. This function computes all simple paths that can be reached starting from a node in the graph, using breath-first search by default. This is done visiting nodes only once. The **relationshipfilter** is "flowsTo ", indicating that the path must traverse only this relation, in downstream direction.
A collection of paths is returned (pp), which is then flattened as a table using UNWIND. All reachable nodes are obtained. For each node in this table, it is tested if this node has more than one outgoing segments. If this is the case, there is a split. The node with vhas:6020612 is chosen for the test because it is one of the farthest from the sea, thus its flow downstream is one of the longest ones.

Problem 4 – Rivers

Find the length, the # of segments, and the IDs of the segments, of the longest branch of upstream flow starting from a given segment.

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.expandConfig(n,{relationshipFilter:"<flowsTo", minLevel: 1}) YIELD path AS pp
WITH reduce(longi= tofloat(0), n IN nodes(pp)| longi+ tofloat(n.lengte)) AS blength, Length(pp) as
alength, [p in NODES(pp) |p.vhas] AS nodelist
WITH blength, alength, nodelist[size(nodelist)-1] as id
WITH id, max(blength) as ml,  collect([id,blength,alength]) as coll
WITH id, ml, [p in coll WHERE p[0]= id  AND p[1]=ml|p[2]] AS lhops
UNWIND lhops as hops
RETURN id,ml,hops order by id desc;
```

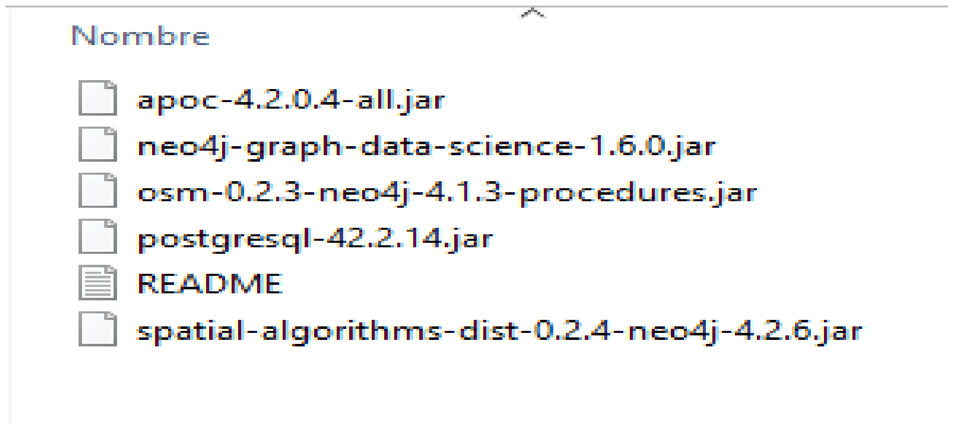
Problem 5 – Rivers

Find all segments reachable from the segment closest to Antwerpen's Groenplaats

```
CALL apoc.spatial.geocodeOnce('Groenplaats Antwerpen Flanders Belgium') YIELD location as ini
MATCH (n:Segment)
WITH n, ini, distance(point({longitude:n.source_long, latitude:n.source_lat}),  point({longitude:ini.longitude, latitude:ini.latitude}) ) as d
WITH n, d order by d asc limit 1
CALL apoc.path.spanningTree (n,{relationshipFilter:"flowsTo>", minLevel: 1})  YIELD path as pp
UNWIND NODES(pp) as p
RETURN p.vhas;
```

OR:

```
RETURN [p in NODES(pp)|p.vhas];
```


Nombre
apoc-4.2.0.4-all.jar
neo4j-graph-data-science-1.6.0.jar
osm-0.2.3-neo4j-4.1.3-procedures.jar
postgresql-42.2.14.jar
README
spatial-algorithms-dist-0.2.4-neo4j-4.2.6.jar