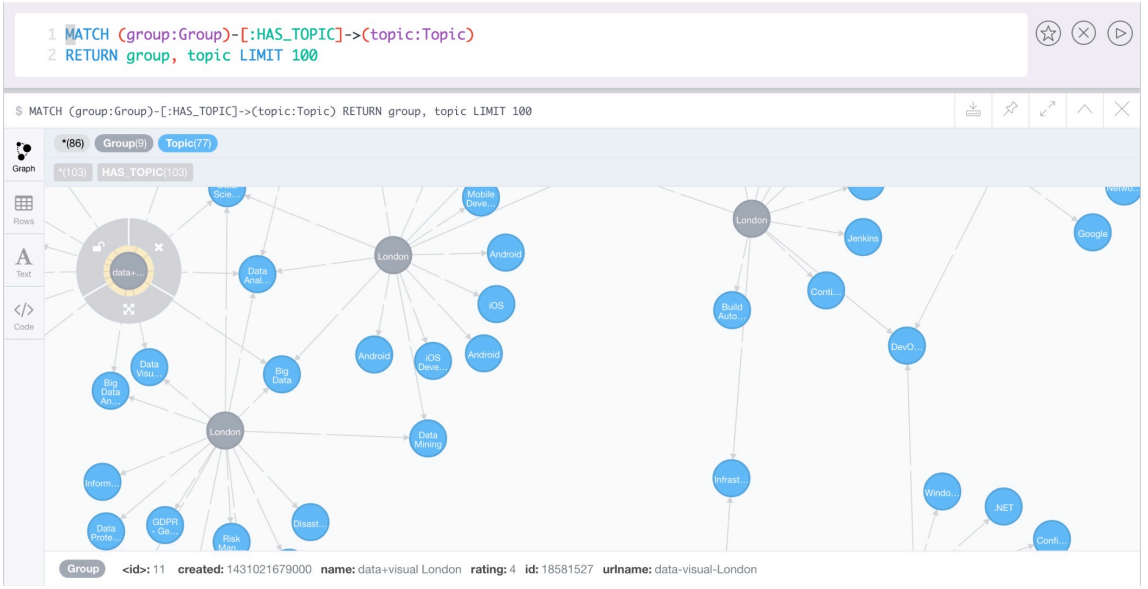
mercredi, 10 mai 2017 à 11:24 AM UTC+1

15 Finsbury Circus, Londres, Angleterre, Royaume-Uni
11°C Mostly Cloudy



Building a recommendation engine with Neo4j · GraphConnect Training

Skills Matter, CodeNode London (https://skillsmatter.com/event-space) · Pieter Cailliau · 10.05.2017 9h30–17h

Material

- Neo4j 3.1.4 (https://neo4j.com/download/)
- Neo4j Cypher Reference Card 3.1.4 (https://neo4j.com/docs/cypher-refcard/current/)
- APOC plug-in (https://github.com/neo4j-contrib/neo4j-apoc-procedures)
- Meetup groups CSV imports · available online, but we're using the subset from the provided Neo4j USB key

Guides

- Building a recommendation engine with Neo4j · Slides (https://s3-eu-west-1.amazonaws.com/neo4j-training-slides/reco.pdf) (all slides (http://s3-eu-west-1.amazonaws.com/neo4j-training-slides/index.html))
- APOC User Guide (https://neo4j-contrib.github.io/neo4j-apoc-procedures/)
- In-Neo4j browser slides:

```
:play http://guides.neo4j.com/reco/file
:play http://guides.neo4j.com/reco/file/01_similar_groups_by_topic.html
:play http://guides.neo4j.com/reco/answers/1.html
:play http://guides.neo4j.com/reco/file/02_my_similar_groups.html
:play http://guides.neo4j.com/reco/file/03_my_interests.html
:play http://guides.neo4j.com/reco/file/04_events.html
:play http://guides.neo4j.com/reco/answers/4.html
:play http://guides.neo4j.com/reco/file/05_venues.html
:play http://guides.neo4j.com/reco/answers/5a.html
:play http://guides.neo4j.com/reco/answers/5b.html
:play http://guides.neo4j.com/reco/file/06_rsvps.html
:play http://guides.neo4j.com/reco/answers/6.html
:play http://guides.neo4j.com/reco/file/07_procedures.html
:play http://guides.neo4j.com/reco/answers/7.html
:play http://guides.neo4j.com/reco/file/08_latent_social_graph.html
:play http://guides.neo4j.com/reco/answers/8.html
:play http://guides.neo4j.com/reco/file/09_scoring.html
:play http://guides.neo4j.com/reco/file/10_free_for_all.html
```

Fast indexes

Operators using indexes:

- equality
- `STARTS WITH`, `ENDS WITH`
- `CONTAINS`
- range searches
- (non-) existence checks

Database setup script

At any stage, an overview of existing node labels and relations can be visualized:

```
CALL db.schema
```

Part one · Recommend Groups by Topic

```
LOAD CSV WITH HEADERS FROM "file:///groups.csv" AS row
 RETURN row LIMIT 10

:play http://guides.neo4j.com/reco-nyc/file/01_similar_groups_by_topic.html

LOAD CSV WITH HEADERS FROM "file:///groups.csv" AS row
 CREATE (:Group { id:row.id, name:row.name,
          urlname:row.urlname, rating:toInt( row.rating),
          created:toInt( row.created) })

MATCH (g:Group) RETURN g.id, g.name, g.urlname

CREATE CONSTRAINT ON (t:Topic) ASSERT t.id IS UNIQUE
CREATE CONSTRAINT ON (g:Group) ASSERT g.id IS UNIQUE
:schema
CALL db.constraints()

LOAD CSV WITH HEADERS FROM "file:///groups_topics.csv" AS row
 RETURN row LIMIT 10

LOAD CSV WITH HEADERS FROM "file:///groups_topics.csv" AS row
 MERGE (topic:Topic {id: row.id})
 ON CREATE SET topic.name = row.name, topic.urlkey = row.urlkey

MATCH (t:Topic) RETURN t.id, t.name

LOAD CSV WITH HEADERS FROM "file:///groups_topics.csv"  AS row
 MATCH (topic:Topic {id: row.id})
 MATCH (group:Group {id: row.groupId})
 MERGE (group)-[:HAS_TOPIC]->(topic)

MATCH (group:Group)-[:HAS_TOPIC]->(topic:Topic)
 RETURN group, topic LIMIT 10

CREATE INDEX ON :Group(name)
CREATE INDEX ON :Topic(name)
CALL db.indexes()
```

Part two · Groups similar to mine

```
LOAD CSV WITH HEADERS FROM "file:///members.csv" AS row
RETURN row LIMIT 10

CREATE CONSTRAINT ON (m:Member) ASSERT m.id IS UNIQUE

USING PERIODIC COMMIT 10000
 LOAD CSV WITH HEADERS
 FROM "file:///members.csv" AS row
 WITH DISTINCT row.id AS id, row.name AS name
 MERGE (member:Member {id: id})
 ON CREATE SET member.name = name

USING PERIODIC COMMIT 50000
 LOAD CSV WITH HEADERS
 FROM "file:///members.csv" AS row
 WITH row WHERE NOT row.joined is null
 MATCH (member:Member {id: row.id})
 MATCH (group:Group {id: row.groupId})
 MERGE (member)-[membership:MEMBER_OF]->(group)
 ON CREATE SET membership.joined=toInt(row.joined);
 // Set 408557 properties, created 408557 relationships, statement completed in 126404 ms.

CREATE INDEX ON :Member(name)
```

Part three · Member Interests (Topics)

```
LOAD CSV WITH HEADERS FROM "file:///members.csv" AS row
RETURN row.id, row.topics LIMIT 10

USING PERIODIC COMMIT 50000
 LOAD CSV WITH HEADERS FROM "file:///members.csv" AS row

 WITH split(row.topics, ";") AS topics, row.id AS memberId
 UNWIND topics AS topicId

 WITH DISTINCT memberId, topicId
 MATCH (member:Member {id: memberId})
 MATCH (topic:Topic {id: topicId})
 MERGE (member)-[:INTERESTED_IN]->(topic)
 // Created 1079735 relationships, statement completed in 219249 ms.

MATCH (m:Member)-[:MEMBER_OF]->()-[:HAS_TOPIC]->(topic)
 WHERE NOT (m)-[:INTERESTED_IN]->(topic)

 WITH m, topic, COUNT(*) AS times
 WHERE times >= 3

 MERGE (m)-[interestedIn:INTERESTED_IN]->(topic)
 SET interestedIn.inferred = true
 // Set 195000 properties, created 195000 relationships, statement completed in 67565 ms.
```

Part four · Event Recommendations

```
LOAD CSV WITH HEADERS FROM "file:///events.csv" AS row
 RETURN row LIMIT 5

CREATE CONSTRAINT ON (e:Event) ASSERT e.id IS UNIQUE
CREATE INDEX ON :Event(time)

USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM "file:///events.csv" AS row
MERGE (event:Event {id: row.id})
ON CREATE SET event.name = row.name,
        event.time = toInt(row.time),
        event.utcOffset = toInt(row.utc_offset)
// Added 10074 labels, created 10074 nodes, set 40296 properties, statement completed in 3734 ms.

MATCH (event:Event) RETURN event LIMIT 10

USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM "file:///events.csv" AS row

WITH distinct row.group_id as groupId, row.id as eventId
MATCH (group:Group {id: groupId})
MATCH (event:Event {id: eventId})
MERGE (group)-[:HOSTED_EVENT]->(event)
// Created 10074 relationships, statement completed in 2694 ms.

MATCH (group:Group)-[hosted:HOSTED_EVENT]->(event)
WHERE group.name STARTS WITH 'Neo4j' AND event.time < timestamp()
RETURN event, group, hosted
ORDER BY event.time DESC LIMIT 10
```

Part five · Venues

```
LOAD CSV WITH HEADERS FROM "file:///venues.csv" AS row
RETURN row LIMIT 10

CREATE CONSTRAINT ON (v:Venue)
ASSERT v.id IS UNIQUE

LOAD CSV WITH HEADERS FROM "file:///venues.csv" AS row
MERGE (venue:Venue {id: row.id})
ON CREATE SET venue.name = row.name,
        venue.latitude = tofloat(row.lat),
        venue.longitude = tofloat(row.lon),
        venue.address = row.address_1
// Added 400 labels, created 400 nodes, set 2000 properties, statement completed in 260 ms.

LOAD CSV WITH HEADERS FROM "file:///events.csv" AS row
MATCH (venue:Venue {id: row.venue_id})
MATCH (event:Event {id: row.id})
MERGE (event)-[:VENUE]->(venue)
// Created 2023 relationships, statement completed in 853 ms

MATCH (venue:Venue)
WHERE EXISTS(venue.latitude) AND EXISTS(venue.longitude)
RETURN COUNT(*)
// COUNT(*): 1706
```

Part six · RSVPs

```
LOAD CSV WITH HEADERS FROM "file:///rsvps.csv" AS row
RETURN row LIMIT 10

USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM "file:///rsvps.csv" AS row
WITH row WHERE row.response = "yes"

MATCH (member:Member {id: row.member_id})
MATCH (event:Event {id: row.event_id})
MERGE (member)-[rsvp:RSVPD {id: row.rsvp_id}]->(event)
ON CREATE SET rsvp.created = toint(row.created),
        rsvp.lastModified = toint(row.mtime),
        rsvp.guests = toint(row.guests)
// Set 1472664 properties, created 368166 relationships, statement completed in 49032 ms.

MATCH (event:Event)<-[rsvp:RSVPD]-(attendee)
WHERE event.name CONTAINS 'An Event'
RETURN *
```

Part seven · Procedures

```
CALL dbms.procedures()

CREATE CONSTRAINT ON (p:Photo) ASSERT p.id IS UNIQUE

CALL apoc.load.json("https://api.meetup.com/graphdb-london/photos?&sign=true&photo-host=public")
YIELD value AS document

WITH document WHERE document.photo_album.event.id IS NOT NULL

MATCH (event:Event {id: document.photo_album.event.id})
OPTIONAL MATCH (member:Member {id: tostring(document.member.id)})

MERGE (photo:Photo {id: document.id})
ON CREATE SET photo.link = document.link, photo.created = document.created

MERGE (photo)-[:POSTED_IN_EVENT]->(event)

WITH photo, member WHERE  member is NOT NULL

MERGE (member)-[:POSTED_PHOTO]->(photo)
// Added 189 labels, created 189 nodes, set 567 properties, created 372 relationships, statement completed in 1050 ms.
```

Part eight · Latent social graph & transactions

Materialising the latent social graph – a first chunk of 500

```
MATCH (m:Member) WHERE size( (m)-[:RSVPD]->() ) >= 7
SET m:Process
RETURN count(*)
// COUNT(*): 12659

MATCH (m1:Member:Process) WITH m1 LIMIT 500
REMOVE m1:Process
WITH m1
MATCH (m1)-[:RSVPD]->(event)<-[:RSVPD]-(m2)

WITH m1, m2, COUNT(*) AS times
WHERE times >= 5

MERGE (m1)-[:FRIENDS]-(m2)
RETURN count(*);
// COUNT(*): 67826
```

Materialising the latent social graph – transacting chunks of 500 relations periodically

```
call apoc.periodic.commit("
  MATCH (m1:Member:Process) WITH m1 LIMIT {limit} REMOVE m1:Process
  WITH m1 MATCH (m1)-[:RSVPD]->(event)<-[:RSVPD]-(m2)
  WITH m1, m2, COUNT(*) AS times WHERE times >= 5
  MERGE (m1)-[:FRIENDS]-(m2) RETURN count(*)
",{limit:500})
// Started streaming 1 record after 126510 ms and completed after 126510 ms.

MATCH ()-[:FRIENDS]-() RETURN COUNT(*)
// COUNT(*): 623584
```

Part nine · Scoring recommendations

Scoring our friends – tag all members that we're going to process

```
MATCH (m:Member)-[:FRIENDS]-()
SET m:Process
RETURN count(DISTINCT m)
// COUNT( DISTINCT m): 13095
```

Adding a score to the FRIENDS relationship

```
call apoc.periodic.commit("
  MATCH (m1:Process)

  WITH m1 LIMIT {limit}
  REMOVE m1:Process

  WITH m1
  MATCH (m1)-[friendship:FRIENDS]-(m2:Member)

  WITH m1, m2, friendship
  MATCH (m1)-[:RSVPD]->(commonEvent)<-[:RSVPD]-(m2)

  WITH m1, m2, friendship, COUNT(commonEvent) AS commonEvents
  WITH m1, m2, friendship, commonEvents, SIZE((m1)-[:RSVPD]->()) AS m1Rsvps, SIZE((m2)-[:RSVPD]->()) AS m2Rsvps
  WITH m1, m2, friendship, commonEvents, m1Rsvps, m2Rsvps, (2 * 1.0 * commonEvents) / (m1Rsvps + m2Rsvps) AS diceSimilarity

  SET friendship.score = diceSimilarity

  RETURN COUNT(*)
",{limit:500})
// Started streaming 1 record after 389120 ms and completed after 389120 ms.
```

Exercises

At any stage, a view of the query execution plan can be visualized:

```
PROFILE MATCH ...
```

Part one · Recommend Groups by Topic

Most popular topic?

```
MATCH (t:Topic)<-[:HAS_TOPIC]-()
RETURN t.name, COUNT(*) AS count
ORDER BY count DESC
```

Group created most recently?

```
MATCH (group:Group) RETURN group
ORDER BY group.created DESC LIMIT 1
```

How much groups have been running since more than 4 years?

```
MATCH (g:Group)
WHERE (timestamp() - g.created) / 1000 / 3600 / 24 / 365 >= 4
RETURN count(g)
```

Find groups with 'Neo4j' or 'Data' in their name

```
MATCH (g:Group)
WHERE g.name CONTAINS 'Neo4j' OR g.name CONTAINS 'Data'
RETURN g
```

What are the distinct topics for those groups ?

```
MATCH (g:Group)-[:HAS_TOPIC]->(t:Topic)
WHERE g.name CONTAINS 'Neo4j' OR g.name CONTAINS 'Data'
RETURN t.name, count(*)
```

Find groups that have the same topics as the Neo4j Meetup group

```
MATCH (group:Group)
WHERE (group.name CONTAINS 'Graph Database' OR group.name CONTAINS 'Neo4j')

MATCH (group)-[:HAS_TOPIC]->(topic)<-[:HAS_TOPIC]-(otherGroup)
RETURN otherGroup.name, COUNT(topic) AS topicsInCommon,
    COLLECT(topic.name) AS topics
ORDER BY topicsInCommon DESC, otherGroup.name
LIMIT 10
```

Part two · Groups similar to mine

Exclude groups I'm a member of

```
MATCH (member:Member) WHERE member.name = 'Pieter Ennes'
MATCH (group:Group)
WHERE group.name CONTAINS 'Graph Database' OR group.name CONTAINS 'Neo4j'

MATCH (group)-[:HAS_TOPIC]->(topic)<-[:HAS_TOPIC]-(otherGroup:Group)
WHERE NOT EXISTS( (member)-[:MEMBER_OF]->(otherGroup) )
RETURN otherGroup.name,
    COUNT(topic) AS topicsInCommon,
    COLLECT(topic.name) AS topics
ORDER BY topicsInCommon DESC
LIMIT 10
```

Find my similar groups

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Ennes'
MATCH (member)-[:MEMBER_OF]->()-[:HAS_TOPIC]->()<-[:HAS_TOPIC]-(otherGroup:Group)
WHERE NOT EXISTS ((member)-[:MEMBER_OF]->(otherGroup))
RETURN otherGroup.name,
    COUNT(*) AS topicsInCommon
ORDER BY topicsInCommon DESC
LIMIT 10
```

Find people who are members of the most groups

```
MATCH (member:Member)-[:MEMBER_OF]->()
WITH member, COUNT(*) AS groups
ORDER BY groups DESC
LIMIT 10
RETURN member.name, groups
```

Part three • Member Interests (Topics)

Find my similar groups – using interests

```
MATCH (member:Member {name: 'Pieter Cailliau'})-[:INTERESTED_IN]->(topic),
    (member)-[:MEMBER_OF]->(group)-[:HAS_TOPIC]->(topic)
WITH member, topic, COUNT(*) AS score
MATCH (topic)<-[:HAS_TOPIC]-(otherGroup)
WHERE NOT (member)-[:MEMBER_OF]->(otherGroup)
RETURN otherGroup.name, COLLECT(topic.name), SUM(score) as score
ORDER BY score DESC
LIMIT 10
```

Find someone who didn't specify any interests but is member of a few groups

```
MATCH (member:Member) WHERE NOT (member)-[:INTERESTED_IN]->()
MATCH (member)-[:MEMBER_OF]->()
RETURN member.name, count(*) as groups
ORDER BY count(*) DESC
LIMIT 50
```

Someone from that list who didn't specify any interests when they signed up for meetup.com

```
MATCH (member:Member {name: 'Pieter Cailliau'})-[:INTERESTED_IN]->(topic), (member)-[:MEMBER_OF]->(group)-[:HAS_TOPIC]->(topic)
WITH member, topic, COUNT(*) AS score
MATCH (topic)<-[:HAS_TOPIC]-(otherGroup)
WHERE NOT (member)-[:MEMBER_OF]->(otherGroup)
RETURN otherGroup.name, COLLECT(topic.name), SUM(score) as score
ORDER BY score DESC
```

Persisting inferred interests

```
MATCH (m:Member)-[:MEMBER_OF]->()-[:HAS_TOPIC]->(topic)
WHERE NOT (m)-[:INTERESTED_IN]->(topic)

WITH m, topic, COUNT(*) AS times
WHERE times >= 3

MERGE (m)-[interestedIn:INTERESTED_IN]->(topic)
SET interestedIn.inferred = true
```

Persisting inferred interests with a production sized dataset – in batches

```
CALL apoc.periodic.iterate(
"MATCH (m:Member)-[:MEMBER_OF]->()-[:HAS_TOPIC]->(topic)
WHERE NOT (m)-[:INTERESTED_IN]->(topic)

WITH m, topic, COUNT(*) AS times
WHERE times >= 3
RETURN m,topic",
"MERGE (m)-[interestedIn:INTERESTED_IN]->(topic)
SET interestedIn.inferred = true",
{batchSize:1000, iterateList:true, parallel:true})
```

Find my similar groups (even if I don't know it yet)

```
MATCH (member:Member)-[:INTERESTED_IN]->(topic),
    (member)-[:MEMBER_OF]->(group)-[:HAS_TOPIC]->(topic)
WHERE member.name CONTAINS 'Pieter Cailliau'
WITH member, topic, COUNT(*) AS score
MATCH (topic)<-[:HAS_TOPIC]-(otherGroup)
WHERE NOT (member)-[:MEMBER_OF]->(otherGroup)
RETURN otherGroup.name, COLLECT(topic.name), SUM(score) as score
ORDER BY score DESC
```

Part four · Event Recommendations

Find future events in my groups

```
MATCH (member:Member)-[:MEMBER_OF]->(group)-[:HOSTED_EVENT]->(futureEvent)
WHERE member.name CONTAINS 'Pieter Cailliau' AND futureEvent.time > timestamp()
RETURN group.name,
    futureEvent.name,
    round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS days
ORDER BY days
```

Find future events for my topics

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event) WHERE futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
MATCH (futureEvent)<-[:HOSTED_EVENT]-(group)

RETURN futureEvent.name, futureEvent.time, group.name, commonTopics, myGroup
ORDER BY futureEvent.time
```

Filter out events which have less than 3 common topics

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event) WHERE futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3

MATCH (futureEvent)<-[:HOSTED_EVENT]-(group)

RETURN futureEvent.name, futureEvent.time, group.name, commonTopics, myGroup
ORDER BY futureEvent.time
```

Filter out events which have less than 3 common topics

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3

MATCH (futureEvent)<-[:HOSTED_EVENT]-(group)

RETURN futureEvent.name, futureEvent.time, group.name, commonTopics, myGroup
ORDER BY futureEvent.time
```

Sorting the results

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
MATCH (futureEvent)<-[:HOSTED_EVENT]-(group)

WITH futureEvent, group, commonTopics, myGroup, CASE WHEN myGroup THEN 5 ELSE 0 END AS myGroupScore
WITH futureEvent, group, commonTopics, myGroup, myGroupScore, round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, commonTopics, myGroup, days, myGroupScore + commonTopics - days AS score
ORDER BY score DESC
LIMIT 10
```

Part five · Venues

Distance to venues

```
WITH point({latitude: 51.518698, longitude: -0.086146}) AS trainingVenue
MATCH (venue:Venue)

WITH venue, distance(point(venue), trainingVenue) AS distance
RETURN venue.id, venue.name, venue.address, distance
ORDER BY distance LIMIT 10
```

Return distance to venue

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)
WITH futureEvent, group, venue, commonTopics, myGroup, distance(point(venue), point({latitude: 51.518698, longitude: -0.086146})) AS distance
WITH futureEvent, group, venue, commonTopics, myGroup, distance, CASE WHEN myGroup THEN 5 ELSE 0 END AS myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, distance, myGroupScore, round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS
days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, days, distance , myGroupScore + commonTopics - days
AS score
ORDER BY score DESC
```

Filter out events < 1km away

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH  futureEvent, group, venue,commonTopics, myGroup, distance(point(venue), point({latitude: 51.518698, longitude: -0.086146})) AS distance
WHERE distance < 1000

WITH futureEvent, group, venue, commonTopics, myGroup, distance, CASE WHEN myGroup THEN 5 ELSE 0 END AS myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, distance, myGroupScore, round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS
days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, days, distance, myGroupScore + commonTopics - days
AS score
ORDER BY score DESC
```

Part six · RSVPs

Our previous RSVPS – add a score for an event based on previous events we've attended in that group

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3

OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents

MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance(point(venue), point({latitude: 51.518698, longitude: -0.086146})) AS
distance
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, CASE WHEN myGroup THEN 5 ELSE 0 END AS myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, myGroupScore, round((futureEvent.time - timestamp()) /
(24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, days, distance, myGroupScore +
commonTopics - days AS score
ORDER BY score DESC
```

Events at my venue

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH member, futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents
MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH member, futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance(point(venue), point({latitude: 51.518698, longitude:
-0.086146})) AS distance
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)-[:VENUE]->(venue)
WHERE previousEvent.time < timestamp()

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, COUNT(previousEvent) AS eventsAtVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, eventsAtVenue, CASE WHEN myGroup THEN 5 ELSE 0 END AS
myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, eventsAtVenue, myGroupScore, round((futureEvent.time -
timestamp()) / (24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, days, distance, eventsAtVenue,
myGroupScore + commonTopics + eventsAtVenue - days AS score
ORDER BY score DESC
```

Events near my venues

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH member, futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents
MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH member, futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance(point(venue), point({latitude: 51.518698, longitude:
-0.086146})) AS distance
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)-[:VENUE]->(aVenue)
WHERE previousEvent.time < timestamp() AND abs(distance(point(venue), point(aVenue))) < 500

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, COUNT(previousEvent) AS eventsNearVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, eventsNearVenue, CASE WHEN myGroup THEN 5 ELSE 0 END AS
myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, distance, eventsNearVenue, myGroupScore, round((futureEvent.time -
timestamp()) / (24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, days, distance, eventsNearVenue,
myGroupScore + commonTopics + eventsNearVenue - days AS score
ORDER BY score DESC
```

Part seven · Procedures

Using a procedure's output in a query

```
CALL db.labels() YIELD label AS label
RETURN label ORDER BY label
```

Check APOC installed correctly

```
CALL dbms.procedures() YIELD name AS name, signature AS signature
WITH name, signature
WHERE name STARTS WITH "apoc"
RETURN name, signature
```

Formatting timestamps

```
MATCH (venue)<-[:VENUE]-(event:Event)<-[:HOSTED_EVENT]-(group:Group)
WHERE event.time < timestamp()
WITH event, venue, group
ORDER BY event.time DESC
LIMIT 5
WITH event, group, venue, apoc.date.format(event.time, 'ms') as dateTime
RETURN event.name, group.name, venue.name, dateTime
```

Query the procedures list to retrieve signature of a procedure

```
CALL dbms.procedures() YIELD name AS name, signature AS signature
WITH name, signature
WHERE name = "apoc.load.json"
RETURN name, signature
```

Importing JSON from the meetup.com API

```
CALL apoc.load.json("https://api.meetup.com/graphdb-london/photos?&sign=true&photo-host=public")
YIELD value AS document
WITH document WHERE EXISTS(document.photo_album.event.id)
RETURN document.link AS link,
    document.created AS time,
    document.id AS photoId,
    document.member.id as memberId,
    document.photo_album.event.id AS eventId
// Started streaming 189 records after 1 ms and completed after 1256 ms.
```

Note: the API above uses the `urlname` property to look up photos for a group.
To find the `urlname` for other groups:

```
MATCH (group:Group)
RETURN group.urlname
ORDER BY rand() LIMIT 10
```

Part eight · Latent social graph + transactions

Finding people that I know

```
MATCH (me:Member)-[:RSVPD]->()<-[:RSVPD]-(otherPerson)
WHERE me.name CONTAINS 'Pieter Cailliau'
WITH otherPerson, COUNT(*) AS commonEvents
ORDER BY commonEvents DESC LIMIT 10
RETURN otherPerson.name, commonEvents
```

Materialising the latent social graph

```
// See the Setup database section above.
```

How many of our friends have RSVPD?

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH member, futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents

OPTIONAL MATCH (member)-[:FRIENDS]-(:Member)-[rsvpFriend:RSVPD]->(futureEvent)
WITH member, futureEvent, commonTopics, myGroup, previousEvents, COUNT(rsvpFriend) AS friendsGoing

MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH member, futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, distance(point(venue), point({latitude: 51.518698,
longitude: -0.086146})) AS distance
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)-[:VENUE]->(aVenue)
WHERE previousEvent.time < timestamp() AND abs(distance(point(venue), point(aVenue))) < 500

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, distance, COUNT(previousEvent) AS eventsAtVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, distance, eventsAtVenue, CASE WHEN myGroup THEN 5 ELSE
0 END AS myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, distance, eventsAtVenue, myGroupScore,
round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, friendsGoing, days, distance,
eventsAtVenue, myGroupScore + commonTopics + eventsAtVenue + (friendsGoing / 2.0) - days AS score
ORDER BY score DESC
```

Who are these friends?

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'

MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH member, futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents

OPTIONAL MATCH (member)-[:FRIENDS]-(friend:Member)-[rsvpFriend:RSVPD]->(futureEvent)
WITH member, futureEvent, commonTopics, myGroup, previousEvents, COUNT(rsvpFriend) AS friendsGoing, COLLECT(friend.name) AS friends

MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH member, futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance(point(venue), point({latitude:
51.518698, longitude: -0.086146})) AS distance
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)-[:VENUE]->(aVenue)
WHERE previousEvent.time < timestamp() AND abs(distance(point(venue), point(aVenue))) < 500

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, COUNT(previousEvent) AS eventsAtVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, eventsAtVenue, CASE WHEN myGroup THEN
5 ELSE 0 END AS myGroupScore
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, eventsAtVenue, myGroupScore,
round((futureEvent.time - timestamp()) / (24.0*60*60*1000)) AS days

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, friendsGoing, friends[..5], days,
distance, eventsAtVenue, myGroupScore + commonTopics + eventsAtVenue + (friendsGoing / 2.0) - days AS score
ORDER BY score DESC
```

Part nine • Scoring recommendations

The Pareto function

```
CALL dbms.functions()
YIELD name AS name, signature AS signature, description AS description
WHERE name = "apoc.scoring.pareto"
RETURN signature, description

UNWIND range( 0,21) AS value
RETURN value, apoc.scoring.pareto( 1,10,20,value) AS score
```

Scoring with Pareto

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (futureEvent:Event)
WHERE timestamp() + (7 * 24 * 60 * 60 * 1000) > futureEvent.time > timestamp()

WITH member, futureEvent, EXISTS((member)-[:MEMBER_OF]->()-[:HOSTED_EVENT]->(futureEvent)) AS myGroup
OPTIONAL MATCH (member)-[:INTERESTED_IN]->()<-[:HAS_TOPIC]-()-[:HOSTED_EVENT]->(futureEvent)

WITH member, futureEvent, myGroup, COUNT(*) AS commonTopics
WHERE commonTopics >= 3
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)<-[:HOSTED_EVENT]-()-[:HOSTED_EVENT]->(futureEvent)
WHERE previousEvent.time < timestamp()

WITH member, futureEvent, commonTopics, myGroup, COUNT(rsvp) AS previousEvents

OPTIONAL MATCH (member)-[:FRIENDS]-(friend:Member)-[rsvpFriend:RSVPD]->(futureEvent)
WITH member, futureEvent, commonTopics, myGroup, previousEvents, COUNT(rsvpFriend) AS friendsGoing, COLLECT(friend.name) AS friends

MATCH (venue)<-[:VENUE]-(futureEvent)<-[:HOSTED_EVENT]-(group)

WITH member, futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance(point(venue), point({latitude:
51.518698, longitude: -0.086146})) AS distance
OPTIONAL MATCH (member)-[rsvp:RSVPD]->(previousEvent)-[:VENUE]->(aVenue)
WHERE previousEvent.time < timestamp() AND abs(distance(point(venue), point(aVenue))) < 500

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, COUNT(previousEvent) AS eventsAtVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, eventsAtVenue
WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, eventsAtVenue,
toint(round((futureEvent.time - timestamp()) / (24.0*60*60*1000))) AS days

WITH futureEvent, group, venue, commonTopics, myGroup, previousEvents, friendsGoing, friends, distance, eventsAtVenue, days,
apoc.scoring.existence(5, myGroup) AS myGroupScore,
apoc.scoring.pareto(1, 3, 10, days) AS daysScore,
apoc.scoring.pareto(1, 5, 10, commonTopics) AS topicsScore,
apoc.scoring.pareto(1, 7, 20, eventsAtVenue) AS eventsAtVenueScore,
apoc.scoring.pareto(1, 5, 20, friendsGoing)  AS friendsGoingScore

RETURN futureEvent.name, futureEvent.time, group.name, venue.name, commonTopics, myGroup, previousEvents, friendsGoing, friends[..5], days,
distance, eventsAtVenue, myGroupScore + topicsScore + eventsAtVenueScore + friendsGoingScore - daysScore AS score
ORDER BY score DESC
```

Scoring our friendships – find the top 10 people similar to you, calculating a « dice similarity »

```
MATCH (m1:Member) WHERE m1.name CONTAINS 'Pieter Cailliau'

MATCH (m1)-[friendship:FRIENDS]-(m2:Member)
WITH m1, m2, friendship
MATCH (m1)-[:RSVPD]->(commonEvent)<-[:RSVPD]-(m2)
WITH m1, m2, COUNT(commonEvent) AS commonEvents
WITH m1, m2, commonEvents, SIZE((m1)-[:RSVPD]->()) AS m1Rsvps, SIZE((m2)-[:RSVPD]->()) AS m2Rsvps
RETURN m1.name, m2.name, commonEvents, m1Rsvps, m2Rsvps, (2 * 1.0 * commonEvents) / (m1Rsvps + m2Rsvps) AS diceSimilarity
ORDER BY diceSimilarity DESC
LIMIT 10
```

Friendship-based recommendations – Find the events that our best 10 friends are planning to attend

Let's get away from the mega event recommendation query we've built up over the day and do some recommendations based purely on our best Meetup friendships.

```
MATCH (member:Member) WHERE member.name CONTAINS 'Pieter Cailliau'
MATCH (member)-[friendship:FRIENDS]-(friend)
```

```
WITH member, friend, friendship
ORDER By friendship.score DESC
LIMIT 10
MATCH (friend)-[:RSVPD]->(futureEvent)<-[:HOSTED_EVENT]-(group)
WHERE futureEvent.time > timestamp()
RETURN futureEvent.name, group.name, COUNT(*) AS friendsGoing, COLLECT(friend.name) AS friends
ORDER BY friendsGoing DESC
```

Dice similarity is just one of the similarity metrics that we could have used. There's also Jaccard, cosine and overlap to name just a few.

Part ten • Your turn

What else can we add to the model to come up with more interesting recommendations? Here you have some time to play around with the data and come up with something new

Here are some ideas that we thought of:

- *Merge duplicate venues* – can we write a query that finds duplicate venues?
- *Social network* – what events do our twitter/Facebook friends attend? Can we import that data and use it?
- *Topic ontology* – how are topics related? Can you find an ontology that we could encode in the graph?
- *Event similarity* based on descriptions - use Latent Dirichlet Allocation to derive categories
- *Day of the week* – do we only go to events on certain days of the week? do we go to different events on weekdays vs weekend?
- meetup.com provides several *streaming APIs* - **http://www.meetup.com/meetup_api/docs/stream/2/rsvps/#polling** - can we use those to update the graph on the fly?

Or if none of those appeal you can try something of your own.

If we have time we'll let a few people show the group what they've come up with.

Noia | Notes de lecture