GraphDB – example – Flights (SQL,Cypher,Gremlin)

Michal Valenta

Katedra softwarového inženýrství
Fakulta informačních technologií
České vysoké učení technické v Praze

NI-PDB, ZS 2020/21 (B201)

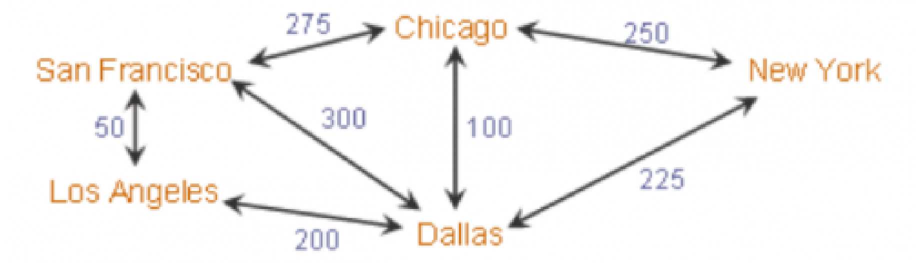
https://courses.fit.cvut.cz/NI-PDB/

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Flights example – outline

- DB description
- Implementation in SQL
  - PostgreSQL
  - using SQL standard (recursive queries)
- Implementation in Neo4j
- Implementation in Gremlin

Database


275
Chicago
250
San Francisco
New York
50
300
100
Los Angeles
225
200
Dallas

Flights – SQL – create DB

```
CREATE TABLE airports (
   code varchar(2) primary key,
   name varchar(100) );
INSERT INTO airports VALUES ('SF','San Francisco');
INSERT INTO airports VALUES ('LA','Los Angeles');
...
CREATE TABLE flights (
  flight_id varchar(3) primary key,
  start varchar(2) references airports(code),
  dest varchar(2) references airports(code),
  price integer);
INSERT INTO flights VALUES ('f1','SF','LA',50);
INSERT INTO flights VALUES ('f2','LA','SF',50);
INSERT INTO flights VALUES ('f3','SF','CH',275);
...
commit;
```

SQL – recursive queries (CTE)

Find all flights starting in NY.

```
WITH RECURSIVE trips (dest, path, total_price) AS
   ((SELECT dest, dest, price
       FROM Flights
       WHERE "start" = 'SF' )
UNION ALL
     (SELECT f.dest ,
        t.path || ',' || f.dest,
            t.total_price + f.price
     FROM Trips t, Flights f
     WHERE t.dest = f."start"))
SELECT path, total_price
FROM Trips
WHERE dest = 'NY';
```

SQL CTE (Common Table Expression)

- anchor – the first part of query
- recursive part
- endless loop – necessary to limit recursion

SQL – recursive query – limit path

Flights starting in NY, finishing in SF, max length 5.

```
WITH RECURSIVE Trips (dest, path, n_flights, total_price) AS
  (SELECT dest, "start"||','||dest , 1, price
      FROM Flights
      WHERE "start" = 'SF'
UNION ALL
   (SELECT f.dest,
         t.path || ',' || f.dest,
         t.n_flights + 1, t.total_price + f.price
    FROM Trips t, Flights f
    WHERE t.dest = f."start"
    AND f.dest <> 'SF'
    AND f."start" <> 'NY'
    AND t.n_flights < 5
    ))
SELECT path, total_price
FROM Trips
```

SQL – complet the query

From NY to SF, max length 5, the cheapest.

```
WITH RECURSIVE Trips (dest, path, n_flights, total_price) AS
  (SELECT dest, "start"||','||dest , 1, price
      FROM Flights
      WHERE "start" = 'SF'
UNION ALL
   (SELECT f.dest,
          t.path || ',' || f.dest,
          t.n_flights + 1, t.total_price + f.price
    FROM Trips t, Flights f
    WHERE t.dest = f."start"
    AND f.dest <> 'SF'
    AND f."start" <> 'NY'
    AND t. n_flights < 5
    ))
SELECT path, total_price
FROM Trips
WHERE dest = 'NY'
    AND total_price=(SELECT min(total_price)
                FROM trips
                WHERE dest='NY');
```

Flights – Neo4j – CreateDB

```
CREATE (sf {name:'San Francisco', code:'sf'}),
       (la {name:'Los Angeles', code:'la'}),
       (da {name:'Dallas', code:'da'}),
       (ch {name:'Chicago', code:'ch'}),
       (ny {name:'New York', code:'ny'}),
       (sf)-[:DIRECT {price:50}]->(la),
       (la)-[:DIRECT {price:50}]->(sf),
       (sf)-[:DIRECT {price:250}]->(ch),
       (ch)-[:DIRECT {price:250}]->(sf),
       (da)-[:DIRECT {price:300}]->(sf),
       (sf)-[:DIRECT {price:300}]->(da),
       (ch)-[:DIRECT {price:100}]->(da),
       (da)-[:DIRECT {price:100}]->(ch),
       (ch)-[:DIRECT {price:250}]->(ny),
       (ny)-[:DIRECT {price:250}]->(ch),
       (ny)-[:DIRECT {price:225}]->(da),
       (da)-[:DIRECT {price:225}]->(ny),
       (da)-[:DIRECT {price:200}]->(la),
       (la)-[:DIRECT {price:200}]->(da);
```

Neo4j – queries

Find nodes with the code SF (2 alternatives)

```
start n=node(*) where n.code='sf' return n;
match (s{code:'sf'}) return s;
```

Find all direct flights from SF (2 alternatives)

```
start s=node(*) match (s)-[:DIRECT]->d
  where s.code='sf' return s,d;
match (s{code:'sf'})-[:DIRECT]->(d) return s,d;
```

Find all flights from SF of max length 5 (display paths)

```
match path=(s{code:'sf'})-[:DIRECT*1..5]->(d)
  return extract(x in nodes(path) |x.code);
```

Neo4j – queries

Flights starting in SF, ending in NY, max length 5, display paths and prices of paths

```
match
 path=(s{code:'sf'})-[:DIRECT*1..5]->(d{code:'ny'})
return
 extract(x in nodes(path) |x.code) as total_path,
 reduce(acc=0, x in relationships(path)|acc+x.price)
   as total_price;
```

Neo4j – queries

Flights starting in SF, ending in NY, max length 5, display paths and prices of paths, order output by total price and limit to 3 cheapest

```
match
 path=(s{code:'sf'})-[:DIRECT*1..5]->(d{code:'ny'})
return
 extract(x in nodes(path) |x.code) as total_path,
 reduce(acc=0, x in relationships(path)|acc+x.price)
   as total_price
order by total_price
limit 3;
```

Gremlin – create DB

```
graph = TinkerGraph.open()
g = graph.traversal()
 sf = g.addV().property("name","San Francisco").property("code","sf").
 next()
 la = g.addV().property("name","Los Angeles").property("code","la").
 next()
 da = g.addV().property("name","Dallas").property("code","da").next()
 ch = g.addV().property("name","Chicago").property("code","ch").next()
 ny = g.addV().property("name","New York").property("code","ny").next()
 g.addE("direct").from(sf).to(la).property("price",50)
 g.addE("direct").from(la).to(sf).property("price",50)
 g.addE("direct").from(sf).to(ch).property("price",275)
 g.addE("direct").from(ch).to(sf).property("price",275)
 g.addE("direct").from(da).to(sf).property("price",300)
 g.addE("direct").from(sf).to(da).property("price",300)
 g.addE("direct").from(ch).to(da).property("price",100)
 g.addE("direct").from(da).to(ch).property("price",100)
 g.addE("direct").from(ch).to(ny).property("price",250)
 g.addE("direct").from(ny).to(ch).property("price",250)
 g.addE("direct").from(ny).to(da).property("price",225)
 g.addE("direct").from(da).to(ny).property("price",225)
 g.addE("direct").from(da).to(la).property("price",200)
 g.addE("direct").from(la).to(da).property("price",200)
```

Neo4j – queries

Find a node wich code ‘sf’

```
g.V().has('code','sf')
```

Find airport names accessed from San Francisca using one transfer.

```
g.V().has('code','sf').out('direct').out('direct').
  V().values('name')
```

First ten pathes starting in SF and ending in NY

```
g.V().has('code','sf').
  repeat(out().simplePath()).until(has('code','ny')).
 path().by('code').limit(10)
```

Gremlin – queries

Flights starting in SF, ending in NY, max length 5, display paths and prices of paths, order output by total price and limit to 3 cheapest

```
g.V().has('code','sf').
  repeat(outE().inV().simplePath()).
    until(has('code','ny')).
      project('path','total').
       by(path().by('code').by('price')).
       by(path().unfold().values('price').sum()).
  order().by(select('total')).
  limit(3)
```