TODO REPORT/IMPLEMENTATION SIDE

1. - Look whether coalescence exists
2. - Translate Interactive Complex 14
3. Rename the ToDo's in label queries
4. Add labels for the interactive queries
5. Add tables for the interactive queries
6. Look whether the composable graph systems is possible for in GQL

Interactive Queries

Business Intelligence Queries

ADD the following QUERIES :

1. BI queries9,10,15 16
2. Add the returns size in each table for the Bi queries
3. Add the return size for the interactive queries(Same for the interactive queries)
4. Fix Bi17/ Write it out

1 Introduction

1. write about :
2. The future of graph databases
3. Why the need for graph databases
4. What is the need for a common language
5. What is the need for a benchmark
6. Give an overview of the remaining chapter
7. Motivation

2 Regular Path Queries

WRITE ABOUT THE Book chapter 2 and 3 of the Book WRITE ABOUT GQL PAPER FROM DR MINGXI()

3 GQL Semnantics

WRITE ABOUT THE paper of MINGXI

4 LDBC community

WRITE ABOUT THE PAPER FROM TigerGRAPH GSQL

Maybe not necessary to write about the Graphs/Datasets ?

5 Conversion from GSQL to GQL

```
createQuery := CREATE [OR REPLACE] [DISTRIBUTED] QUERY queryName
               "(" [parameterList] ")"
               [FOR GRAPH graphName]
               [RETURNS "("  baseType | accumType ")"]
               [API ( v2 )]
               [SYNTAX syntaxName]
               "{" queryBody "}"
```

Listing 1: GSQL query overview

```
The GQL query in Cypher Style

FROM graphName
MATCH  match traversal from node 1 to node n.
WHERE
AND
RETURN
```

Listing 2: GQL

In the FROM clause of the GQL query the graphname is specified. Do note that we do not have to create a specific query with the function create query in GQL.

However in GSQL in most queries a temp variable is used. This is done to indicate that there is a temporary variable, or helper variable. The helper variable is also used for the accumulators. Since GQL (or at least the current version of it) does not support, helper views are constructed, and the MATCH clause is heavily utilized.

Since most queries of both the LDBC Interactive and Business Intelligence graph, are constructed or the results obtained or stored are through accumulators, GQL mainly uses the MATCH clause to traverse the path and return the results.Contrary to GSQL where the results are stored in a list like notation such as [value1 value2, value3,value4 ...value], GQL only has the RETURN Statement.

During translation it was noticeable that the GSQL queries consist of query blocks, that are in sequence dependant on each other, through the temp variable and accumulators.As a workaround for that, the views are constructed and the consequent view specifies the next view in the FROM clause as depicted in 110

CREATE QUERY query_name_VIEW **AS** (
FROM <LDBC_SNB|LDBC_INTERACTIVE>
MATCH (node_name:node1)-[label_predicate:label]-(node_name:node2)
RETURN <param>
CONDITIONALS : **WHERE** |**AND**|**OR**
AGGREGATION FUNCTIONS : **HAVING**|**GROUP BY**|**ORDER BY**
lstlistingGQL Query view sequence1

CREATE QUERY query_name_VIEW **AS** (
FROM <>
MATCH (node_name:node1)-[label_predicate:label]-(node_name:node2)
RETURN <param>
CONDITIONALS : **WHERE** |**AND**|**OR**
AGGREGATION FUNCTIONS : **HAVING**|**GROUP BY**|**ORDER BY** lstlistingGQL Query view sequence 2

The Match clause of the GSQL query is slightly different than that of the GQL. In GQL the predicate name comes before the node or label name as shown in 110

textbfMATCH (node_name:node1)-[label_predicate:label]-(node_name:node2)

6 Parser/Validity Testing

7 Discussion

8 Future Work

9 Appendix

9.1 Business Intelligence queries

In the following subsections the business intelligence queries are translated or converted towards the Cypher style query.

9.1.1 Business Intelligence 1

In 3 the message node is selected by the MATCH clause. After selection a verification is executed

```
CREATE QUERY messageCount_VIEW AS {
FROM message
MATCH (message:Message)
WHERE message.creationDate < $datetime
Return count(message) AS totalMessageCountInt}
```

3: message count

by the WHERE clause. The verification is based on whether the creation date of the message is lower than the provided date. At last, the count of the messages are returned as totalMessage Count.

```
CREATE  QUERY messagePrep_View AS (
From message
MATCH(message:message)
WHERE  creationDate < :datetime
    AND content IS NOT NULL
   SELECT extract(year from creationDate) AS messageYear
         , ParentMessageId IS NOT NULL AS isComment

Return ParentMessageId IS NOT NULL AS isComment,
       , CASE
            WHEN length <  40 THEN 0 -- short
            WHEN length <  80 THEN 1 -- one liner
            WHEN length < 160 THEN 2 -- tweet
            ELSE                   3 -- long
          END AS lengthCategory
         , length
)
```

4: message prep

In 4, the MATCH clause selects the message node. Although the query construction might seem similar to **??**, it is quite different. After the match clause the where clause validates whether the original date of the message is lower than the actual date. Moreover, there is another verification carried out, which verifies to see whether there is content or not. At last, the id of the messages is returned with their respective length category.

The last query is the final query and uses the returned results from **??** and 4. Also here first the message node is selected and returns the results, which are grouped by the year, the length of the category, and the id of the parent message (variable: isComment).

The variables which are returned

9.1.2 Business Intelligence 2

In 9the message node is first selected through the optional match. After a selection of that node, a right path traversal is done towards the node tag along the path `HAS_TAG`.The where clause, which is present together with an AND clause, verifies whether the message is newer than the provided date, but it should not be older than 100 days. At last, the count of the messages is returned under the variable name countWindow1.

```
FROM messagePrep_VIEW, messageCount_VIEW
MATCH(message:message)
Return messageYear,
       isComment,
       lengthCategory,
       count(message) AS messageCount,
       sum(message.length)/toFloat (count(message)) AS averageMessageLength,
       sum(length) AS sumMessageLength,
       messageCount / toFloat(totalMessageCountInt) AS percentageOfMessages

 GROUP BY messageYear,
          isComment,
          lengthCategory,
 ORDER BY messageYear DESC,
          isComment ASC,
          lengthCategory ASC
```

5: final query

Variables	Meaning
messageYear	creation year of the message
isComment	id of the parent message
lengthCategory	length of the mesage categorized
messageCount	the count of how many messages based on the critera in the where clause of 5
averageMessageLength	average length of the message

Table 1: Returned variables and their meaning

```
Select count(message) AS totalMessageCount
FROM LDBC_SNB
    MATCH (message:Message)
WHERE message.creationDate < $datetime
```

6: message count

In 10the match message node is selected again, and is somewhat similar to 9. The major difference is in the WHERE clause.There the message must be 100 days newer than the provided creation date, but not an additionally 200 days newer.

In 11, which is the final query the view created in 9 and 10 are first selected. The MATCH CLAUSE selects the tag node and from there on a right path traversal is done towards the tagclass node. The traversal is done along the path named `HAS_TYPE`. At last, the variables are returned and ordered by the difference of the variable named diff (descending order)and the tag name in ascending order.

The returned variables are

9.1.3 Business Intelligence 3

The creation of a view is omitted in 15.In the MATCH clause there first is a left traversal from the forum node towards the person node. This traversal is done along the path `HAS_MODERATOR`, and a second

```
Select ParentMessageId IS NOT NULL AS isComment,
       , CASE
            WHEN length <  40 THEN 0 -- short
            WHEN length <  80 THEN 1 -- one liner
            WHEN length < 160 THEN 2 -- tweet
            ELSE                   3 -- long
          END AS lengthCategory
        , length
From LDBC_SNB
     MATCH(message:message)
WHERE  creationDate < :datetime
AND content IS NOT NULL
```

7: message prep

```
SELECT messageYear, isComment, lengthCategory,count(message) AS messageCount,
sum(message.length)/toFloat (count(message)) AS averageMessageLength,
sum(length) AS sumMessageLength,
(messageCount / toFloat(totalMessageCountInt)) AS percentageOfMessages

FROM  messagePrep_VIEW, messageCount_VIEW
      MATCH(message:message)
GROUP BY messageYear,
          isComment,
          lengthCategory,
 ORDER BY messageYear DESC,
          isComment ASC,
          lengthCategory ASC
```

8: final query

```
CREATE QUERY TAG1_VIEW as (
FROM LDBC_SNB
OPTIONAL MATCH (message1:Message)-[:HAS_TAG]->(tag)
  WHERE $date <= message1.creationDate
    AND message1.creationDate < $date + duration({days: 100})
Return count(message1) AS countWindow1
)
```

9: tag view

traversal starts after that, along the path `IS_LOCATED_IN`. The second traversal is from the person node towards the city node, and another left traversal is executed from the city node towards the country node. After those left traversals, there are also two right traversals and an undirected one. The first right traversal is from the forum node towards the post node and traverses through the path of `CONTAINER_OF`.Along the path of `HAS_TAG`, the second right traversal is done from the message node towards the tag node and the third towards the TagClass. `HAS_TYPE` is the path along which the third traversal is executed. Do note that

```
CREATE QUERY TAG2_VIEW as (
FROM LDBC_SNB
OPTIONAL MATCH (message2:Message)-[:HAS_TAG]->(tag)
WHERE $date + duration({days: 100}) <= message2.creationDate
    AND message2.creationDate < $date + duration({days: 200})
Return
       count(DISTINCT CASE WHEN Message.creationDate >= :date + INTERVAL

       '100 days' THEN Message.id ELSE NULL END)
       AS countMonth2
)
```

10: tag2 view

```
FROM TAG1_VIEW,TAG2_VIEW,LDBC
MATCH (tag:Tag)-[:HAS_TYPE]->(:TagClass {name: $tagClass})
RETURN tag.name
       countWindow1,
       countWindow2,
       abs(countWindow1 - countWindow2) AS diff
ORDER BY diff desc,
         tag.name ASC
LIMIT 100
```

11: final query

Variables name	Meaning
tag.name	name of the tag
countWindow1	count of the messages with a count lower than the creation date -100
countWindow2,	count of the messages with a count higher (100 days)than the creation dat, but lower than 200 days
diff	the absolute difference of countWindow1 and countWindow2

Table 2: Returned variables and their meaning

```
SELECT count(message1) AS countWindow1
FROM LDBC_SNB
     OPTIONAL MATCH (message1:Message)-[:HAS_TAG]->(tag)
WHERE $date <= message1.creationDate
AND message1.creationDate < $date + duration({days: 100})
```

12: tag view 1

there is another left traversal present from the message node to the post node that happens along the path `REPLY_OF`. The path traversal on the path is done at least zero or more times

```
SELECT count(DISTINCT CASE WHEN Message.creationDate >= :date + INTERVAL '100 days' THEN Message.id ELSE NULL END) AS countMont
FROM LDBC_SNB
     OPTIONAL MATCH (message2:Message)-[:HAS_TAG]->(tag)
WHERE $date + duration({days: 100}) <= message2.creationDate
AND message2.creationDate < $date + duration({days: 200})
```

13: tag view 2

```
SELECT  tag.name,countWindow1,countWindow2,abs(countWindow1 - countWindow2) AS diff
FROM TAG1_VIEW,TAG2_VIEW,LDBC
    MATCH (tag:Tag)-[:HAS_TYPE]->(:TagClass {name: $tagClass})
ORDER BY diff desc,
         tag.name ASC
LIMIT 100
```

14: final query

```
FROM LDBC_SNB
MATCH
  (:Country {country: $country})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-
  (person:Person)<-[:HAS_MODERATOR]-(f:FORUM)-[:CONTAINER_OF]->
  (post:Post)<-[:REPLY_OF]{0,...}-(message:Message)-[:HAS_TAG]->(:Tag)-[:HAS_TYPE]->(:TagClass {name: $tagClass})
RETURN
f.id            AS "f.id"
f.title         AS "f.title"
f.creationDate   AS "f.creationDate"
f.ModeratorPersonId AS "person.id"
 count(DISTINCT MessageThread.MessageId) AS messageCount
GROUP BY f.id
        ,f.title
        ,f.creationDate
        ,f.ModeratorPersonId
 ORDER BY messageCount DESC,
          f.id
 LIMIT 20
```

15: final query

The variables returned by the final query

9.1.4 Business Intelligence 4

In 20 the query first selects the city node and from there on a left path traversal is done towards the country node along the path of `IS_PART_OF`. A second left path traversal is carried out towards the city node, which comes from the forum node through the path named `HAS_MEMBER`.

The `ForumMembershipPerCountry_VIEW` query has two MATCH clauses. In the first match clause, there is a left path traversal from the forum node towards the person node. The path traversal is done along

Variables name	Meaning
f.id	id of the forum
,f.title	title of the forum
,f.creationDate	the date of the when the forum is created
,f.ModeratorPersonId	the id of the person who is the moderator

Table 3: Returned variables and their meaning

```
SELECT Forum.id AS "forum.id", Forum.title  AS "forum.title",
     Forum.creationDate   AS "forum.creationDate",
      Forum.ModeratorPersonId AS "person.id",
      count(DISTINCT MessageThread.MessageId) AS messageCount
FROM LDBC_SNB
      MATCH (:Country {country: $country})<-[:IS_PART_OF]-(:City)
             <-[:IS_LOCATED_IN]-(person:Person)<-[:HAS_MODERATOR]-(f:Forum)
             -[:CONTAINER_OF]->(post:Post)<-[:REPLY_OF]{0,...}
             -(message:Message)-[:HAS_TAG]->(:Tag)

             -[:HAS_TYPE] ->(:TagClass {name:$tagClass})
GROUP BY f.id
        ,f.title
        ,f.creationDate
        ,f.ModeratorPersonId
 ORDER BY messageCount DESC,
          Forum.id
 LIMIT 20
```

16: final query

```
Create Query as ForumMembershipPerCountry_VIEW (
FROM LDBC_SNB
MATCH (country:Country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person:Person)<-[:HAS_MEMBER]-(forum:Forum)
Return Forum.id AS ForumId
     , count(Person.id) AS numberOfMembers
     , Country.id AS CountryId
     ,  DISTINCT forum AS topForum1
 GROUP BY Country.Id, Forum.Id
)
```

17: Membership per forum per Country

the path named HAS_MEMBER. The OPTIONAL MATCH has two right path traversals and one left path traversal. The first right path traversal is from topforum 1 node or the results retrieved from 20 and goes towards the post node, through the path named CONTAINER_OF. The second traversal is from the message node towards the person node and travels along the path of HAS_CREATOR.

The last query is the final one and selects all the results from 20 and 4-2, and returns the variables, which are described in table 4

```
Create Query as  Top100_Popular_Forums_VIEW(
FROM ForumMembershipPerCountry_VIEW,LDBC_SNB
MATCH (person:Person)<-[:HAS_MEMBER]-(topForum2:Forum)
OPTIONAL MATCH (topForum1)
-[:CONTAINER_OF]->(post:Post)
<-[:REPLY_OF]{0,...}-(message:Message)-[:HAS_CREATOR]->(person)
WHERE topForum2 IN topForums
RETURN DISTINCT ForumId AS id
        ,max(numberOfMembers) AS maxNumberOfMembers
        person,
        message,
        topForum2
  GROUP BY ForumId

  ORDER BY maxNumberOfMembers DESC
          ,ForumId
  LIMIT 100
)
```

18: Top 100Popular Forums

```
FROM Top100_Popular_Forums_VIEW,ForumMembershipPerCountry_VIEW
MATCH ()
 RETURN person.id AS personId,
  person.firstName AS personFirstName,
  person.lastName AS personLastName,
  person.creationDate AS personCreationDate,
  count(DISTINCT message) AS messageCount
ORDER BY
  messageCount DESC,
  person.id ASC
LIMIT 100
```

19: Final Query

personId	the id of the person
personFirstName	the first name
personLastName	last name
personCreationDate	the date at which the account was created
messageCount	the amount of messages sent by a user in a forum

Table 4: Returned variables and their meaning

9.1.5 Business Intelligence 5

In 24 the query has two optional MATCH clauses and one MATCH clause. In the MATCH clause, there is a left, and right. the left path traversal starts from the message node towards the tag node, and traverses through the path with label named `HAS_TAG`, while the right path traversal happens through the `HAS_CREATOR` label. The label comes from the message node and goes toward the person node At last, the query returns the variables described in table 5

```
//ForumMembershipPerCountry_VIEW

SELECT Forum.id AS ForumId,
        count(Person.id) AS numberOfMember,Country.id AS CountryId,
        DISTINCT forum AS topForum1
FROM LDBC_SNB
     MATCH (country:Country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]
     -(person:Person)<-[:HAS_MEMBER]-(forum:Forum)
 GROUP BY Country.Id
          ,Forum.Id
```

20: ForumMembership Per Country

```
//Top100_Popular_Forums_VIEW

SELECT DISTINCT ForumId AS id,max(numberOfMembers) AS maxNumberOfMembers
                ,person, message, topForum2
FROM ForumMembershipPerCountry_VIEW,LDBC_SNB
     MATCH (person:Person)<-[:HAS_MEMBER]-(topForum2:Forum)
     OPTIONAL MATCH (topForum1)-[:CONTAINER_OF]->(post:Post)
                    <-[:REPLY_OF]{0,...}
                    -(message:Message)-[:HAS_CREATOR]->(person)
WHERE topForum2 IN topForums
  GROUP BY ForumId

  ORDER BY maxNumberOfMembers DESC
          ,ForumId
  LIMIT 100
```

21: Top 100 Popular Forums VIEW

```
Select person.id AS personId,person.firstName AS personFirstName
     ,person.lastName AS personLastName, person.creationDate AS personCreationDate
     ,count(DISTINCT message) AS messageCount
FROM Top100_Popular_Forums_VIEW,ForumMembershipPerCountry_VIEW
     MATCH ()
ORDER BY
  messageCount DESC,
  person.id ASC
LIMIT 100
```

22: Top 100 Popular Forums VIEW

9.1.6 Business Intelligence 6

In 29 the left traversal goes through the label `HAS_TAG`and from the message node towards the tag node. Also, from the message node, there is another right path traversal towards the person node, which

```
FROM LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(p:Person)
OPTIONAL MATCH (message)<-[likes:LIKES]-(:Person)
OPTIONAL MATCH (message)<-[:REPLY_OF]-(reply:Comment)
REUTRN CreatorPerson.id AS person.id
     , count(DISTINCT Comment.Id)  AS replyCount
     , count(DISTINCT Person_likes_Message.MessageId||' '||Person_likes_Message.PersonId) AS likeCount
     , count(DISTINCT Message.Id)  AS messageCount
     ,1*messageCount + 2*replyCount + 10*likeCount AS score

 GROUP BY CreatorPerson.id
 ORDER BY score DESC, CreatorPersonId
 LIMIT 100
```

23: final query

person.id	the id of the person
replyCount	the amount of times there is replied on a message
likeCount	count of likes
messageCount	the amount of distinct messages
score	the score of the message dependant on the count of likes, the amount of times a reply is done and the me

Table 5: Returned variables and their meaning

```
SELECT CreatorPerson.id AS person.id
     , count(DISTINCT Comment.Id)  AS replyCount
     , count(DISTINCT Person_likes_Message.MessageId||' '||Person_likes_Message.PersonId) AS likeCount
     , count(DISTINCT Message.Id)  AS messageCount
     ,1*messageCount + 2*replyCount + 10*likeCount AS score
FROM LDBC_SNB
      MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(p:Person)
      OPTIONAL MATCH (message)<-[likes:LIKES]-(:Person)
      OPTIONAL MATCH (message)<-[:REPLY_OF]-(reply:Comment)
 GROUP BY CreatorPerson.id
 ORDER BY score DESC, CreatorPersonId
 LIMIT 100
```

24: final query

happens along the path `HAS_CREATOR`. Both traversals happen within the MATCH CLAUSE. The other match clause, which is an OPTIONAL MATCH has a left path traversal from the person node towards the message node. In there

The `poster_w_liker_VIEW` has a left traversal through the path `HAS_TAG`, which comes from the message node towards the tag node. From the message node towards the person node, there is a right path traversal that travels through the `HAS_CREATOR` node.

The `popularity_score` view has two left path traversals. The first starts from the person node and travels through the LIKES label towards the message node (from 29). From that message node, the second traversal is done from the message node towards the person node of 32. That traversal is done through the `HAS_CREATOR` label.

```
CREATE QUERY detail_VIEW as (
FROM LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(person:Person)
OPTIONAL MATCH [:REPLY_OF]-(reply:Comment)->(message)<-[likes:LIKES]-(:Person)
REUTRN CreatorPerson.id AS CreatorPersonId
     , count(DISTINCT Comment.Id)  AS replyCount
     , count(DISTINCT Person_likes_Message.MessageId||' '||Person_likes_Message.PersonId) AS likeCount
     , count(DISTINCT Message.Id)  AS messageCount
     , NULL as score
GROUP BY CreatorPerson.id
)
```

25: detail view

```
Create Query poster_w_liker_VIEW AS(
FROM detail_VIEW,LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message1:Message)-[:HAS_CREATOR]->(person1:Person)
OPTIONAL MATCH (message1)<-[:LIKES]-(person2:Person)
RETURN    DISTINCT   m1.CreatorPersonId AS posterPersonid
         ,l2.PersonId AS likerPersonid
)
```

26: poster liker view

```
CREATE QUERY popularity_score_VIEW AS (
FROM LDBC_SNB,poster_w_liker_VIEW
OPTIONAL MATCH (person2)<-[:HAS_CREATOR]-(message2:Message)<-[like:LIKES]-(person3:Person)
RETURN  CreatorPersonId AS PersonId
        ,count(*) AS popularityScore
GROUP BY m3.CreatorPersonId
)
```

27: Popularity score

The final query also has a MATCH CLAUSE. This clause does a left path traversal from the person node towards the likerPersonid, and travels along the path named `POPULARITY_SCORE`. The variables returned from the final query are described in table 9.1.7

Variables	**Meaning**
person1.id	the id of the person
authorityScore	sum of the popularity score retrieved from the traversal in the last queryn

Table 6: Returned variables and their meaning

9.1.7 Business Intelligence 7

```
/*Final Query*/
FROM poster_w_liker_VIEW,popularity_score_VIEW,Query poster_w_liker_VIEW ,detail_VIEW
MATCH (pl: likerPersonid) <- [:POPULARITY_SCORE]-(ps:Person)
RETURN  pl.posterPersonid AS "person1.id"
     , sum(coalesce(ps.popularityScore, 0)) AS authorityScore
 GROUP BY pl.posterPersonid
 ORDER BY authorityScore DESC, pl.posterPersonid ASC
 LIMIT 100
;
```

28: final query

```
// detail_VIEW
SELECT CreatorPerson.id AS CreatorPersonId
     , count(DISTINCT Comment.Id)  AS replyCount
     , count(DISTINCT Person_likes_Message.MessageId||' '||Person_likes_Message.PersonId) AS likeCount
     , count(DISTINCT Message.Id)  AS messageCount
     , NULL as score
FROM LDBC_SNB
    MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(person:Person)
    OPTIONAL MATCH (message)<-[likes:LIKES]-(:Person)
    OPTIONAL MATCH (message)<-[:REPLY_OF]-(reply:Comment)
GROUP BY CreatorPerson.id
```

29: detail view

```
//poster_w_liker_VIEW
SELECT DISTINCT  m1.CreatorPersonId AS posterPersonid
         ,l2.PersonId AS likerPersonid
FROM detail_VIEW,LDBC_SNB
     MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(message1:Message)-[:HAS_CREATOR]->(person1:Person)
     OPTIONAL MATCH (message1)<-[:LIKES]-(person2:Person)
```

30: poster liker view

```
//popularity_score_VIEW
SELECT  CreatorPersonId AS PersonId, count(*) AS popularityScore
FROM Message m3
     OPTIONAL MATCH (person2)<-[:HAS_CREATOR]-(message2:Message)<-[like:LIKES]-(person3:Person)
GROUP BY m3.CreatorPersonId
```

31: Popularity score

```
//final query
SELECT  pl.posterPersonid AS "person1.id"
       ,sum(COALESCE(ps.popularityScore, 0)) AS authorityScore
FROM poster_w_liker_VIEW
MATCH (pl: likerPersonid) <- [:POPULARITY_SCORE] (ps:Person)

 GROUP BY pl.posterPersonid
 ORDER BY authorityScore DESC
         ,pl.posterPersonid ASC
LIMIT 100
```

32: final query

```
/*Translation Bi7 query*/
FROM LDBC_SNB
MATCH
  (tag:Tag {name: $tag})<-[:HAS_TAG]-(m:Message),
  (m)<-[:REPLY_OF]-(c:Comment)-[:HAS_TAG]->(rTag:Tag)
    WHERE NOT (c)-[:HAS_TAG]->(tag)

RETURN RelatedTag.name AS "relatedTag.name"
       , count(*) AS count
 GROUP BY RelatedTag.name
 ORDER BY count DESC
          ,RelatedTag.name
 LIMIT 100
```

33: final query

The query above traverses from the message node towards the tag node. This traversal happens through the `HAS_TAG` label. A second traversal is present in the MATCH clause, which consists of a left path traversal and a right path traversal, unlike the previous traversal that is only a left path traversal. The first right path traversal is from the comment node and travels through the `HAS_TAG` path towards the Tag node, while the left path traversal goes from the comment node towards the message node.`REPLY_OF` is the path through which the left path traversal happens.

The 37 is composed with a WHERE EXIST CLAUSE. In there the query starts with a left path traversal from the message node towards the tag node, traversing through the path named `HAS_TAG`. The other path `HAS_CREATOR` has a right traversal from the message node towards the person node. The inner clause is a condition, therefore the outer MATCH clause first gets executed. In that clause, the person node is selected and from there on a traversal is done towards the tag node through the `HAS_INTEREST` label.

9.1.8 Business Intelligence 8

The last query retrieves the results from the `person_VIEW`. This however, might be redundant ,and

Variables	Meaning
relatedTag.name	the name of the tag
count	count of the messages related to the tag ID

```
SELECT RelatedTag.name AS "relatedTag.name",count(*) AS count
FROM LDBC_SNB
     MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(m:Message),
           (m)<-[:REPLY_OF]-(c:Comment)-[:HAS_TAG]->(rTag:Tag)

            WHERE NOT (c)-[:HAS_TAG]->(tag)

GROUP BY RelatedTag.name
ORDER BY count DESC
          ,RelatedTag.name
LIMIT 100
```

34: final query

```
/*Translation Bi8 query*/

CREATE Query Person_VIEW as (
FROM  Person
MATCH (tag:Tag {name: $tag})
// score
MATCH (tag)<-[interest:HAS_INTEREST]-(person:Person)
tag, collect(person) AS interestedPersons
WHERE EXIST{
    MATCH (tag)<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(person:Person)
}
RETURN DISTINCT tag
       ,DISTINCT(interestedPersons + COLLECT (person) )AS persons
)
```

35: Person view

could be constructed in one query. But the current version of GQL described in the available resources does not mention composable graph queries. As a result, a second query is constructed as the final query. In it we have a traversal from the friend node towards the person node. This traversal is a LUR and happens through the KNOWS path. As a final step the variables described in table 7 are returned.

Variables
tag,
100 * size([(tag)<-[interest:HAS_INTEREST]-(person) — interest]) + size([(tag)<-[:HAS_TAG]-(message:Message)-[:HAS_

Table 7: Returned variables and their meaning

9.1.9 Business Intelligence 9

The query starts of by having two left path traversals. The first is from the message node, and traverses to the post node. During traversal between 0 and 1 path are traversed.The traversal is executed along the path named `REPLY_OF`.Second traversal ends at the person node,which starts from the post node,

```
FROM Person_VIEW
 MATCH (person)-[:KNOWS]-(friend)
// We need to use a redundant computation due to the lack of composable graph queries in the currently supported Cypher version.
// This might change in the future with new Cypher versions and GQL.
RETURN
 tag,
  person,
  100 * size([(tag)<-[interest:HAS_INTEREST]-(person) | interest]) + size([(tag)<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->
  AS score
  ,sum(score) AS friendScore

ORDER BY
  score + friendsScore DESC,
  person.id ASC
LIMIT 100
```

36: Final query

```
SELECT DISTINCT tag
       ,DISTINCT(interestedPersons + COLLECT (person) )AS persons
FROM  LDBC_SNB
      MATCH (tag:Tag {name: $tag}),(tag)<-[interest:HAS_INTEREST]-(person:Person)
            tag, collect(person) AS interestedPersons
WHERE EXIST{
    MATCH (tag)<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(person:Person)
}
```

37: Person view

```
//final query
SELECT tag, person,
  100 * size([(tag)<-[interest:HAS_INTEREST]-(person) | interest]) + size([(tag)<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->
  AS score
  ,sum(score) AS friendScore
FROM LDBC_SNB,Person_VIEW
     MATCH (person)-[:KNOWS]-(friend)
ORDER BY
  score + friendsScore DESC,
  person.id ASC
LIMIT 100
```

38: Final query

and travels through the path `HAS_CREATOR`. At last variables are returned which are described in table 8

9.1.10 Business Intelligence 11

```
FROM LDBC_SNB
MATCH (person:Person)<-[:HAS_CREATOR]-(post:Post)<-[:REPLY_OF]{0,1}-(reply:Message)
WHERE  post.creationDate >= $startDate
  AND  post.creationDate <= $endDate
  AND reply.creationDate >= $startDate
  AND reply.creationDate <= $endDate
RETURN person.id
      ,person.firstName
      ,person.lastName
      ,count(DISTINCT post) AS threadCount
      ,count(DISTINCT reply) AS messageCount
ORDER BY
  messageCount DESC,
  person.id ASC
LIMIT 100
 LIMIT 100
```

39: final query

Variables	Meaning
person.id	id of the person
person.firstName	first name of the person
person.lastName	last name of the person
threadCount	amount of posts by that individual
messageCount	amount of replies by that same individual

Table 8: Returned variables and their meaning

```
 RelatedTag.name AS "relatedTag.name",count(*) AS count
FROM LDBC_SNB
     MATCH (tag:Tag {name: $tag})<-[:HAS_TAG]-(m:Message),
           (m)<-[:REPLY_OF]-(c:Comment)-[:HAS_TAG]->(rTag:Tag)

            WHERE NOT (c)-[:HAS_TAG]->(tag)

GROUP BY RelatedTag.name
ORDER BY count DESC
          ,RelatedTag.name
LIMIT 100
```

40: final query

The query has in the match clause six right path traversals. The first traversal happens from the person node towards the city node and travels through the `IS_LOCATED_IN` path. From the city node, the second traversal happens toward the country node through the label `IS_PART_OF`. The other right path traversals follow the same routine. The only difference is that there are three different person node traversals towards the country node. In the end a LUR is executed to see whether the three persons know each other, and the query returns the variables described in the table 9

```
FROM LDBC_SNB
MATCH (country:Country {name: $country}),
 (person1:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
 (person2:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
 (person3:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
 (person1)-[k1:KNOWS]-(person2)-[k2:KNOWS]-(person3)-[k3:KNOWS]-(a)
WHERE a.id < b.id
  AND b.id < c.id
  AND $startDate <= k1.creationDate <= k1.creationDate
  AND $startDate <= k2.creationDate
  AND $startDate <= k3.creationDate
RETURN count(*) AS count
       ,person1
       ,person2
       ,person3
```

41: final query

Variables	Meaning
person1	results retrieved for the person 1 node
person2	results retrieved for the person 2 node
person3	results retrieved for ther person 3 node

Table 9: Returned variables and their meaning

```
SELECT count(*) AS count
       ,person1
       ,person2
       ,person3
FROM LDBC_SNB
     MATCH (country:Country {name: $country}),
           (person1:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
           (person2:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
           (person3:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country),
           (person1)-[k1:KNOWS]-(person2)-[k2:KNOWS]-(person3)-[k3:KNOWS]-(a)
WHERE a.id < b.id
  AND b.id < c.id
  AND $startDate <= k1.creationDate <= k1.creationDate
  AND $startDate <= k2.creationDate
  AND $startDate <= k3.creationDate
```

42: final query

9.1.11 Business Intelligence 12

From the person node, there is an incoming left traversal from the message node, which travels through the `HAS_CREATOR` path. Another traversal is present, which travels toward the post node, of which at least 0 paths are traversed through the `REPLY_OF` path.The WHERE clause described the conditions for the retrieved results.The query returns the variable messageCount, which indicates the amount of messages

```
FROM  LDBC_SNB
OPTIONAL MATCH (person: person)<-[:HAS_CREATOR]-(message:Message)-[:REPLY_OF]{0,..}->(post:Post)
WHERE message.content IS NOT NULL
  AND message.length < 20
  AND message.creationDate > \$startDate
  AND post.language IN   ['ar', 'hu']
RETURN
  messageCount,
  count(person) AS personCount
ORDER BY
  personCount DESC,
  messageCount DESC
```

43: Final Query

created within the specified conditions

```
SELECT messageCount,
       count(person) AS personCount
 FROM  LDBC_SNB
       OPTIONAL MATCH (person: person)<-[:HAS_CREATOR]-(message:Message)
                      -[:REPLY_OF]{0,..}->(post:Post)
  WHERE message.content IS NOT NULL
    AND message.length < 20
    AND message.creationDate > $startDate
    AND post.language IN   ['ar', 'hu']
  ORDER BY
    personCount DESC,
    messageCount DESC
```

44: final query

9.1.12 Business Intelligence 13

```
CREATE QUERY Located_VIEW as (
FROM LDBC_SNB
MATCH (country:Country {name: $country})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(:Person)
WHERE person.creationDate < $endDate
RETURN country
       ,person
)
```

45: Location Retrieval

From the query above we have the person node and from there onwards there are only left path traversals. The first one ends at the city node, which travels through the path named `IS_PART_OF`.After traversal of that path a second path is followed named `IS_LOCATED_IN` and ends at the country node. Furthermore, the query has a WHERE condition. In the condition, a check is executed to see whether the date of the created profile is lower than the provided end date.

```
CREATE QUERY HAS_CREATOR_VIEW as (
OPTIONAL MATCH (person)<-[:HAS_CREATOR]-(message:Message)
WHERE message.creationDate < $endDate
    AND  messageCount / months < 1
RETURN  country
       ,person
       ,count(message) AS messageCount

)
```

46: Creator of the message

In 46 the OPTIONAL MATCH clause has a left path traversal from the message node towards the Person node through the path `HAS_CREATOR`. Also in this subquery, the WHERE condition verifies whether the created profile date is lower than the end date.

The query above has an OPTIONAL MATCH, in which a left path traversal is done from the person node towards the message node through the `HAS_CREATOR` path towards another person node. Do note that this WHERE clause has another type of condition. In there a check is done to see whether those two results retrieved from the node are identical. At last, another optional MATCH is present which retrieves the results from 45 and 46. The query returns the variables described in table 9.1.12

Variables	Meaning
person.id	the id of person
personLikeCount	amount a person liked a comment or message
totalLikeCount	the total amount of messages/comments liked by a person

Table 10: Variables and their meaining

9.1.13 Business Intelligence 14

```
CREATE QUERY LikerPerson_VIEW as (
OPTIONAL MATCH
  (person)<-[:HAS_CREATOR]-(message:Message)<-[:LIKES]-(likerperson:Person)
WHERE likerperson IN persons
RETURN country,
       COLLECT (person) AS persons
         12 * ($endDate.year  - person.creationDate.year )
     + ($endDate.month - person.creationDate.month)
     + 1 AS months
     ,messageCount
)
```

47: Person likes Person message

```
FROM LDBC_SNB,LikerPerson_VIEW,HAS_CREATOR_VIEW,Located_VIEW
OPTIONAL MATCH
  (person)<-[:HAS_CREATOR]-(message:Message)<-[:LIKES]-(likerPerson:Person)
WHERE likerPerson.creationDate < $endDate

RETURN person.id
       ,count(likerperson) AS personLikeCount
       ,totalLikeCount
  CASE totalLikeCount
    WHEN 0 THEN 0.0
    ELSE personLikeCount / toFloat(totalLikeCount)
  END AS personScore
ORDER BY
  personScore DESC,
  person.id ASC
LIMIT 100
```

48: final query

```
SELECT country
       ,zombie
       ,count(message) AS messageCount
FROM LDBC_SNB
MATCH (country:Country {name: $country})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(zombie:Person)
WHERE zombie.creationDate < $endDate
RETURN country, zombie
OPTIONAL MATCH (zombie)<-[:HAS_CREATOR]-(message:Message)
WHERE message.creationDate < $endDate
    AND  messageCount / months < 1
```

49: final query

```
SELECT country,
       collect(zombie) AS zombies
         12 * ($endDate.year  - zombie.creationDate.year )
     + ($endDate.month - zombie.creationDate.month)
     + 1 AS months
     ,messageCount
FROM  ,LDBC_SNB
OPTIONAL MATCH
  (zombie)<-[:HAS_CREATOR]-(message:Message)<-[:LIKES]-(likerZombie:Person)
WHERE likerZombie IN zombies
```

50: final query

```
SELECT zombie.id
       ,count(likerZombie) AS zombieLikeCount
       ,zombieLikeCount,
       ,totalLikeCount,
    CASE totalLikeCount
    WHEN 0 THEN 0.0
    ELSE zombieLikeCount / toFloat(totalLikeCount)
    END AS zombieScore
FROM ,LDBC
OPTIONAL MATCH (zombie)<-[:HAS_CREATOR]-(message:Message)<-[:LIKES]-(likerPerson:Person)
WHERE likerPerson.creationDate < $endDate
ORDER BY
  zombieScore DESC,
  zombie.id ASC
LIMIT 100
```

51: final query

```
/*Translate Bi14 query*/
CREATE QUERY KNOW_VIEW AS (
FROM LBC_SNB
MATCH
  (country1:Country {name: $country1})<-[:IS_PART_OF]-(city1:City)<-[:IS_LOCATED_IN]-(person1:Person),
  (country2:Country {name: $country2})<-[:IS_PART_OF]-(city2:City)<-[:IS_LOCATED_IN]-(person2:Person),
  (person1)-[:KNOWS]-(person2)
Return person1
       ,person2
       ,city1
       ,0 AS score
)
```

52: Know view

In the **??** the MATCH clause consists of three independent clauses. The first clause executes a traversal from the person node towards the city node through the path of the `IS_LOCATED_IN`. From the city node, there is another left traversal, which goes to the country node through the path `IS_PART_OF`. The last clause verifies whether the two people know each other.

```
//case 1
FROM LDBC_SNB
OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(c:Comment)-[:REPLY_OF]->(:Message)-[:HAS_CREATOR]->(person2)
Return DISTINCT person1
                ,person2
                ,city1
                ,score + (CASE c WHEN null THEN 0 ELSE  4 END) AS score
```

53: Case 1

In 53 the MATCH clause has a left path traversal as well as a right one. The right one starts from the message node and moves towards the person node, through the label 53. Towards the message node, there is another incoming right path traversal from the comment node, which comes through the `REPLY_OF` path. From the coming node, there is a left path traversal from the comment node towards the person node and travels through the path `HAS_CREATOR`.

```
// case 2// Ommit FROM, it requires person from the first tag
FROM LDBC_SNB
OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(m:Message)<-[:REPLY_OF]-(:Comment)-[:HAS_CREATOR]->(person2)
RETURN DISTINCT person1
                ,person2
                ,city1
                ,score + (CASE m WHEN null THEN 0 ELSE  1 END) AS score
```

54: Case 2

The contents of the OPTIONAL MATCH are similar to that of CASE 1 in 54. However, the return statements differ. The difference lies in the score variable.

```
// case 3
FROM LDBC_SNB
OPTIONAL MATCH (person1)-[:LIKES]->(m:Message)-[:HAS_CREATOR]->(person2)
RETURN  DISTINCT person1
                ,person2
                ,city1
                ,score + (CASE m WHEN null THEN 0 ELSE 10 END) AS score
```

55: Case 3

In 55 and 56 the traversal is almost similar. The main difference is that in 56 there is a right path traversal from the person1 node towards the message node through the path named `LIKES`, and the second right path traversal is from the message node through the path `HAS_CREATOR` towards the person2 node,

```
// case 4
FROM LDBC_SNB
OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(m:Message)<-[:LIKES]-(person2)
RETURN DISTINCT person1.id
                ,person2.id
                ,city1.name
                ,score + (CASE m WHEN null THEN 0 ELSE  1 END) AS score DESC
ORDER BY
  top.score DESC,
  top.person1.id ASC,
  top.person2.id ASC
```

56: Final Query

Variables	Meaning
person1.id	id of person retrieved from node person1
person2.id	id of person retrieved from node person 2
city1	city the person lives in

Table 11: Returned variables and their meaning

whereas in 56 the traversal is a left path, and the labels are exchanged. The query returns the following variables described in table 11

```
SELECT  person1
       ,person2
       ,city1
       ,0 AS score

FROM LBC_SNB
     MATCH (country1:Country {name: $country1})<-[:IS_PART_OF]-(city1:City)<-[:IS_LOCATED_IN]-(person1:Person),
           (country2:Country {name: $country2})<-[:IS_PART_OF]-(city2:City)<-[:IS_LOCATED_IN]-(person2:Person),
           (person1)-[:KNOWS]-(person2)
```

57: final query

```
SELECT DISTINCT person1
               ,person2
               ,city1
               ,score + (CASE c WHEN null THEN 0 ELSE  4 END) AS score
FROM LBC_SNB
     OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(c:Comment)-[:REPLY_OF]->(:Message)-[:HAS_CREATOR]->(person2)
```

58: Mutual Friend

```
SELECT DISTINCT person1,person2,city1,score + (CASE m WHEN null THEN 0 ELSE  1 END) AS score
FROM LBC_SNB
      OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(m:Message)<-[:REPLY_OF]-(:Comment)-[:HAS_CREATOR]->(person2)
```

59: Creator of the message

```
SELECT  DISTINCT person1 ,person2 ,city1,score + (CASE m WHEN null THEN 0 ELSE 10 END) AS score
FROM LBC_SNB
     OPTIONAL MATCH (person1)-[:LIKES]->(m:Message)-[:HAS_CREATOR]->(person2)
```

60: extended message creation results

```
SELECT DISTINCT person1.id,person2.id,city1.name,score + (CASE m WHEN null THEN 0 ELSE  1 END) AS score DESC
FROM LBC_SNB
     OPTIONAL MATCH (person1)<-[:HAS_CREATOR]-(m:Message)<-[:LIKES]-(person2)
ORDER BY
  top.score DESC,
  top.person1.id ASC,
  top.person2.id ASC
```

61: final query

9.1.14 Business Intelligence 18

```
FROM  LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_INTEREST]-(person1:Person)-[:KNOWS]-(commonFriend:Person)-[:HAS_INTEREST]->(tag)
/*Reason for an optional match is the left join in sql*/
OPTIONAL MATCH (commonFriend:Person)-[:KNOWS]-(person2:Person)
WHERE person1 <> person2
  AND NOT (person1)-[:KNOWS]-(person2)
RETURN person1.id AS person1Id
      ,person2.id AS person2Id
      ,count(DISTINCT mutualFriend) AS mutualFriendCount
ORDER BY mutualFriendCount DESC, person1Id ASC, person2Id ASC
LIMIT 20
```

62: final query

In 62 the query has two MATCH clauses. In the first MATCH clause, the query first traverses from the person node towards the tag node, in a left manner through the path named `HAS_INTEREST`. From the person node, there is a LUR towards itself that travels through the label `KNOWS`. From the person node then there is a right traversal from the person node towards the tag node. That traversal is executed through the path named `HAS_INTEREST`. The second OPTIONAL MATCH does another traversal from the person node

towards the person node itself through the label `KNOWS`. This is done to see whether person1 is a mutual friend of person 2. At last, the following variables are returned as described in table 12

Variables	Meaning
idPerson1	the id of person 1
idPerson2	id of person 2
mutualFriendCount	the amount of mutual friends

Table 12: Returned variables and their meaning

```
SELECT person1.id AS person1Id,person2.id AS person2Id,count(DISTINCT mutualFriend) AS mutualFriendCount
FROM  LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_INTEREST]-(person1:Person)-[:KNOWS]-(mutualFriend:Person)-[:HAS_INTEREST]->(tag)
OPTIONAL MATCH (mutualFriend:Person)
-[:KNOWS]-(person2:Person)
WHERE person1 <> person2
  AND NOT (person1)-[:KNOWS]-(person2)
ORDER BY mutualFriendCount DESC, person1Id ASC, person2Id ASC
LIMIT 20
```

63: final query

```
SELECT person1.id AS person1Id,person2.id AS person2Id,count(DISTINCT mutualFriend) AS mutualFriendCount
FROM  LDBC_SNB
MATCH (tag:Tag {name: $tag})<-[:HAS_INTEREST]-(person1:Person)-[:KNOWS]-(mutualFriend:Person)-[:HAS_INTEREST]->(tag)
OPTIONAL MATCH (mutualFriend:Person)
-[:KNOWS]-(person2:Person)
WHERE person1 <> person2
  AND NOT (person1)-[:KNOWS]-(person2)
ORDER BY mutualFriendCount DESC, person1Id ASC, person2Id ASC
LIMIT 20
```

64: final query

9.2 Interactive Queries

In the following paragraphs the interactive queries are shown along with the variables meaning, return size and description of each query.

9.2.1 Interactive Complex 1

Since GQL does not yet support accumulators the query is constructed by first creating small views that return the results. Each consequent view is a follow-up of the previous.In `Query_friend_VIEW1` the query first accumulates all the person's first names and last names, the friends of each other. Then the person's ID and that friend are returned In the second view `friend_VIEW2` the query first retrieves the shorted path between person and friend, of which the path can be between 1 and 3 , indicated by 1,3. After that the minimal length is returned from that traversed path and the friend as second result.

```
CREATE QUERY friend_VIEW1  AS (
FROM LDBC_SNB
MATCH (p:Person {id: $personId}) ~[:friend] ~ (Person {firstName: $firstName})
RETURN p
      ,friend
)
```

```
CREATE QUERY  friend_VIEW2 AS (
FROM LDBC_SNB, friend_VIEW1
MATCH path = shortestPath((p)-[:KNOWS]-(f:friend)) {1,3}
WHERE p.id == friend_VIEW1.p.id
Return min(length(path)) AS distance
       , f
ORDER BY
    distance ASC,
    f.lastName ASC,
    toInteger(f.id) ASC
LIMIT 20
```

In the following query the use of `friend_VIEW2` is used, since the distance result is required from the previous views. At first the query looks in which city each friend is located. In the second match (optional match), the the city in which each friend or person studies is retrieved. The returned results are the uni name provided a value of T if the university name is missing or else the name, class year city, and the distance value.

```
CREATE QUERY friend_VIEW3 as (
MATCH (f:friend)-[:IS_LOCATED_IN]
       ->(friendCity:City)
OPTIONAL MATCH (friend)-[studyAt:STUDY_AT]
           ->(uni:University)-[:IS_LOCATED_IN]->(uniCity:City)
CASE uni.name
        WHEN null T
        THEN null
        ELSE [uni.name, studyAt.classYear, uniCity.name]
 END ) AS
WHERE f.id == friend_VIEW1.f.id
 Return  unis
        ,friendCity
        ,distance
)
```

The query below is the last query without a view. I In there the company information at where that friend work is obtained by first traversing the path between friend and works at the company and afterwards the path between company and the country at which the company is located.After path traversal he company name is returned (if not null), university information of `friend_VIEW3`, and the distance. Furthermore all remaining results of the views are returned.

```
FROM friend_VIEW1,friend_VIEW2,friend_VIEW3
OPTIONAL MATCH (f:friend)-[workAt:WORK_AT]->(company:Company)-[:IS_LOCATED_IN]
          ->(companyCountry:Country)
CASE company.name
        WHEN null THEN null
        ELSE [company.name, workAt.workFrom, companyCountry.name]
END ) AS companies
WHERE f.id = friend_VIEW1.id
Return
        p.id AS friendId,
        f.lastName AS friendLastName,
        distance   AS distanceFromPerson,
        p.birthday AS friendBirthday,
        p.creationDate AS friendCreationDate,
        p.gender AS friendGender,
        p.browserUsed AS friendBrowserUsed,
        p.locationIp AS friendLocationIp,
        p.email AS friendEmails,
        p.speaks AS friendSpeaks,
        friendCityName AS friendCityName,
        companies AS friendCompanies,
        universities as friendUniversities
```

```
SELECT p,friend
FROM LDBC_SNB
     MATCH (p:Person {id: $personId}) ~[:friend] ~ (Person {firstName: $firstName})
```

```
//friend_VIEW2
SELECT min(length(path)) AS distance,f
FROM LDBC_SNB, friend_VIEW1
     MATCH path = shortestPath((p)-[:KNOWS]-(f:friend)) {1,3}
     WHERE p.id == friend_VIEW1.p.id
ORDER BY
    distance ASC,
    f.lastName ASC,
    toInteger(f.id) ASC
LIMIT 20
```

9.2.2 Interactive Complex 2

```
SELECT  unis,friendCity,distance
MATCH (f:friend)-[:IS_LOCATED_IN]->(friendCity:City)
OPTIONAL MATCH (friend)-[studyAt:STUDY_AT]->(uni:University)-[:IS_LOCATED_IN]->(uniCity:City)
CASE uni.name
        WHEN null T
        THEN null
        ELSE [uni.name, studyAt.classYear, uniCity.name]
 END  AS
WHERE f.id == friend_VIEW1.f.id
```

```
//final query
SELECT  p.id AS friendId,f.lastName AS friendLastName, distance AS distanceFromPerson,
        p.birthday AS friendBirthday,p.creationDate AS friendCreationDate,
        p.gender AS friendGender,p.browserUsed AS friendBrowserUsed,
        p.locationIp AS friendLocationIp,p.email AS friendEmails,
        p.speaks AS friendSpeaks,friendCityName AS friendCityName,companies AS friendCompanies,
        universities as friendUniversities
OPTIONAL MATCH (f:friend)-[workAt:WORK_AT]->(company:Company)-[:IS_LOCATED_IN]->(companyCountry:Country)
        CASE company.name
        WHEN null THEN null
        ELSE [company.name, workAt.workFrom, companyCountry.name]
        END AS companies
        WHERE f.id = friend_VIEW1.id
```

68: Fianl Query

```

FROM LDBC_SNB
MATCH (P:Person {id: $personId })-[:KNOWS]-(f:Person)<-[:HAS_CREATOR]-(m:Message)
WHERE message.creationDate <= $maxDate
CASE :
    m.content <> ""
    THEN
            result = coalesce(m.content,m.imageFile)

    ELSE   NULL

    RETURN
        f.id AS personId,
        f.firstName AS personFirstName,
        f.lastName AS personLastName,
        m.id AS postID,
        result AS postOrCommentContent,
        m.creationDate AS postDate
    ORDER BY
        postOrCommentContent DESC,
        toInteger(postOrCommentId) ASC
    LIMIT 20
```

Query 2 is a simple query in which there is a path traversal from person to person between whom an interaction happened. From the node Person, a traversal is traversed to another node person along with the label of edge KNOWS. From there on a left path traversal is carried out from the node message to the friend along the edge path `HAS_CREATOR`. The query returns the id, name of the people between who an interaction is carried out, and the date of the post as well as its ID.

```
SELECT f.id AS personId,f.firstName AS personFirstName,f.lastName AS personLastName,
        m.id AS postID,result AS postOrCommentContent,m.creationDate AS postDate
FROM LDBC_SNB
MATCH (P:Person {id: $personId })-[:KNOWS]-(f:Person)<-[:HAS_CREATOR]-(m:Message)
      WHERE message.creationDate <= $maxDate
      CASE :
            m.content <> ""
            THEN
                    result = coalesce(m.content,m.imageFile)

            ELSE   NULL


            ORDER BY
                postOrCommentContent DESC,
                toInteger(postOrCommentId) ASC
            LIMIT 20
```

69: ToDo

9.2.3 Interactive Complex 3

At first a view is created to select the nodes countryXname and countryYName and the person

```
CREATE QUERY city_VIEW1 AS (
FROM LDBC_SNB
MATCH (x:Country {name: $countryXName }),
      (y:Country {name: $countryYName }),
      (p:Person {id: $personId })
RETURN person
       ,x AS countryA
       ,y AS countryB

LIMIT 1
)
```

In `city_VIEW1` the path between city and country nodes is traversed right along the path `IS_PART_OF`. From the previous view we do need the variables countryA and countryB, to see in which cities a person is or has been After the path traversal the cities a person has been to are returned with their respective country.

```
CREATE QUERY city_VIEW2 AS (
FROM country_VIEW1
MATCH (c:City)-[:IS_PART_OF]->(c:Country)
```

```
WHERE country IN [countryA, countryB]
RETURN person
       ,countryA
       ,countryB
       ,COLLECT(c) AS cities
)
```

In the view hereunder, we have a path traversal from the predicate p indicating the node person to a city along the edge with the label `IS_LOCATED_IN`. After the path traversal, the friend its id is returned along with the country.

```
CREATE QUERY city_VIEW3 AS (
FROM city_VIEW2
MATCH (p:Person where  p.id <> f:friend.id)-[:IS_LOCATED_IN]
     ->(c:City WHERE c.id <> c.id)
WHERE p.id == country_VIEW1.p.id
RETURN DISTINCT f
               ,countryA
               ,countryB
)
```

In the query below there are two distinct path traversals. In the first path traversal, there is a left path traversal from the node message to a friend to see to which individual the message belongs, and traverses along the edge with the label `HAS_CREATOR`. The second path traversal is a right path traversal to retrieve the country from where the message is sent .After traversal, the id is retrieved from the friend and to the related country a value of 1 is assigned if the message is sent from the same country as the location of a person or 0 if the message is sent from a different location.

```
FROM LDBC_SNB,city_VIEW3,city_VIEW2,city_VIEW1
MATCH (f:Friend)<-[:HAS_CREATOR]-(m:message),
      (m:message)-[:IS_LOCATED_IN]->(c:country)
WHERE $endDate > m.creationDate >= $startDate AND
      country IN [countryA, countryB] AND city_VIEW3.f.id == f.id
RETURN f,
     CASE WHEN country=countryA THEN 1 ELSE 0 END AS messageA,
     CASE WHEN country=countryB THEN 1 ELSE 0 END AS messageB
     sum(messageA) AS countA, sum(messageB) AS countB

GROUP BY f.id
HAVING countA >0

AND
       countB>0
```

The last query is a union of union of all the views together that retrieves the respective variables

```
% CALL {
% city_VIEW1
% UNION
% city_VIEW2
% UNION
% city_VIEW3
% RETURN country_VIEW1.f.id AS friend,
%       country_VIEW2.f.firstName AS friend_FirstName,
%       country_VIEW2.f.lastName AS friend_LastName,
%       countA,
%       countB,
%       countA + countA AS ABCount
% }
```

```
% ORDER BY ABCount DESC
%          , friend ASC
% LIMIT 20


FROM LDBC_SNB,city_VIEW3,city_VIEW2,city_VIEW1
MATCH (f:Friend)<-[:HAS_CREATOR]-(m:message),
      (m:message)-[:IS_LOCATED_IN]->(c:country)
WHERE $endDate > m.creationDate >= $startDate AND
      country IN [countryA, countryB] AND city_VIEW3.f.id == f.id
// RETURN f,
     CASE WHEN country=countryA THEN 1 ELSE 0 END AS messageA,
     CASE WHEN country=countryB THEN 1 ELSE 0 END AS messageB
//      sum(messageA) AS countA, sum(messageB) AS countB

// GROUP BY f.id
HAVING countA >0 AND countB>0

// CALL {
// city_VIEW1
// UNION
// city_VIEW2
// UNION
// city_VIEW3

// Perhahps we can remove the call Option

RETURN country_VIEW1.f.id AS friend,
       country_VIEW2.f.firstName AS friend_FirstName,
       country_VIEW2.f.lastName AS friend_LastName,
       sum(messageA) AS countA
       ,sum(messageB) AS countB
       countA + countA AS ABCount
// }
ORDER BY ABCount DESC
        , friend ASC
LIMIT 20
```

```
Select  person,countryX,countryY
FROM LDBC_SNB
MATCH (countryX:Country {name: $countryXName }),
      (countryY:Country {name: $countryYName }),
      (person:Person {id: $personId })
LIMIT 1
```

```
SELECT  person,countryX,countryY,COLLECT(city) AS cities
FROM LDBC,country_VIEW
MATCH (city:City)-[:IS_PART_OF]->(country:Country)
WHERE country IN [countryX, countryY]
```

```
SELECT  DISTINCT person2,countryX,countryY
FROM  LDBC_SNB,part_of_VIEW
MATCH (person)-[:KNOWS]{1,2}-(person2:person)-[:IS_LOCATED_IN]->(city)
WHERE person=person2 AND NOT city IN citiy
```

```
SELECT person2.id AS person2Id, person2.firstName AS person2FirstName
      ,person2.lastName AS person2LastName,sum(messageX) AS xCount
      ,sum(messageY) AS yCount,xCount + yCount AS xyCount
FROM LDBC_SNB,city1_VIEW,city2_VIEW,city3_VIEW
MATCH (person2)<-[:HAS_CREATOR]-(message),
      (message)-[:IS_LOCATED_IN]->(country)
WHERE $endDate > message.creationDate >= $startDate AND
      country IN [countryX, countryY]
CASE WHEN country=countryX
     THEN 1
     ELSE 0
     END AS messageX,
CASE WHEN country=countryY
     THEN 1
     ELSE 0
     END AS messageY
WHERE xCount>0 AND yCount>0
ORDER BY xyCount DESC, person2Id ASC
LIMIT 20
```

73: Final Query

9.2.4 Interactive Complex 4

```
FROM LDBC_SNB
MATCH (p:Person {id: $personId})
      -[:KNOWS]-(f:Person) <-[:HAS_CREATOR]-
       (post:Post)-[:HAS_TAG]->(tag)
CASE $tag :
       WHEN $endDate > post.creationDate >= $startDate THEN 1
       ELSE 0
     END AS valid,
     CASE
       WHEN $startDate > post.creationDate THEN 1
       ELSE 0
     END AS inValid
WHERE countOfPost>0
AND inValidCountOfPost=0
RETURN tag.name AS tagName,
       ,sum(valid) AS countOfPost
       ,sum(inValid) AS inValidCountOfPost
ORDER BY postCount DESC
         ,tagName ASC
LIMIT 10
```

The construction of a view is omitted since there is only one MATCH clause. In the MATCH clause a path is traversed from the node Person to another person traversing the first edge label named KNOWS. Afterward, a left traversal is done from the node post to person along the label named `HAS_CREATOR`. From the node post, there is another traversal going out or coming in from post to tag and is a right traversal along the edge to tag. In short, this query verifies whether a new post or comment is interchanged between two people.

```
SELECT tag.name AS tagName,sum(valid) AS countOfPost,sum(inValid) AS inValidCountOfPost
FROM LDBC_SNB
    MATCH (p:Person {id: $personId })-[:KNOWS]-(f:Person) <-[:HAS_CREATOR]-(post:Post)-[:HAS_TAG]->(tag)
       CASE $tag :
       WHEN $endDate > post.creationDate >= $startDate THEN 1
       ELSE 0
     END AS valid,
     CASE
       WHEN $startDate > post.creationDate THEN 1
       ELSE 0
     END AS inValid
WHERE countOfPost>0 AND inValidCountOfPost=0
ORDER BY postCount DESC
         ,tagName ASC
LIMIT 10
```

74: TODO

9.2.5 Interactive Complex 5

In the view `know_VIEW`, the path of KNOWS is traversed between person and friend of lengths of between 1 and 2 and returns the distinct friends.

```
CREATE QUERY know_VIEW as (
MATCH (p:Person { id: $personId })-[:KNOWS]-(f: friend) {1,2}
WHERE
     person <> f
Return DISTINCT f
)
```

From the view `know_VIEW` the path with the label `HAS_MEMBER` is traversed from the node friend to the node forum. The traversal is the right one. Upon that path, all the friends who have joined a forum are accumulated and returned.

```
CREATE QUERY forum_VIEW AS (
FROM know_VIEW
MATCH (f)<-[m:HAS_MEMBER]-(forum)
WHERE
    m.joinDate > $minDate

AND know_VIEW.f.id == f.id
RETURN forum
      ,COLLECT(f) AS friends
)
```

In the Query below from the edge forum, there is a path traversal along the edge of label `CONTAINER_OF` to post (left traversal). Another left traversal is done from the node post to a friend along with the edge

label HAS_CREATOR, to see whether a person posted in the forum. Eventually, the name of the forum and the number of posts posted by a person is returned.

```
FROM know_VIEW,forum_VIEW
OPTIONAL MATCH (f)<-[:HAS_CREATOR]-(post)<-[:CONTAINER_OF]-(forum)
WHERE
    f IN friends
RETURN
    f.id as person
    forum.title AS nameForum,
    count(post) AS countPost
ORDER BY
    postCount DESC,
    forum.id ASC
LIMIT 20
```

```
SELECT  DISTINCT f
MATCH (p:Person { id: $personId })-[:KNOWS]-(f: friend) {1,2}
WHERE
     person <> f
```

```
SELECT forum, COLLECT(f) AS friends
FROM know_VIEW
MATCH (f)<-[m:HAS_MEMBER]-(forum)
WHERE
    m.joinDate > $minDate
AND know_VIEW.f.id == f.id
```

```
//final Query
SELECT f.id as person, forum.title AS nameForum, count(post) AS countPost
FROM know_VIEW,forum_VIEW
OPTIONAL MATCH (f)<-[:HAS_CREATOR]-(post)<-[:CONTAINER_OF]-(forum)
WHERE
    f IN friends
ORDER BY
    postCount DESC,
    forum.id ASC
LIMIT 20
```

77: Final Query

9.2.6 Interactive Complex 6

In the view below `tag_VIEW`, all the tags-id are retrieved and returned. Within the MATCH clause, there is no path traversal just selection of the specific node Tag

```
Create Query tag_VIEW AS (
MATCH (knowTag:Tag { name: $tagName })
WITH knowTag.id as knownTagId
)
```

The query `knowTag_VIEW` traverses from the node person to friend and does at most 2 undirected traversals between 1 and 2. The traversal along the path with edge label: KNOWS makes sure that each person is not a friend within himself or herself. In the end, the relatedTagID and the friendID information are returned.

```
CREATE Query knowTag_VIEW as (
FROM tag_VIEW
MATCH (p:Person { id: $personId })-[:KNOWS]-(f: friend){1,2}
WHERE  p<>f AND tag_VIEW.knownTagId == knownTagId
RETURN
    knownTagId,
    (distinct f) as friends
)
```

In the final query, the path traversal is executed from the friend node to the post node along the path `HAS_CREATOR`. There is also path traversal from post to TagID. Do note that there are two path traversals from post to Tag, indicating that they are two different tags. At last, the query returns the tag names and the number of posts posted by each person.

```
FROM tag_VIEW, knowTag_VIEW
MATCH (f)<-[:HAS_CREATOR]-(post:Post),
        (post)-[:HAS_TAG]->(t1:Tag{id: knownTagId}),
        (post)-[:HAS_TAG]->(t2:Tag)
    WHERE  t1 <> t2
RETURN
   t.name as nametag,
 count(post) as countPost
ORDER BY
    nametag DESC,
    countPost ASC
LIMIT 10
```

```
SELECT  knowTag.id as knownTagId
FROM LDBC_SNB
     MATCH (knowTag:Tag { name: $tagName })
```

9.2.7 Interactive Complex 7

For the query above, there is no view necessary since it consists of one query. There are two match clauses. In the first match clause, a left traversal is done from comment to messageId, and also a right traversal to the node person along with the edge label `HAS_CREATOR`. In the second match, which is an optional MATCH,

```
SELECT knownTagId, (distinct f) as friends
FROM tag_VIEW
     MATCH (p:Person { id: $personId })-[:KNOWS]-(f: friend){1,2}
     WHERE  p<>f AND tag_VIEW.knownTagId == knownTagId
```

```
SELECT  t.name as nametag,count(post) as countPost
FROM tag_VIEW, knowTag_VIEW
MATCH (f)<-[:HAS_CREATOR]-(post:Post),
      (post)-[:HAS_TAG]->(t1:Tag{id: knownTagId}),
      (post)-[:HAS_TAG]->(t2:Tag)
WHERE  t1 <> t2
ORDER BY
    nametag DESC,
    countPost ASC
LIMIT 10
```

80: final query

a right traversal is carried out from the message node to the person, and then another right traversal to the person node along the path `HAS_CREATOR`. From the node person, there is another traversal to the node person. This seems confusing, but the query verifies whether a reply is given from person A to person B. In addition, a verification is done to see whether two people know each other.

```
FROM LDBC_SNB
MATCH (m:Message {id: $messageId })<-[:REPLY_OF]-(c:Comment)-
      [:HAS_CREATOR]->(p:Person)
OPTIONAL MATCH (m)-[:HAS_CREATOR]->(a:Person)-[r:KNOWS]-(p)
RETURN c.id AS commentId,
        c.content AS Content,
        c.creationDate AS CreationDate,
        p.id AS AuthorId,
        p.firstName AS replyFirstName,
        p.lastName AS replyLastName,
        CASE r
            WHEN null THEN false
            ELSE true
       END AS KnowsOriginalMessageAuthor
ORDER BY creationDate DESC
         ,AuthorId
```

9.3 Interactive 7

9.3.1 Interactive Complex 8

As in Query 7, there construction of a view is omitted as well. Here the match clause has four nodes. First, from p1 indicating the person node, there is a path traversal to the message node, which is a left traversal. Also from the message node, there is a left traversal from comment to message and at last from comment node to person node there is a right traversal along with the `HAS_CREATOR` label. In short, the match clause

```
SELECT c.id AS commentId, c.content AS Content, c.creationDate AS CreationDate, p.id AS AuthorId,
        p.firstName AS replyFirstName,p.lastName AS replyLastName,
        CASE r
            WHEN null THEN false
            ELSE true
       END AS KnowsOriginalMessageAuthor
FROM LDBC_SNB
MATCH (m:Message {id: $messageId })<-[:REPLY_OF]-(c:Comment)-[:HAS_CREATOR]->(p:Person)
OPTIONAL MATCH (m)-[:HAS_CREATOR]->(a:Person)-[r:KNOWS]-(p)
ORDER BY creationDate DESC
         ,AuthorId
```

81: final query

in principle is a follow-up of the query executed before. The clause looks up at all the replies between two people.

```
FROM LDBC_SNB
MATCH (p1:Person {id: $personId})<-[:HAS_CREATOR]-(:Message)<-[:REPLY_OF]
     -(c:Comment)-[:HAS_CREATOR]->(p2:Person)

RETURN
    p1.id AS personId,
    p1.firstName AS personFirstName,
    p1.lastName AS personLastName,
    c.creationDate AS commentCreationDate,
    c.id AS commentId,
    c.content AS content
ORDER BY
    commentCreationDate DESC,
    commentId ASC
LIMIT 20
```

```
//
SELECT p1.id AS personId, p1.firstName AS personFirstName, p1.lastName AS personLastName,
       c.creationDate AS commentCreationDate,c.id AS commentId,c.content AS content
FROM LDBC_SNB
MATCH (p1:Person {id: $personId})<-[:HAS_CREATOR]-(:Message)<-[:REPLY_OF]
      -(c:Comment)-[:HAS_CREATOR]->(p2:Person)
ORDER BY
    commentCreationDate DESC,
    commentId ASC
LIMIT 20
```

82: ToDo

9.3.2 Interactive Complex 9

The match clause of `friends_VIEW` verifies the friends of friends and makes sure that the friendship is not a reflexive relationship. The path traversal is done to a path of at most 2 lengths, with a minimum of 1

length.

```
CREATE Query friends_VIEW as (
MATCH (p1:Person {id: $personId })-[:KNOWS]-(p2:Person){1,2}
WHERE f<>p1
RETURN  COLLECT (distinct f) as friends
)
```

The query below is a continuation of the one above, and does a right path traversal from the message node to the friend node. The combination of this query, and the constructed view above verifies whether a new message is sent between two friends and the type of message. The RETURN clause returns the details of the owner of that message, and the content of the message

```
FROM friends_VIEW,LDBC_SNB
MATCH (f) <-[:HAS_CREATOR]-(m:Message)
    WHERE m.creationDate < $maxDate
RETURN
    f.id AS personId,
    f.firstName AS FirstName,
    f.lastName AS LastName,
    m.id AS commentId,
    coalesce(m.content,m.imageFile) AS Content,
    m.creationDate AS contentCreationDate
ORDER BY
    contentCreationDate DESC,
    message.id ASC
LIMIT 20
```

```
SELECT  COLLECT (distinct f) as friends
FROM LDBC_SNB
MATCH (p1:Person {id: $personId })-[:KNOWS]-(p2:Person){1,2}
WHERE f<>p1
```

```
SELECT  f.id AS personId,f.firstName AS FirstName,f.lastName AS LastName, m.id AS commentId,
        coalesce(m.content,m.imageFile) AS Content,
        m.creationDate AS contentCreationDate
FROM friends_VIEW
    MATCH (f) <-[:HAS_CREATOR]-(m:Message)
    WHERE m.creationDate < $maxDate
ORDER BY
    contentCreationDate DESC,
    message.id ASC
LIMIT 20
```

84: Final Query

9.3.3 Interactive Complex 10

In the first query view below we do two-match clauses. In the first match clause, there is an undirected path traversal from both the person node to the friend node traversed along the path KNOWS that traverses between 1 or 2 paths. In the second MATCH clause, a traversal is done from the person node to the friend node along the path of KNOWS.

```
Create Query friend_VIEW as {
MATCH (p:Person {id: \$personId})-[:KNOWS]-(friend){2,..}
       (friend)-[:IS_LOCATED_IN]->(city:City)
OPTIONAL MATCH   (friend where friend <> person)-[:KNOWS]-(person)
WHERE  (birthday.month=$month AND birthday.day>=21) OR
        (birthday.month=($month%12)+1 AND birthday.day<22)
RETURN
      person
      ,city
      ,friend
      ,datetime({f.birthday}) as birthday
}
```

```
Create Query post_VIEW as {
OPTIONAL MATCH (friend)<-[:HAS_CREATOR]-(post:Post)
RETURN friend
       ,city
       ,COLLECT(post) AS posts
       ,person
}
```

```
Create Query postExtend_VIEW as {
FROM post_VIEW
Return city,
     size(posts) AS postCount,
     friend.id AS personId,
     friend.firstName AS personFirstName,
     friend.lastName AS personLastName,
     commonPostCount - (postCount - commonPostCount) AS commonInterestScore,
     friend.gender AS personGender,
     city.name AS personCityName
WHERE EXIST{
  FROM LBC_SNB
  MATCH
  (p)-[:HAS_TAG]->()<-[:HAS_INTEREST]-(person))

  Return  AS commonPostCount
   }
ORDER BY commonInterestScore DESC
        , personId ASC
LIMIT 10
}
```

Do notice that the `postExtend_VIEW` query is an extension of the `post_VIEW`. The `post_VIEW` first returns the friend, city, and posts values through the optional match, which is a right traversal from the post node to the friend node along the path of the edge label `HAS_CREATOR`.Afterward, the extended view carries out an inner match by traversing the path from person node to another person node. This, initially might

not make sense. But this inner traversal is done to see whether person1 might know person2 based on their location, posts, and comments.

```
SELECT person,city,friend,datetime({f.birthday}) as birthday
FROM LDBC_SNB
    MATCH (p:Person {id: $personId})-[:KNOWS]-(friend){2,..}
       (friend)-[:IS_LOCATED_IN]->(city:City)
    OPTIONAL MATCH   (friend where friend <> person)-[:KNOWS]-(person)
    WHERE  (birthday.month=$month AND birthday.day>=21) OR
           (birthday.month=($month/12)+1 AND birthday.day<22)
```

```
SELECT friend,city,COLLECT(post) AS posts,person
FROM LDBC_SNB
OPTIONAL MATCH (friend)<-[:HAS_CREATOR]-(post:Post)
```

```
SELECT city,size(posts) AS postCount,friend.id AS personId,
       friend.firstName AS personFirstName,friend.lastName AS personLastName,
       commonPostCount - (postCount - commonPostCount) AS commonInterestScore,
       friend.gender AS personGender,city.name AS personCityName
FROM
WHERE EXIST{
  SELECT AS commonPostCount
  FROM LBC_SNB
  MATCH
  (p)-[:HAS_TAG]->()<-[:HAS_INTEREST]-(person)

   }
ORDER BY commonInterestScore DESC
        , personId ASC
LIMIT 10
```

9.3.4 Interactive Complex 11

The view retrieves people who are friends. The construction of this is done by first traversing the path with edge label KNOWS from predicate p1 (person node 1) to p2(another person).

The query above traverses the path from company to country name along the path `IS_LOCATED_IN` to the node country,which is a left traversal. There is also an undirected traversal from the friend node towards the company node.In short, the query retrieves the job referrals.

```
Create Query personFriend_VIEW AS (
FROM LDBC_SNB
MATCH (p1:Person {id: \$personId })-[:KNOWS]-(p2:Person){1,2}
WHERE not(person <> p2)
RETURN DISTINCT friend
)
```

```
FROM LDBC_SNB,personFriend_VIEW
MATCH (:Country {name: \$countryName })
      <-[:IS_LOCATED_IN] <- (friend)
      -[workAt:WORK_AT] -(company:Company)
WHERE workAt.workFrom < \$workFromYear
RETURN
        f.id AS personId,
        f.firstName AS personFirstName,
        f.lastName AS personLastName,
        company.name AS organizationName,
        workAt.workFrom AS organizationWorkFromYear
ORDER BY
        organizationWorkFromYear ASC,
        toInteger(personId) ASC,
        organizationName DESC
LIMIT 10
```

```
SELECT  DISTINCT friend
FROM LDBC_SNB
MATCH (p1:Person {id: $personId })-[:KNOWS]-(p2:Person){1,2}
WHERE (person <> p2)
```

```
SELECT f.id AS personId,f.firstName AS personFirstName,f.lastName AS personLastName,company.name AS organizationName,workAt.wor
FROM LDBC_SNB,personFriend_VIEW
MATCH (:Country {name: $countryName }) <-[:IS_LOCATED_IN] <- (friend) -[workAt:WORK_AT] -(company:Company)
WHERE workAt.workFrom < $workFromYear
ORDER BY
        organizationWorkFromYear ASC,
        toInteger(personId) ASC,
        organizationName DESC
LIMIT 10
```

89: Final Query

9.3.5 Interactive Complex 12

In the query above we have the MATCH clause that does a traversal from the baseTagclass that comes from the tag node in a left manner and traveled through the `HAS_TYPE` label. During traversal, at least one path is traveled. In the second query, we have the traversal from the comment node towards the

```
FROM LDBC_SNB
MATCH (baseTagClass:TagClass)<-[:HAS_TYPE|IS_SUBCLASS_OF]{0,..}-(tag:Tag)
WHERE tag.name = $tagClassName OR baseTagClass.name = $tagClassName
RETURN collect(tag.id) as tags
```

```
MATCH (:Person {id: \$personId})-[:KNOWS]-(friend:Person)
     <-[:HAS_CREATOR]-(comment:Comment)
     -[:REPLY_OF]->(:Post)-[:HAS_TAG]->(tag:Tag)
WHERE tag.id in tags
RETURN
    friend.id AS personId,
    friend.firstName AS personFirstName,
    friend.lastName AS personLastName,
    COLLECT(DISTINCT tag.name) AS tagNames,
    COUNT(DISTINCT comment) AS replyCount
ORDER BY
    replyCount DESC,
    toInteger(personId) ASC
LIMIT 20
```

friend node first through the label `HAS_CREATOR`. From the friend nodes, there is also an undirected traversal towards the person node. Moreover, there are two right traversals from the comment node towards the post node, which is the first through the POST label. The second traversal is the traversal from the comment node towards the post node, that travels through the path `REPLY_OF`. Afterward, a second traversal is done from the post node towards the tag node through the `HAS_TAG` path.

```
SELECT collect(tag.id) as tags
FROM LDBC_SNB
      MATCH (baseTagClass:TagClass)<-[:HAS_TYPE|IS_SUBCLASS_OF]{0,..}-(tag:Tag)
WHERE tag.name = $tagClassName OR baseTagClass.name = $tagClassName

SELECT friend.id AS personId,friend.firstName AS personFirstName,friend.lastName AS personLastName, COLLECT(DISTINCT tag.name)
FROM
     MATCH (:Person {id: $personId })-[:KNOWS]-(friend:Person)<-[:HAS_CREATOR]-(comment:Comment)-[:REPLY_OF]->(:Post)-[:HAS_TAG
WHERE tag.id in tags
ORDER BY
    replyCount DESC,
    toInteger(personId) ASC
LIMIT 20
```

90: ToDo

9.3.6 Interactive Complex 13

```
FROM LDBC_SNB
MATCH(p1:Person {id: $person1Id}),(p2:Person {id: $person2Id}),
    path = shortest((p1)-[:KNOWS]-(p2)){*}
RETURN
    CASE path IS NULL
        WHEN true THEN -1
        ELSE length(path)
    END AS shortestPathLength
```

The shortest path available **SCHRIJF DIE PAGINA NUMMER OP OF REFERENCE** functionality is used here and is traversed at most between 0 or more repetitions. As a final result, the length of the path is returned.

```
SELECT  CASE path IS NULL  WHEN true THEN -1 ELSE length(path)
        END AS shortestPathLength
FROM LDBC_SNB
     MATCH( p1:Person {id: $person1Id}),(p2:Person {id: $person2Id}),
     path = SHORTEST((p1)-[:KNOWS]-(p2)){*}
```

91: ToDo

9.3.7 Interactive Short 1

The query below returns the results of a single person by traversing the path labeled `IS_LOCATED_IN`to the left from person to city node.

```
MATCH (p1:Person {id: $personId })-[:IS_LOCATED_IN]->(c:City)
RETURN
    p1.firstName AS firstName,
    p1.lastName AS lastName,
    p1.birthday AS birthday,
    p1.locationIP AS locationIP,
    p1.browserUsed AS browserUsed,
    p1.id AS cityId,
    p1.gender AS gender,
    p1.creationDate AS creationDate
```

// IS2. Recent messages of a person

```
SELECT p1.firstName AS firstName,p1.lastName AS lastName,p1.birthday AS birthday,p1.locationIP AS locationIP,p1.browserUsed AS
FROM LDBC_SNB
MATCH (p1:Person {id: $personId })-[:IS_LOCATED_IN]->(c:City)
```

92: Profile of a person

9.3.8 Interactive Short 2

In the query below the view carries out a left traversal from the unlabeled message node to the person node in a left manner. This traversal is done to obtain all the people who are the creator of a message.

```
Create Query message_VIEW as (
MATCH (:Person {id: $personId})<-[:HAS_CREATOR]-(:message)
Return
 message,
 message.id AS messageId,
 message.creationDate AS messageCreationDate
ORDER BY messageCreationDate DESC, messageId ASC
```

```
LIMIT 10
)
```

```
From message_VIEW, LDBC_SNB
MATCH (message)-[:REPLY_OF]->(post:Post)-[:HAS_CREATOR]->(person)
RETURN  messageId,
 ,coalesce(message.imageFile,message.content) AS messageContent
,messageCreationDate
 ,post.id AS postId,
 ,person.id AS personId,
 ,person.firstName AS FirstName,
 ,person.lastName AS LastName
ORDER BY messageCreationDate DESC
        ,messageId ASC
```

From the Match clause above in query 2, there is a traversal from the message node to the post node traveled along with the `REPLY_OF` label. Do note that this is a left path traversal. After the left path traversal, there is a left traversal from the person node to the post node along the path of `HAS_CREATOR` to the person node. This traversal looks for the people that replied to a message. // IS2. Recent messages of a person

```
//  Query message_VIEW
SELECT message,message.id AS messageId,message.creationDate AS messageCreationDate
MATCH (:Person {id: $personId})<-[:HAS_CREATOR]-(:message)
ORDER BY messageCreationDate DESC, messageId ASC
LIMIT 10


\begin{listing}[!ht]
\begin{minted}
[
frame=lines,
framesep=2mm,
baselinestretch=1.2,
bgcolor=LightGray,
fontsize=\footnotesize,
linenos
]
{cypher}
SELECT  messageId,coalesce(message.imageFile,message.content) AS messageContent ,messageCreationDate,post.id AS postId,person.i
From LDBC_SNB,...
MATCH (message)-[:REPLY_OF]->(post:Post)-[:HAS_CREATOR]->(person)
ORDER BY messageCreationDate DESC
        ,messageId ASC
```

93: ToDo

9.3.9 Interactive Short 3

```
FROM LDBC_SNB
MATCH (n:Person {id: \$personId })-[r:KNOWS]-(f:friend)
RETURN
    f.id AS personId,
    f.firstName AS firstName,
    f.lastName AS lastName,
    r.creationDate AS friendshipDate
ORDER BY
    friendshipDate DESC,
    CAST(personId AS INTEGER) ASC
```

This query retrieves all the friends of a person through the undirected traversal between the node person and the another node friend by traversing the path KNOWS.

```
SELECT f.id AS personId,f.firstName AS firstName,f.lastName AS lastName,r.creationDate AS friendshipDate
FROM LDBC_SNB
     MATCH (n:Person {id: $personId })-[r:KNOWS]-(f:friend)
ORDER BY
    friendshipDate DESC,
    CAST(personId AS INTEGER) ASC
```

94: Friends of a person

9.3.10 Interactive Short 4

```
FROM LDBC_SNB
MATCH (message:Message {id:  \$messageId })
RETURN
    message.creationDate as messageCreationDate,
    COALESCENCE (m.content, m.imageFile) as messageContent
```

95: message contents

The query selects the message node and afterwards returns the creation date, that a message was created and message content. The message content consist out of a COALESCENCE clause. This mean that wehn content is empty then the imageFile is chosen as the messageContent,and vice versa. **TODO**

9.3.11 Interactive Short 5

The query traverses from the node message to the person in a left manner along the path `HAS_CREATOR` in retrieving the details of the creator of a message.

```
SELECT
    message.creationDate as messageCreationDate,
    COALESCENCE (m.content, m.imageFile) as messageContent
FROM LDBC_SNB
     MATCH (message:Message {id:  $messageId })
```

96: ToDo

```

FROM LDBC_SNBI
MATCH (m:Message {id: \$messageId })-[:HAS_CREATOR]->(p1:Person)
RETURN
    p1.id AS personId,
    p1.firstName AS firstName,
    p1.lastName AS lastName

```

```
SELECT
FROM LDBC_SNB
    MATCH (forum:Forum {id: $forumId}), (person:Person {id: $personId})
    CREATE (forum)-[:HAS_MEMBER {joinDate: $joinDate}]->(person)
```

97: ToDo

9.3.12 Interactive Short 6

```

FROM LDBC_SNBI
OPTIONAL MATCH (m:Message {id: $messageId})-[:REPLY_OF]{0,}->(p:Post)<-[:CONTAINER_OF]-(f:Forum)-[:HAS_MODERATOR]->(p1:Person)
RETURN
    f.id AS forumId,
    f.title AS forumTitle,
    mod.id AS moderatorId,
    mod.firstName AS FirstName,
    mod.lastName AS LastName

```

An optional match is used in the query rather than an if clause in GSQL.The use of an OPTIONAL MATCH will result in a returned NULL value if there is no value found for such. From the message node, there is a right traversal to the post node along the path of `REPLY_OF`. In addition a left path traversal is present, and travels along the path `CONTAINER_OF` from the forum node to the post node. Additionally another right traversal is done from the forum node towards the person node, through the path `HAS_MODERATOR` . Since Comments are not directly contained in Forums, for Comments, the Forum containing the original Post in the thread to which the Comment is replying is returned. // IS6. Forum of a message

```
SELECT f.id AS forumId, f.title AS forumTitle, mod.id AS moderatorId, mod.firstName AS FirstName, mod.lastName AS LastName
FROM LDBC_SNBI
     OPTIONAL MATCH (m:Message {id: $messageId })-[:REPLY_OF]{0,}->(p:Post)<-[:CONTAINER_OF]-(f:Forum)-[:HAS_MODERATOR]->(p1:Pe
```

98: ToDo

9.3.13 Interactive Short 7

```
FROM LDBC_SNBI
MATCH (m:Message {id: \$messageId})<-[:REPLY_OF]-(c:Comment)
      -[:HAS_CREATOR]->(p:Person)
OPTIONAL MATCH (m)-[:HAS_CREATOR]->(p2:Person)-[r:KNOWS]-(p1)
RETURN c.id AS commentId,
c.content AS content,
c.creationDate AS creationDate,
p.id AS AuthorId,
p.firstName AS AuthorFirstName,
p.lastName AS AuthorLastName,
CASE r
     WHEN null THEN false
    ELSE true
        END AS replyAuthorKnowsOriginalMessageAuthor
ORDER BY commentCreationDate DESC, AuthorId
```

In the match clause, the query looks at whether there is a reply to a message or not by carrying out a left path traversal from the comment node to the message node along the path of `REPLY_OF`. From the comment node also a right traversal is done to the person node through the path of `HAS_CREATOR`. Furthermore, there is an OPTIONAL MATCH from another person to another one to see if the message exchanged between two people do know each other.

// IS7. Replies of a message

```
SELECT c.id AS commentId,c.content AS content,c.creationDate AS creationDate,p.id AS AuthorId,p.firstName AS AuthorFirstName,p.
       CASE r WHEN null THEN false ELSE true  END AS replyAuthorKnowsOriginalMessageAuthor
FROM LDBC_SNBI
     MATCH (m:Message {id: $messageId })<-[:REPLY_OF]-(c:Comment)-[:HAS_CREATOR]->(p:Person)
     OPTIONAL MATCH (m)-[:HAS_CREATOR]->(p2:Person)-[r:KNOWS]-(p1)
ORDER BY commentCreationDate DESC, AuthorId
```

99: ToDo

9.3.14 Interactive Insert 1

In the query below a selection is done on the city node. After selecting that node a temporary view is constructed, in which the person's details are included. Also in the view a path is traversed from the person

```
CREATE QUERY tag_VIEW AS {
FROM LDBC_SNBI
MATCH (c:City {id: $cityId})
CREATE (p:Person {
    id: $personId,
    firstName: $personFirstName,
    lastName: $personLastName,
    gender: $gender,
    birthday: $birthday,
    creationDate: $creationDate,
    locationIP: $locationIP,
    browserUsed: $browserUsed,
    languages: $languages,
    email: $emails
  })-[:IS_LOCATED_IN]->(c)
RETURN  $tagIds AS tagId
}
```

```
Create Query as  interest_VIEW AS {
FROM tag_VIEW
  MATCH (t:Tag {id: tagId})
  CREATE (p)-[:HAS_INTEREST]->(t)
 Return p, count(*) AS times,
        \$studyAt AS place
}
```

node towards the city node. The traversal is executed through the path named `IS_LOCATED_IN` . After retrieval of the id's with their respective tags, the following query selects the results obtained, which are id's of the tag and creates another node view which is traversed from the person to the tag node, along the path named `HAS_INTEREST`

In the view `interest_VIEW` there is a selection of the organization node. Also in that sub-query, another node view is created from the node p where there is a traversal from the person node to the organization node, at which a traversal is done along the path of `STUDY_AT`. In addition, there is another path created from the person node to the organization node, and a path traversal is done through the `WORKS_AT` path.

```
FROM interest_VIEW,LDBC_SNB
MATCH (o:Organisation {id: s[0]})
CREATE (p)-[:STUDY_AT {classYear: s[1]}]->(o)
Return p
      ,count(*) AS times2
      ,$workAt AS w
```

```
MATCH (comp:Organisation {id: w[0]})
  CREATE (p)-[:WORKS_AT {workFrom: w[1]}]->(comp)
```

```
SELECT  $tagIds AS tagId
FROM LDBC_SNBI
     MATCH (c:City {id: $cityId})
     CREATE (p:Person {id: $personId,firstName: $personFirstName,lastName: $personLastName,gender: $gender,birthday: $birthday,
         browserUsed: $browserUsed,languages: $languages,email: $emails })-[:IS_LOCATED_IN]->(c)
```

```
SELECT p, count(*) AS times,
FROM
    MATCH (t:Tag {id: tagId})
    CREATE (p)-[:HAS_INTEREST]->(t)
        $studyAt AS place
```

```
SELECT p, count(*) AS times,$studyAt AS place
FROM
    MATCH (t:Tag {id: tagId})
    CREATE (p)-[:HAS_INTEREST]->(t)
```

102: ToDo

Can we construct it like this? the workAt variable (ask)

9.3.15 Interactive Insert 2

TODO

```
SELECT p,count(*) AS times2,$workAt AS w MATCH (comp:Organisation {id: w[0]})
FROM interest_VIEW,LDBC_SNB
     MATCH (o:Organisation {id: s[0]})
     CREATE (p)-[:STUDY_AT {classYear: s[1]}]->(o)
     CREATE (p)-[:WORKS_AT {workFrom: w[1]}]->(comp)
```

103: ToDo

```
SELECT
From LDBC_SNB
     MATCH (person:Person {id: $personId}), (comment:Comment {id: $commentId})
     CREATE (person)-[:LIKES {creationDate: \$creationDate}]->(comment)
```

9.3.16 Interactive Insert 3

```
From LDBC_SNB
MATCH (person:Person {id: $personId}), (comment:Comment {id: $commentId})
CREATE (person)-[:LIKES {creationDate: $creationDate}]->(comment)
```

The query above first selects the person node and the comment node and then creates path traversal from the person node towards the comment node in a right traversal manner carried out through the constructed path named LIKES.

```
SELECT
From LDBC_SNB
     MATCH (person:Person {id: $personId}), (comment:Comment {id: $commentId})
     CREATE (person)-[:LIKES {creationDate: $creationDate}]->(comment)
```

104: ToDo

9.3.17 Interactive Insert 4

```
FROM LDBC_SNB
MATCH (p:Person {id: $moderatorPersonId})
CREATE (f:Forum {id: $forumId, title: $forumTitle, creationDate: $creationDate})-[:HAS_MODERATOR]->(p)
Return $tagIds AS tagId
```

The construction of views is omitted. The first query selects the person node and then creates a path traversal from the forum node towards the person node, along the path of HAS_MODERATOR. From the returned values of the query above, query 2 continues or starts with the tag node, and from there on a traversal is created along the path of HAS_TAG from the forum node towards the tag node.

```
MATCH (tag:Tag {id: tagId})
CREATE (f)-[:HAS_TAG]->(tag)
```

```
SELECT $tagIds AS tagId
FROM LDBC_SNB
     MATCH (p:Person {id: $moderatorPersonId})
     CREATE (f:Forum {id: $forumId, title: $forumTitle, creationDate: $creationDate})-[:HAS_MODERATOR]->(p)

SELECT ...
FROM ...
MATCH (tag:Tag {id: tagId})
CREATE (f)-[:HAS_TAG]->(tag)
```

105: ToDo

9.3.18 Interactive Insert 5

```
FROM LDBC_SNB
MATCH (forum:Forum {id: $forumId}), (person:Person {id: $personId})
CREATE (forum)-[:HAS_MEMBER {joinDate: $joinDate}]->(person)
```

A selection is made in the match clause, selecting the forum node and the person node. After that, a path is created from the forum node towards the person node in a right manner along the path of HAS_MEMBER.

```
SELECT
FROM LDBC_SNB
     MATCH (forum:Forum {id: $forumId}), (person:Person {id: $personId})
CREATE (forum)-[:HAS_MEMBER {joinDate: $joinDate}]->(person)
```

106: ToDo

9.3.19 Interactive Insert 6

```
FROM  LDBC_SNB
MATCH (m:Message {id: \$messageId })-[:REPLY_OF*]{0,..}->(p:Post)<-[:CONTAINER_OF]-(f:Forum)-[:HAS_MODERATOR]->(mod:Person)
RETURN
    f.id AS forumId,
    f.title AS forumTitle,
    mod.id AS moderatorId,
    mod.firstName AS moderatorFirstName,
    mod.lastName AS moderatorLastName
```

107: Moderator

In here the query first selects the message node and from there on a right path traversal is executed, through the path `REPLY_OF`. A minimum of zero paths are traversed. Towards the post node there is an incoming path traversal from the forum node, and traverses through the path named `CONTAINER_OF`. This path is a LUR.Another path traversal is present and happens from the forum node towards the person node through the path named `HAS_MODERATOR`

```
SELECT
MATCH (p:Person {id: $authorPersonId}), (country:Country {id: $countryId}), (forum:Forum {id: $forumId})
CREATE (author)<-[:HAS_CREATOR]-(pm:Post:Message {
    id: $postId,creationDate: $creationDate,locationIP: $locationIP, browserUsed: $browserUsed,content: CASE $content WHEN '' T
    length: $length})<-[:CONTAINER_OF]-(forum), (p)-[:IS_LOCATED_IN]->(country)
RETURN $tagIds AS tagId
```

108: ToDo

```
SELECT ...
FROM ...
     MATCH (t:Tag {id: tagId})
CREATE (p)-[:HAS_TAG]->(t)
```

109: ToDo

9.3.20 Interactive Insert 7

In the MATCH clause above, the country, person, and message node are selected. After selection, a left traversal is done from the comment node towards the author node along the path of `HAS_CREATOR`.From the comment node, there is a right traversal towards the message node through the path named `REPLY_OF`. In addition, there is a second MATCH clause that does a traversal from the country node towards the country node along the path `IS_LOCATED_IN`.

In the query above there is no from clause needed since it latches on the results of the previous query. The MATCH clause selects the tag's returned in the query and creates a path from the country node towards the the tag node.

```
FROM LDBC_SNB
MATCH
  (p1:Person {id:\$authorPersonId}),
  (c:Country {id: \$countryId}),
  (m:Message {id: \$replyToPostId + \$replyToCommentId + 1}) // \$replyToCommentId is -1 if the message is a reply to a post an

CREATE (author)<-[:HAS_CREATOR]-(c:Comment:Message {
    id: \$commentId,
    creationDate: \$creationDate,
    locationIP: $locationIP,
    browserUsed: $browserUsed,
    content: \$content,
    length: \$length
  })-[:REPLY_OF]->(message),
  (c)-[:IS_LOCATED_IN]->(country)
Return comment
      ,\$tagIds AS tagId
```

110:

```
  FROM ...
  MATCH (t:Tag {id: tagId})
  CREATE (c)-[:HAS_TAG]->(t)
```

```
SELECT c,$tagIds AS tagId
FROM LDBC_SNB
     MATCH (author:Person {id: $authorPersonId}),(country:Country {id: $countryId}),
           (message:Message {id: $replyToPostId + $replyToCommentId + 1})
       //$replyToCommentId is -1 if the message is a reply to a post and vica versa (see spec)
     CREATE (author)<-[:HAS_CREATOR]-(c:Comment:Message {id: $commentId,creationDate: $creationDate,locationIP: $locationIP,bro
           ,content: $content,length: $length})
           -[:REPLY_OF]->(message),
      (c)-[:IS_LOCATED_IN]->(country)
```

```
SELECT ...
FROM .... //grab the tagID from the results retreived from above
    MATCH (t:Tag {id: tagId})
    CREATE (c)-[:HAS_TAG]->(t)
```

111: ToDo

9.3.21 Interactive Insert 8

```
MATCH (p1:Person {id: \$person1Id}), (p2:Person {id: \$person2Id})
CREATE (p1)-[:KNOWS {creationDate: \$creationDate}]->(p2)
```

In the match clause the person node is selected,and a relationship is created between the selected nodes and the label :KNOWS is given.

```
SELECT ...
FROM   ...
      MATCH (p1:Person {id: $person1Id}), (p2:Person {id: $person2Id})
CREATE (p1)-[:KNOWS {creationDate: $creationDate}]->(p2)
```

112: ToDo