Big Data

Graph Databases

von
Benjamin Ellmer (S2210455012)

Mobile Computing Master
FH Hagenberg

May 30, 2023

Installation

Start Neo4j

```
git clone https://github.com/Digital-Media/neo4j.git
docker compose -f neo4j/docker-compose.yml up -d
```

Start Postgres

```
docker run --name postgres-big-data-ex4 -e POSTGRES_PASSWORD=
   geheim -d postgres:14
```

Step 1 - Translation

Connect to postgres container

```
docker exec -it postgres-big-data-ex4 psql -U postgres
```

Drop, create and select schema:

```
DROP SCHEMA IF EXISTS graph_demos CASCADE;
CREATE SCHEMA IF NOT EXISTS graph_demos;
SET search_path TO graph_demos;
```

Create folks table:

```
CREATE TABLE IF NOT EXISTS folks (
    id bigint NOT NULL,
    name varchar(100) NOT NULL,
    father bigint NULL,
    mother bigint NULL,
    PRIMARY KEY (id),
    CONSTRAINT father_fk FOREIGN KEY (father) REFERENCES folks
       (id),
    CONSTRAINT mother_fk FOREIGN KEY (mother) REFERENCES folks
       (id)
);
```

Insert folks:

```
INSERT INTO folks (id, name, father, mother) VALUES
(100, 'Alex', 20, 30),
(20, 'Dad', 10, null),
(30, 'Mom', null, null),
(10, 'Grandpa Bill', null, null),
(98, 'Sister Amy', 20, 30);
```

Create vertices table:

```
CREATE TABLE vertices (
    vertex_id bigint NOT NULL,
    alias varchar (255),
    label varchar (255),
    name varchar (255),
    type varchar (255),
    properties jsonb,
    PRIMARY KEY (vertex_id)
);
```

Insert vertices:

```
INSERT INTO vertices (vertex_id, alias, label, name, type)
   VALUES
(1, 'NAmerica', 'Location', 'North America', 'continent'),
(2, 'Europe', 'Location', 'Europe', 'continent'),
(3, 'USA', 'Location', 'United States', 'country'),
(4, 'UK', 'Location', 'United Kingdom', 'country'),
(5, 'England', 'Location', 'England', 'country'),
(6, 'Austria', 'Location', 'Österreich', 'country'),
(7, 'Idaho', 'Location', 'Idaho', 'state'),
(8, 'London', 'Location', 'London', 'city'),
(9, 'UpperAustria', 'Location', 'Oberösterreich', 'Bundesland'
   ),
(10, 'Waldviertel', 'Location', 'Waldviertel', 'Viertel'),
(11, 'Grein', 'Location', 'Grein', 'city'),
(12, 'Andrea', 'Person', 'Andrea', 'person'),
(13, 'Bert', 'Person', 'Bert', 'person'),
(14, 'Christian', 'Person', 'Christian', 'person');
```

Create edges table:

```
CREATE TABLE edges (
    edge_id bigint NOT NULL,
    tail_vertex bigint REFERENCES vertices (vertex_id),
    head_vertex bigint REFERENCES vertices (vertex_id),
    label varchar(255),
    properties jsonb,
    PRIMARY KEY (edge_id),
    CONSTRAINT tail_vertex_fk FOREIGN KEY (tail_vertex)
       REFERENCES vertices(vertex_id),
    CONSTRAINT head_vertex_fk FOREIGN KEY (head_vertex)
       REFERENCES vertices(vertex_id)
);
```

Insert edges:

```
INSERT INTO edges (edge_id, tail_vertex, head_vertex, label)
   VALUES
(1, 3, 1, 'within'),
(2, 4, 2, 'within'),
(3, 5, 4, 'within'),
(4, 6, 2, 'within'),
(5, 7, 3, 'within'),
(6, 8, 5, 'within'),
(7, 9, 6, 'within'),
(8, 10, 9, 'within'),
(9, 11, 10, 'within'),
(10, 12, 7, 'born_in'),
(11, 12, 8, 'lives_in'),
(12, 13, 11, 'born_in'),
(13, 13, 8, 'lives_in'),
(14, 14, 8, 'born_in'),
(15, 12, 13, 'married'),
(16, 13, 12, 'married');
```

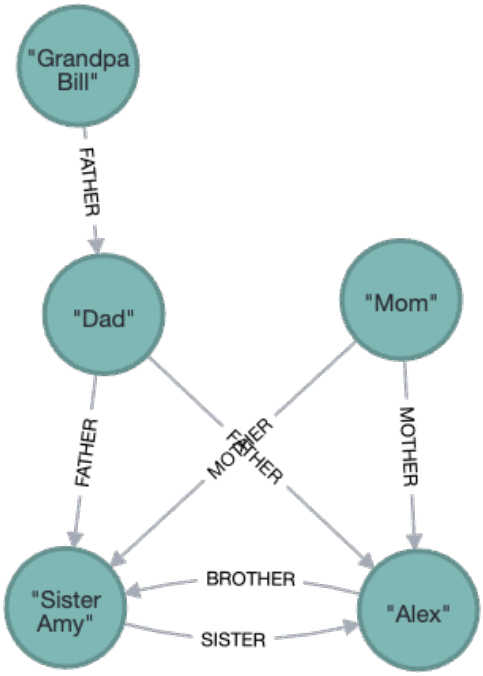
Step 2

Connect to neo4j container:

```
docker exec -it neo4j cypher-shell -u neo4j -p password
```

Create Folks with relationships:

```
CREATE (alex:Folk {id: 100, name: 'Alex'})
CREATE (dad:Folk {id: 20, name: 'Dad'})
CREATE (mom:Folk {id: 30, name: 'Mom'})
CREATE (grandpa:Folk {id: 10, name: 'Grandpa Bill'})
CREATE (amy:Folk {id: 98, name: 'Sister Amy'})

CREATE (dad)-[:FATHER]->(alex)
CREATE (mom)-[:MOTHER]->(alex)

CREATE (dad)-[:FATHER]->(amy)
CREATE (mom)-[:MOTHER]->(amy)

CREATE (alex)-[:BROTHER]->(amy)
CREATE (amy)-[:SISTER]->(alex)

CREATE (grandpa)-[:FATHER]->(dad);
```



Create Locations with relationships:

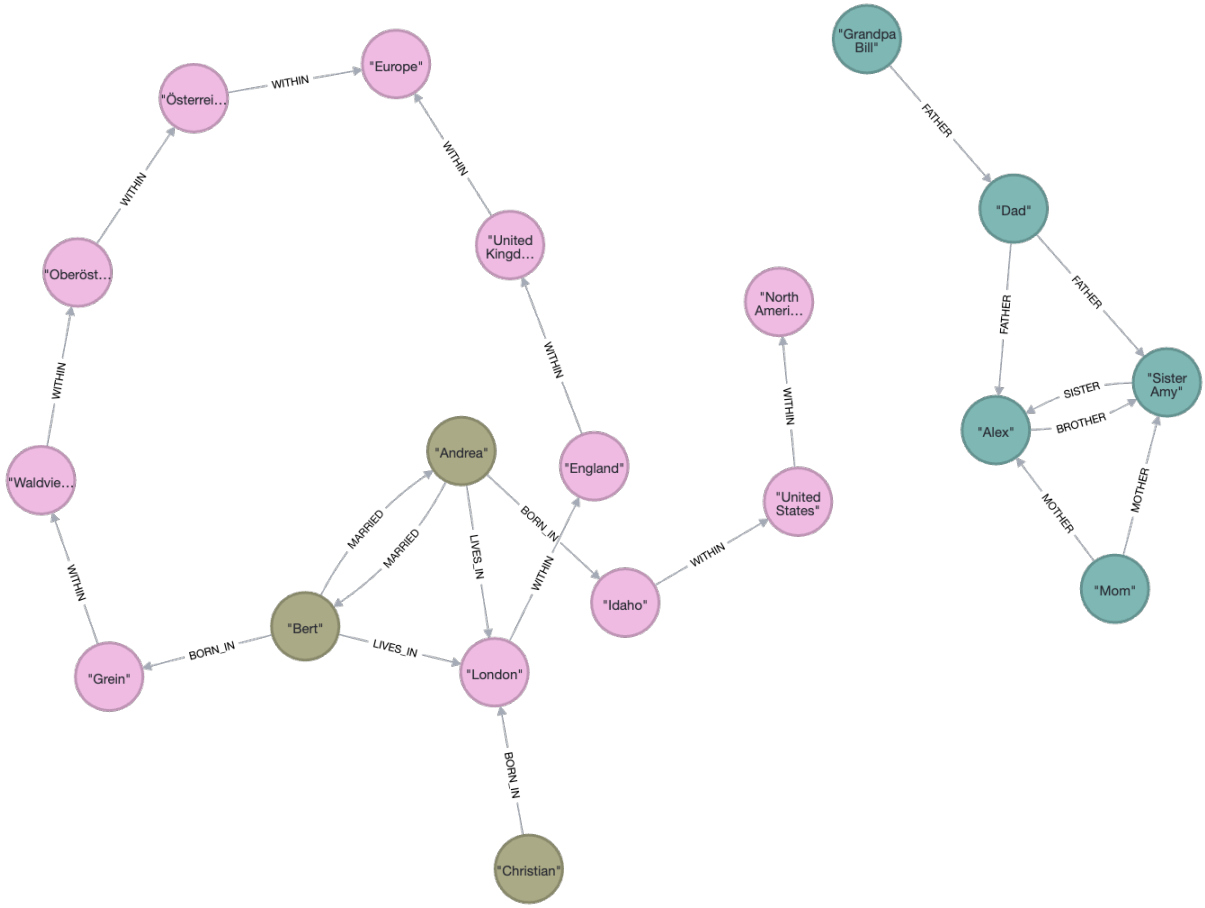
```
CREATE (namerica:Location {alias: 'NAmerica', label: 'Location
   ', name: 'North America', type: 'continent'})
CREATE (europe:Location {alias: 'Europe', label: 'Location',
   name: 'Europe', type: 'continent'})
CREATE (usa:Location {alias: 'USA', label: 'Location', name: '
   United States', type: 'country'})
CREATE (uk:Location {alias: 'UK', label: 'Location', name: '
   United Kingdom', type: 'country'})
CREATE (england:Location {alias: 'England', label: 'Location',
    name: 'England', type: 'country'})
CREATE (austria:Location {alias: 'Austria', label: 'Location',
    name: 'Österreich', type: 'country'})
CREATE (idaho:Location {alias: 'Idaho', label: 'Location',
   name: 'Idaho', type: 'state'})
CREATE (london:Location {alias: 'London', label: 'Location',
   name: 'London', type: 'city'})
CREATE (upperaustria:Location {alias: 'UpperAustria', label: '
   Location', name: 'Oberösterreich', type: 'Bundesland'})
CREATE (waldviertel:Location {alias: 'Waldviertel', label: '
   Location', name: 'Waldviertel', type: 'Viertel'})
CREATE (grein:Location {alias: 'Grein', label: 'Location',
   name: 'Grein', type: 'city'})

CREATE (usa)-[:WITHIN]->(namerica)
CREATE (uk)-[:WITHIN]->(europe)
CREATE (england)-[:WITHIN]->(uk)
CREATE (austria)-[:WITHIN]->(europe)
CREATE (idaho)-[:WITHIN]->(usa)
CREATE (london)-[:WITHIN]->(england)
CREATE (upperaustria)-[:WITHIN]->(austria)
CREATE (waldviertel)-[:WITHIN]->(upperaustria)
CREATE (grein)-[:WITHIN]->(waldviertel);
```

Create Persons with relationships:

```
MATCH (idaho:Location {alias: 'Idaho'})
MATCH (london:Location {alias: 'London'})
MATCH (grein:Location {alias: 'Grein'})

CREATE (andrea:Person {alias: 'Andrea', label: 'Person', name:
    'Andrea', type: 'person'})
CREATE (bert:Person {alias: 'Bert', label: 'Person', name: '
   Bert', type: 'person'})
CREATE (christian:Person {alias: 'Christian', label: 'Person',
    name: 'Christian', type: 'person'})

CREATE (andrea)-[:BORN_IN]->(idaho)
CREATE (andrea)-[:LIVES_IN]->(london)

CREATE (bert)-[:BORN_IN]->(grein)
CREATE (bert)-[:LIVES_IN]->(london)

CREATE (christian)-[:BORN_IN]->(london)

CREATE (andrea)-[:MARRIED]->(bert)
CREATE (bert)-[:MARRIED]->(andrea);
```

Show the whole graph:

```
MATCH (n) OPTIONAL MATCH (n)-[r]-() RETURN n, r;
```



Read all :Person nodes:

```
MATCH (p:Person) RETURN p;
```

```
neo4j@neo4j> MATCH (p:Person) RETURN p;
+-----------------------------------------------------------------------------------------------------+
| p                                                                                                   |
+-----------------------------------------------------------------------------------------------------+
| (:Person {name: "Andrea", alias: "Andrea", label: "Person", type: "person", vertex_id: 12})         |
| (:Person {name: "Bert", alias: "Bert", label: "Person", type: "person", vertex_id: 13})             |
| (:Person {name: "Christian", alias: "Christian", label: "Person", type: "person", vertex_id: 14}) |
+-----------------------------------------------------------------------------------------------------+

3 rows
ready to start consuming query after 108 ms, results consumed after another 3 ms
```

Return all names of :Person nodes:

```
MATCH (p:Person) RETURN p.name;
```

```
neo4j@neo4j> MATCH (p:Person) RETURN p.name;
+-------------+
| p.name      |
+-------------+
| "Andrea"    |
| "Bert"      |
| "Christian" |
+-------------+

3 rows
```

Attention: The exercise steps tell us to create a new person Grandma Mary. Since Grandma Mary fits much better to the Folks than to the Persons I decided to make her a Folk.

Create a new ~~Person~~ $\xrightarrow{Folk}$ - Grandma Mary:

```
CREATE (grandma:Folk {name: "Grandma Mary"});
```

Make Grandma Mary Mother of Mom and Dad:

```
MATCH (mom:Folk {name: 'Mom'})
MATCH (dad:Folk {name: 'Dad'})
MATCH (grandma:Folk {name: 'Grandma Mary'})

CREATE (grandma)-[:MOTHER]->(mom)
CREATE (grandma)-[:MOTHER]->(dad);
```

Return the ancestors of Alex as nodes:

```
MATCH (alex:Folk {name: 'Alex'})
OPTIONAL MATCH path = (alex)<-[:MOTHER|FATHER*]-(ancestor:Folk
    )
RETURN DISTINCT ancestor AS ancestors;
```

```
neo4j@neo4j> MATCH (alex:Folk {name: 'Alex'})
             OPTIONAL MATCH path = (alex)<-[:MOTHER|FATHER*]-(ancestor:Folk)
             RETURN DISTINCT ancestor AS ancestors;
+--------------------------------------+
| ancestors                            |
+--------------------------------------+
| (:Folk {name: "Mom", id: 30})        |
| (:Folk {name: "Dad", id: 20})        |
| (:Folk {name: "Grandma Mary"})       |
| (:Folk {name: "Grandpa Bill", id: 10}) |
+--------------------------------------+

4 rows
ready to start consuming query after 203 ms, results consumed after another 4 ms
```

Return the names of the ancestors of Alex:

```
MATCH (alex:Folk {name: 'Alex'})
OPTIONAL MATCH path = (alex)<-[:MOTHER|FATHER*]-(ancestor:Folk
    )
RETURN DISTINCT ancestor.name AS ancestors;
```

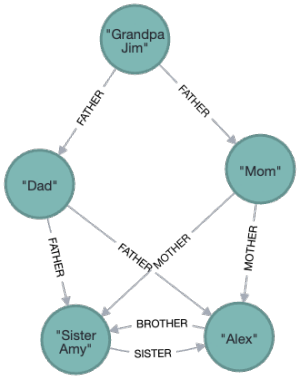
```
neo4j@neo4j> MATCH (alex:Folk {name: 'Alex'})
             OPTIONAL MATCH path = (alex)<-[:MOTHER|FATHER*]-(ancestor:Folk)
             RETURN DISTINCT ancestor.name AS ancestors;
+----------------+
| ancestors      |
+----------------+
| "Mom"          |
| "Dad"          |
| "Grandma Mary" |
| "Grandpa Bill" |
+----------------+

4 rows
ready to start consuming query after 7 ms, results consumed after another 7 ms
```

Create Grandpa Jim and make him father of mom and dad:

```
MATCH (mom:Folk {name: 'Mom'})
MATCH (dad:Folk {name: 'Dad'})

CREATE (grandpajim:Folk {name: 'Grandpa Jim'})

CREATE (grandpajim)-[:FATHER]->(mom)
CREATE (grandpajim)-[:FATHER]->(dad);
```

Get the ancestor tree of Grandpa Jim:

```
MATCH (grandpaJim:Folk {name: 'Grandpa Jim'})
MATCH path=(grandpaJim)-[*]->(ancestor)
RETURN DISTINCT ancestor as ancestortree, path;
```

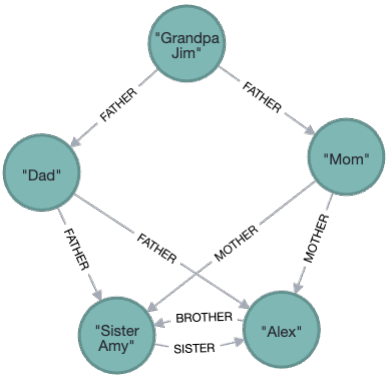


Marry Mom with Dad and Grandma with Grandpa:

```
MATCH (mom:Folk {name: 'Mom'})
MATCH (dad:Folk {name: 'Dad'})

MATCH (grandpaJim:Folk {name: 'Grandpa Jim'})
MATCH (grandmaMary:Folk {name: 'Grandma Mary'})

CREATE (mom)-[:MARRIED]->(dad)
CREATE (dad)-[:MARRIED]->(mom)

CREATE (grandpaJim)-[:MARRIED]->(grandmaMary)
CREATE (grandmaMary)-[:MARRIED]->(grandpaJim);
```

Get the ancestor tree of Grandpa Jim without the married edges

```
MATCH (grandpaJim:Folk {name: 'Grandpa Jim'})
MATCH path=(grandpaJim)-[*]->(ancestor)
WHERE NONE(rel IN relationships(path) WHERE type(rel) = '
   MARRIED')
RETURN DISTINCT ancestor AS ancestortree, path;
```



Add properties to nodes and eges

Married properties:

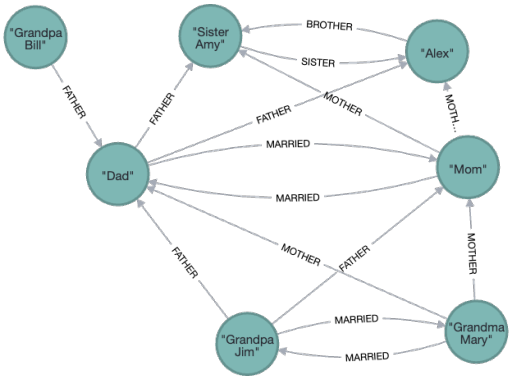
```
MATCH (mom:Folk {name: 'Mom'})
MATCH (dad:Folk {name: 'Dad'})

MATCH (grandpaJim:Folk {name: 'Grandpa Jim'})
MATCH (grandmaMary:Folk {name: 'Grandma Mary'})

MATCH (mom)-[momMarriedDad:MARRIED]->(dad)
MATCH (dad)-[dadMarriedMom:MARRIED]->(mom)
MATCH (grandpaJim)-[jimMarriedMary:MARRIED]->(grandmaMary)
MATCH (grandmaMary)-[maryMarriedJim]->(grandpaJim)

SET momMarriedDad.year = '2000'
SET dadMarriedMom.year = '2000'
SET jimMarriedMary.year = '1970'
SET maryMarriedJim.year = '1970';
```

Born Year:

```
MATCH (alex:Folk {name: 'Alex'})
MATCH (amy:Folk {name: 'Sister Amy'})

MATCH (mom:Folk {name: 'Mom'})
MATCH (dad:Folk {name: 'Dad'})

MATCH (grandpaJim:Folk {name: 'Grandpa Jim'})
MATCH (grandmaMary:Folk {name: 'Grandma Mary'})

SET alex.born_year = '2002'
SET amy.born_year = '2005'

SET mom.born_year = '1980'
SET dad.born_year = '1985'

SET grandpaJim.born_year = '1950'
SET grandmaMary.born_year = '1947';
```

Ancestor tree of persons born in year:

```
MATCH (person:Folk)
WHERE person.born_year = '2002'
MATCH path=(ancestor)-[*]->(person)
RETURN DISTINCT ancestor as ancestortree, path;
```



Find Persons married in a particular year:

```
MATCH (person:Folk)-[marriage:MARRIED]->(other:Folk)
WHERE marriage.year = '2000'
RETURN person, other;
```

Step 3:

```
MATCH
(person)-[:BORN_IN]->()-[:WITHIN*0..]->(us:Location {name:'
   United States'}),
(person)-[:LIVES_IN]->()-[:WITHIN*0..]->(eu:Location {name:'
   Europe'})
WHERE person.label = 'Person'
RETURN person AS relevantPersons;
```

```
neo4j@neo4j> MATCH
             (person)-[:BORN_IN]->()-[:WITHIN*0..]->(us:Location {name:'United States'}),
             (person)-[:LIVES_IN]->()-[:WITHIN*0..]->(eu:Location {name:'Europe'})
             WHERE person.label = 'Person'
             RETURN person AS relevantPersons;
+-------------------------------------------------------------------------------+
| relevantPersons                                                               |
+-------------------------------------------------------------------------------+
| (:Person {name: "Andrea", alias: "Andrea", label: "Person", type: "person"}) |
+-------------------------------------------------------------------------------+

1 row
ready to start consuming query after 24 ms, results consumed after another 2 ms
```

Step 4:

Since I am a cat fan, and because the names are funny I used the ancestor tree of my cat including the breeder and the new cat holder.

```
CREATE (lilly:Cat {name: 'Lilly', year_of_birth: '2022',
   gender: 'female'})
CREATE (lennox:Cat {name: 'Lennox', year_of_birth: '2022',
   gender: 'male'})
CREATE (lakota:Cat {name: 'Lakota', year_of_birth: '2022',
   gender: 'male'})

CREATE (nikanor:Cat {name: 'Nikanor Laminur', year_of_birth: '
   2020', gender: 'male'})
CREATE (fibi:Cat {name: 'Fibi Tandillo', year_of_birth: '2020'
   , gender: 'female'})

CREATE (bc:Cat {name: 'Benjamin Chemchug', year_of_birth: '
   2016', gender: 'male'})
CREATE (dt:Cat {name: 'Dashutka Tale', year_of_birth: '2016',
   gender: 'female'})

CREATE (mk:Cat {name: 'Mozart Kocilandia', year_of_birth: '
   2014', gender: 'male'})
CREATE (nm:Cat {name: 'Naomi Mizar', year_of_birth: '2016',
   gender: 'female'})

CREATE (benjamin:Person {name: 'Benjamin'})
CREATE (iris:Person {name: 'Iris'})

CREATE (rpp:Breeder {name: 'Royal Pink Paws'})

CREATE (tieber:Family {name: 'Fam. Tieber'})
CREATE (wurm:Family {name: 'Fam. Wurm'})

CREATE (lilly)-[:SISTER]->(lennox)
CREATE (lilly)-[:SISTER]->(lakota)

CREATE (lennox)-[:BROTHER]->(lilly)
CREATE (lennox)-[:BROTHER]->(lakota)

CREATE (lakota)-[:BROTHER]->(lilly)
CREATE (lakota)-[:BROTHER]->(lennox)

CREATE (nikanor)-[:FATHER]->(lilly)
CREATE (nikanor)-[:FATHER]->(lennox)
CREATE (nikanor)-[:FATHER]->(lakota)
```

```
CREATE (fibi)-[:MOTHER]->(lilly)
CREATE (fibi)-[:MOTHER]->(lennox)
CREATE (fibi)-[:MOTHER]->(lakota)

CREATE (bc)-[:FATHER]->(nikanor)
CREATE (dt)-[:MOTHER]->(nikanor)

CREATE (mk)-[:FATHER]->(fibi)
CREATE (nm)-[:MOTHER]->(fibi)

CREATE (rpp)-[:BREEDER]->(lilly)
CREATE (rpp)-[:BREEDER]->(lennox)
CREATE (rpp)-[:BREEDER]->(lakota)

CREATE (benjamin)-[:BOYFRIEND]->(iris)
CREATE (iris)-[:GIRLFRIEND]->(benjamin)

CREATE (benjamin)-[:HOLDER]->(lilly)
CREATE (iris)-[:HOLDER]->(lilly)

CREATE (rpp)-[:HOLDER]->(nikanor)
CREATE (rpp)-[:HOLDER]->(fibi)

CREATE (tieber)-[:HOLDER]->(lennox)
CREATE (wurm)-[:HOLDER]->(lakota);
```