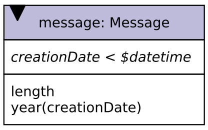
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 1

query	BI / read / 1
title	Posting summary
pattern	

desc.

Given a datetime, find all Messages created before that moment. Group them by a 3-level grouping:

1. by year of creation
2. for each year, group into Message types: is Comment or not
3. for each year-type group, split into four groups based on length of their content
   - `0`: $0 \leq$ length < 40 (short)
   - `1`: $40 \leq$ length < 80 (one liner)
   - `2`: $80 \leq$ length < 160 (tweet)
   - `3`: $160 \leq$ length (long)

params

1	`datetime`	DateTime	For later microbatches, later `datetime` parameters are selected keep the variance low (<0.5%)

result

1	`year`	32-bit Integer	R	`year(message.creationDate)`
2	`isComment`	Boolean	M	`True` for Comments, `False` for Posts
3	`lengthCategory`	32-bit Integer	C	`0` for short, `1` for one-liner, `2` for tweet, `3` for long
4	`messageCount`	32-bit Integer	A	Total number of Messages in that group
5	`averageMessageLength`	32-bit Float	A	Average length of the Message content in that group
6	`sumMessageLength`	32-bit Integer	A	Sum of all Message content lengths
7	`percentageOfMessages`	32-bit Float	A	Number of Messages in group as a percentage of all messages created before the given date

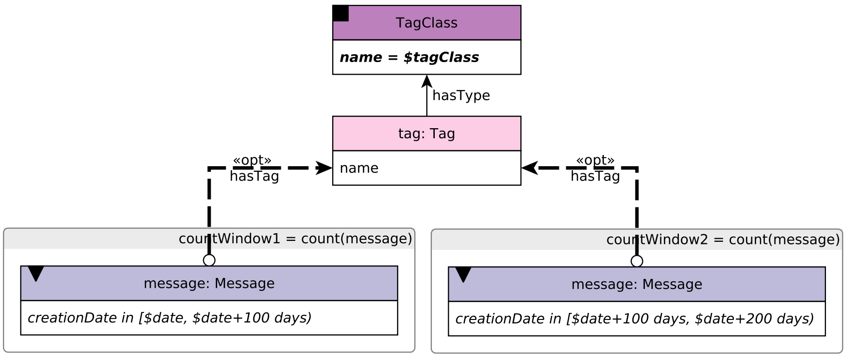
sort

1	`year`	↓	
2	`isComment`	↑	`False` < `True`, i.e. Posts come first and Comments second
3	`lengthCategory`	↑	

limit	n/a
CPs	1.2, 3.2, 4.1, 4.2, 8.5

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 2

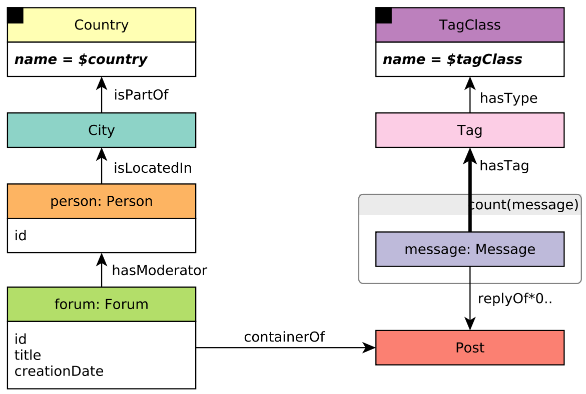
<table>
<tr><td>query</td><td>BI / read / 2</td></tr>
<tr><td>title</td><td>Tag evolution</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Find the Tags under a given TagClass that were used in Messages during in the 100-day time window starting at date and compare it with the 100-day time window that follows. For the Tags and for both time windows, compute the count of Messages.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>date</td><td>Date</td><td>Based on the creation day – TagClass – number of Messages factor table:
(A) A flashmob date
(B) A non-flashmob date</td></tr>
<tr><td>2</td><td>tagClass</td><td>Long String</td><td>For both (A) and (B), TagClasses with a similar amount of Messages are selected</td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>tag.name</td><td>Long String</td><td>R</td><td></td></tr>
<tr><td>2</td><td>countWindow1</td><td>32-bit Integer</td><td>A</td><td>Occurrences of the tag during the first time window</td></tr>
<tr><td>3</td><td>countWindow2</td><td>32-bit Integer</td><td>A</td><td>Occurrences of the tag during the second time window</td></tr>
<tr><td>4</td><td>diff</td><td>32-bit Integer</td><td>A</td><td>Absolute difference of countWindow1 and countWindow2</td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>diff</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>tag.name</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>100</td></tr>
<tr><td>CPs</td><td>2.4, 3.1, 3.2, 4.1, 4.2, 4.3, 5.3, 6.1, 8.2, 8.5</td></tr>
</table>

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 3

query	BI / read / 3
title	Popular topics in a country

pattern



desc.

Given a TagClass and a Country, find all the Forums created in the given Country, containing at least one Message with Tags belonging directly to the given TagClass, and count the Messages by the Forum which contains them.
The location of a Forum is identified by the location of the Forum's moderator.

params

1	`tagClass`	Long String	TagClasses with a similar amount of Messages are selected
2	`country`	Long String	Big Countries are selected

result

1	`forum.id`	ID	R	
2	`forum.title`	Long String	R	
3	`forum.creationDate`	DateTime	R	
4	`person.id`	ID	R	
5	`messageCount`	32-bit Integer	A	

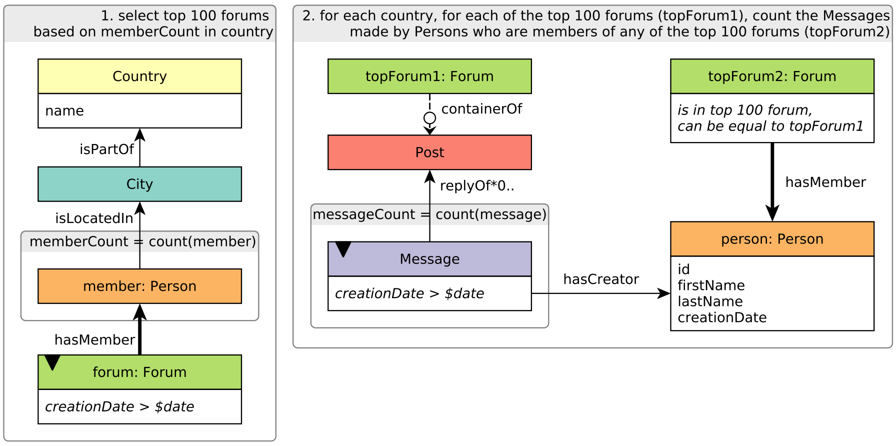
sort

1	`messageCount`	↓	
2	`forum.id`	↑	

limit	20
CPs	1.1, 1.2, 1.3, 2.1, 2.2, 2.4, 3.3, 8.2

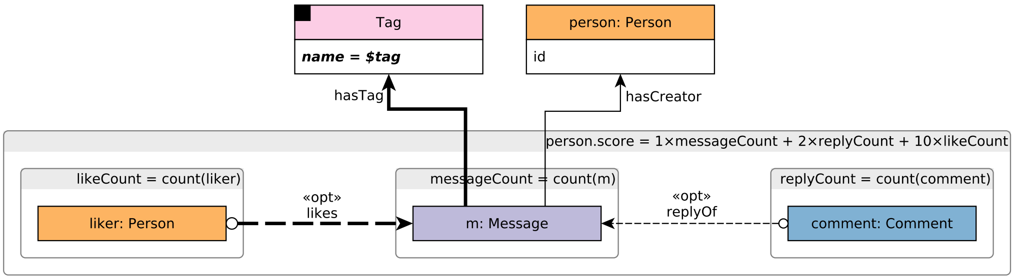
BI / read / 4

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

query	BI / read / 4
title	Top message creators by country
pattern	
desc.	Find the most popular Forums by Country, where the popularity of a Forum is measured by the number of members that Forum has from a given Country. Calculate the top 100 most popular Forums. If a Forum is popular in multiple countries, it should only be calculated once with its largest membership. In case of a tie, the Forum(s) with the smaller id value(s) should be selected. For each member Person of the 100 most popular Forums, count the number of Messages (messageCount) they made in any of those (most popular) Forums. Also include those member Persons who have not posted any Messages (have a messageCount of 0).
params	1 | date | Date | Selected from the first 30 days of the network
result	1 | person.id | ID | R 2 | person.firstName | String | R 3 | person.lastName | String | R 4 | person.creationDate | DateTime | R 5 | messageCount | 32-bit Integer | A
sort	1 | messageCount | ↓ 2 | person.id | ↑
limit	100
CPs	1.2, 1.3, 2.1, 2.2, 2.3, 2.4, 3.3, 5.3, 6.1, 8.2, 8.4

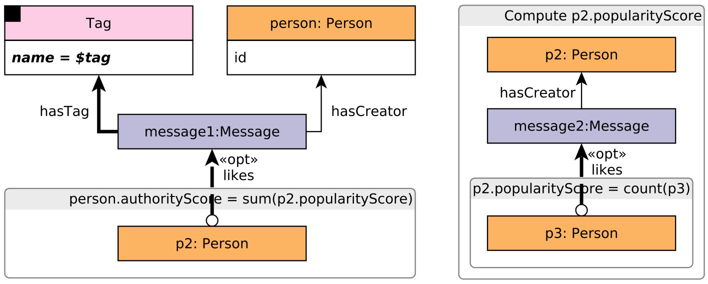
BI / read / 5

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

query	BI / read / 5
title	Most active posters of a given topic
pattern	
desc.	Get each Person (`person`) who has created a Message (`message`) with a given Tag (direct relation, not transitive). Considering only these Messages, for each Person node: • Count its Messages (`messageCount`). • Count likes (`likeCount`) to its Messages. • Count Comments (`replyCount`) in reply to it Messages. The `score` is calculated according to the following formula: $1 \times$ `messageCount` $+ 2 \times$ `replyCount` $+ 10 \times$ `likeCount`.
params	1 | `tag` | Long String | Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q6 and Q7.
result	1 | `person.id` | ID | R 2 | `replyCount` | 32-bit Integer | A 3 | `likeCount` | 32-bit Integer | A 4 | `messageCount` | 32-bit Integer | A 5 | `score` | 32-bit Integer | A
sort	1 | `score` | ↓ 2 | `person.id` | ↑
limit	100
CPs	1.2, 2.3, 8.2

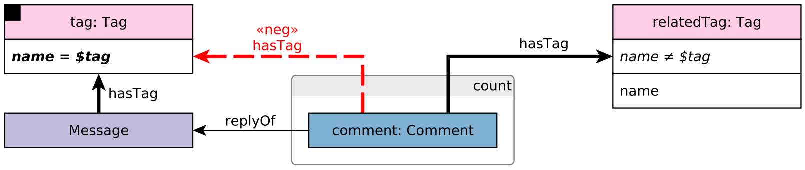
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 6

<table>
<tr><td>query</td><td>BI / read / 6</td></tr>
<tr><td>title</td><td>Most authoritative users on a given topic</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Given a Tag (tag), find all Persons (person) that ever created a Message with the Tag. For each of these Persons (person) compute their “authority score” as follows:
• The “authority score” is the sum of “popularity scores” of the Persons (p2) that liked any of that Person’s Messages with the given Tag (same criterion as for message1).
• A Person’s (p2) “popularity score” is defined as the total number of likes on all of their Messages (message2).</td></tr>
<tr><td>params</td><td><table><tr><td>1</td><td>tag</td><td>Long String</td><td>Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q5 and Q7.</td></tr></table></td></tr>
<tr><td>result</td><td><table><tr><td>1</td><td>person.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>2</td><td>authorityScore</td><td>32-bit Integer</td><td>A</td><td></td></tr></table></td></tr>
<tr><td>sort</td><td><table><tr><td>1</td><td>authorityScore</td><td>↓</td><td></td></tr><tr><td>2</td><td>person1.id</td><td>↑</td><td></td></tr></table></td></tr>
<tr><td>limit</td><td>100</td></tr>
<tr><td>CPs</td><td>1.2, 2.3, 3.3, 6.1, 8.2</td></tr>
<tr><td>relevance</td><td>Computing the authority scores might involve computing the popularity score for the same Person multiple times. Implementations are advised to avoid such redundant computations.</td></tr>
</table>

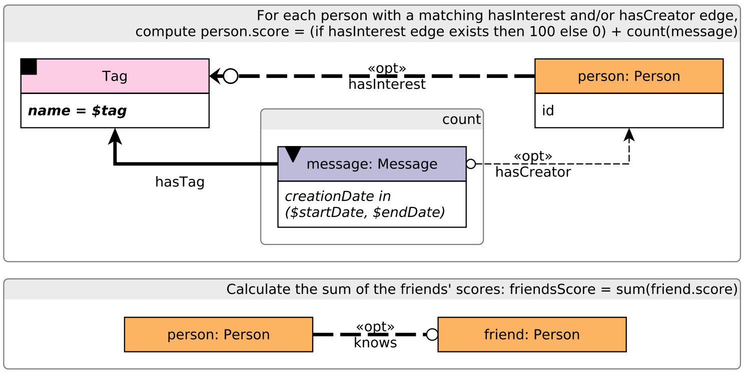
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 7

query	BI / read / 7
title	Related topics
pattern	
desc.	Find all Messages that have a given Tag. Find the related Tags attached to (direct) reply Comments of these Messages, but only of those reply Comments that do not have the given Tag. Group the Tags by name, and get the count of replies in each group.
params	1 | tag | Long String | Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q5 and Q6.
result	1 | relatedTag.name | Long String | R 2 | count | 32-bit Integer | A
sort	1 | count | ↓ 2 | relatedTag.name | ↑
limit	100
CPs	1.4, 3.3, 5.2, 8.1

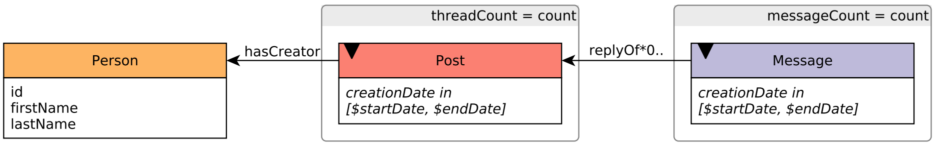
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 8

<table>
<tr><td>query</td><td>BI / read / 8</td></tr>
<tr><td>title</td><td>Central person for a tag</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Given a Tag, find all Persons that are interested in the Tag and/or have written a Message (Post or Comment) with a creationDate after a given date and that has a given Tag. For each Person, compute the score as the sum of the following two aspects:
100, if the Person has this Tag as their interest, or 0 otherwise
number of Messages by this Person with the given Tag
Also, for each Person, compute the sum of the score of the Person's friends (friendsScore).</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>tag</td><td>Long String</td><td>Tags with a similar amount of Messages are selected</td></tr>
<tr><td>2</td><td>startDate</td><td>Date</td><td>(A): A range during which a flashmob event happened (it should yield at least a 5× difference)
(B): A regular range (does not include a flashmob event)</td></tr>
<tr><td>3</td><td>endDate</td><td>Date</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>person.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>score</td><td>32-bit Integer</td><td>A</td><td></td></tr>
<tr><td>3</td><td>friendsScore</td><td>32-bit Integer</td><td>A</td><td>The sum of the score of the person's friends</td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>score + friendsScore</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>person.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>100</td></tr>
<tr><td>CPs</td><td>1.2, 2.1, 2.3, 3.2, 5.3, 8.2, 8.4, 8.5</td></tr>
<tr><td>relevance</td><td>Similarly to BI 16, there are two major ways to compute this query: (1) creating an induced subgraph of the interested Persons and their friends and performing the scoring on this graph or (2) performing the scoring without creating an induced subgraph and scoring the friends of a Person on-the-fly. The first approach is more efficient as it avoids redundant computations, however, specifying it needs support for composable graph queries.</td></tr>
</table>

BI / read / 9

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

query	BI / read / 9
title	Top thread initiators
pattern	

desc.	For each Person, count the number of Posts they created in the time interval `[startDate, endDate]` (equivalent to the number of threads they initiated) and the number of Messages in each of their (transitive) reply trees, including the root Post of each tree. When calculating Message counts only consider Messages created within the given time interval. Return each Person, number of Posts they created, and the count of all Messages that appeared in the reply trees (including the Post at the root of tree).

params

1	`startDate`	Date	Selected around the same date
2	`endDate`	Date	80-100 days after the `startDate`

result

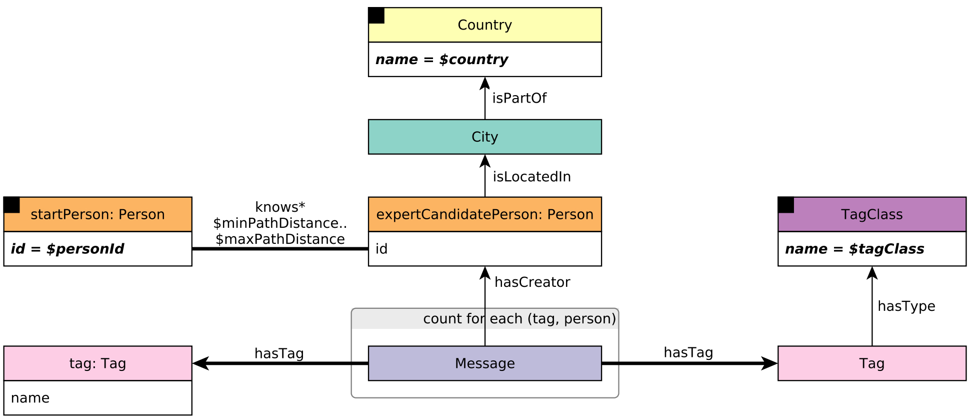
1	`person.id`	ID	R	
2	`person.firstName`	String	R	
3	`person.lastName`	String	R	
4	`threadCount`	32-bit Integer	A	The number of Posts created by that Person (the number of threads initiated)
5	`messageCount`	32-bit Integer	A	The number of Messages created in all the threads this Person initiated

sort

1	`messageCount`	↓	
2	`person.id`	↑	

limit	100
CPs	1.2, 2.2, 2.3, 3.2, 7.2, 7.3, 7.4, 8.1, 8.5

BI / read / 10

query	BI / read / 10
title	Experts in social circle
pattern	
desc.	Given a Person (`startPerson`), find all other Persons (`expertCandidatePerson`) that live in a given Country and are connected to given Person by a *shortest path* with length in range `[minPathDistance, maxPathDistance]` through the knows relation. For each of these `expertCandidatePerson` nodes, retrieve all of their Messages that contain at least one Tag belonging to a given TagClass (direct relation not transitive). For each Message, retrieve all of its Tags. Group the results by Persons and Tags, then count the Messages by a certain Person having a certain Tag.

params

1	`personId`	ID	(A) Persons with an average degree of knows edges are selected (B) Persons who have only one friend and that Person has two friends in total (including the original Person)
2	`country`	String	Select mid-sized Countries
3	`tagClass`	Long String	TagClasses with a similar degree of hasType edges are selected
4	`minPathDistance`	32-bit Integer	3
5	`maxPathDistance`	32-bit Integer	4

result

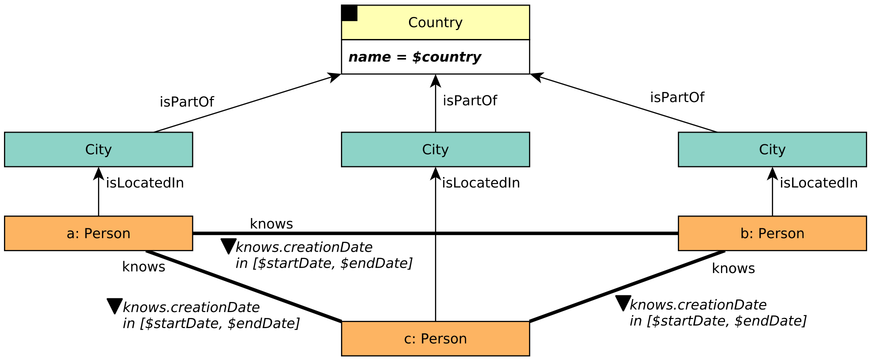
1	`expertCandidatePerson.id`	ID	R	
2	`tag.name`	Long String	R	
3	`messageCount`	32-bit Integer	A	Number of Messages created by that Person containing that Tag

sort

1	`messageCount`	↓	
2	`tag.name`	↑	
3	`expertCandidatePerson.id`	↑	

limit	100
CPs	1.2, 1.3, 2.3, 2.4, 3.3, 5.3, 7.1, 7.2, 7.3, 8.1, 8.6

BI / read / 11

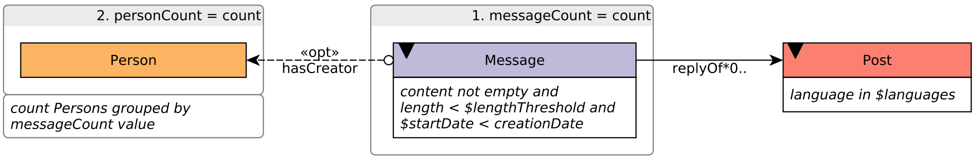
<table>
<tr><td>query</td><td>BI / read / 11</td></tr>
<tr><td>title</td><td>Friend triangles</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>For a given <code>country</code>, count all the distinct triples of Persons such that:

<code>a</code> is friend of <code>b</code>,
<code>b</code> is friend of <code>c</code>,
<code>c</code> is friend of <code>a</code>,

and these friendships were created in the range [<code>startDate</code>, <code>endDate</code>].
Distinct means that given a triple t_1 in the result set R of all qualified triples, there is no triple t_2 in R such that t_1 and t_2 have the same set of elements.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td><code>country</code></td><td>Long String</td><td>Selected from the largest Countries (India, China)</td></tr>
<tr><td>2</td><td><code>startDate</code></td><td>Date</td><td>Selected from a 30-day interval towards the end of the simulation time</td></tr>
<tr><td>3</td><td><code>endDate</code></td><td>Date</td><td>Selected to yield around a 100-day interval</td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td><code>count</code></td><td>64-bit Integer</td><td>A</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>n/a</td></tr>
<tr><td>CPs</td><td>1.1, 2.3, 2.5</td></tr>
</table>

BI / read / 12

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

query	BI / read / 12
title	How many persons have a given number of messages
pattern	

desc.

For each Person, count the number of Messages they made (messageCount). Only count Messages with the following attributes:

- Its content is not empty (and consequently, the imageFile attribute is empty for Posts).
- Its length is below the lengthThreshold (exclusive, equality is not allowed).
- Its creationDate is after startDate (exclusive, equality is not allowed).
- It is written in any of the given languages.
  - The language of a Post is defined by its language attribute.
  - The language of a Comment is that of the Post that initiates the thread where the Comment replies to.

  The Post and Comments in the reply tree's path (from the Message to the Post) do not have to satisfy the constraints for content, length, and creationDate.

For each messageCount value, count the number of Persons with exactly messageCount Messages (with the required attributes).

params

1	startDate	Date	Selected randomly from a 60-day interval.
2	lengthThreshold	32-bit Integer	Balanced against startDate to filter around 30% of the Messages within a language and keep the variance low. The selection of this parameter uses a factor table of bucketed Message lengths and creation dates.
3	languages	{String}	Only the most frequently used languages

result

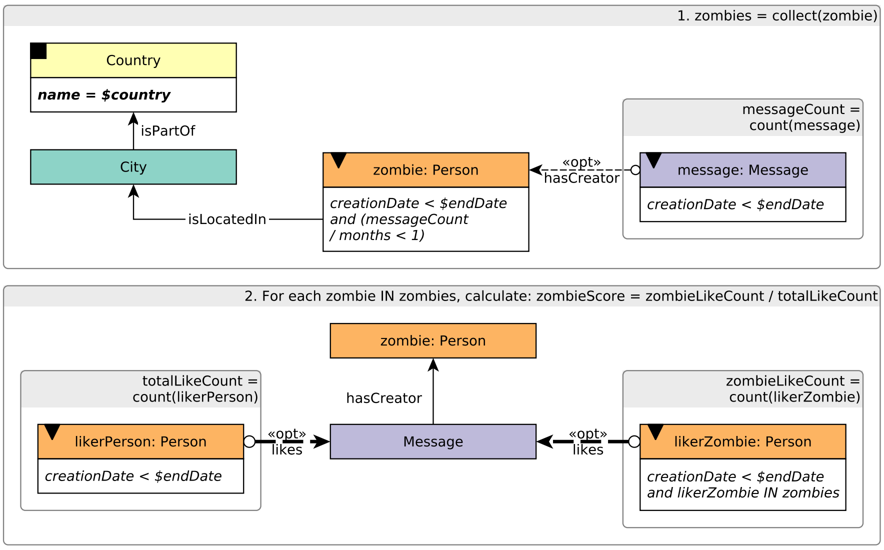
1	messageCount	32-bit Integer	A	Number of Messages created
2	personCount	32-bit Integer	A	Number of Persons with messageCount Messages

sort

1	personCount	↓	
2	messageCount	↓	

limit	n/a
CPs	1.1, 1.2, 1.4, 3.2, 4.2, 4.3, 8.1, 8.2, 8.3, 8.4, 8.5

BI / read / 13

<table>
<tr><td>query</td><td>BI / read / 13</td></tr>
<tr><td>title</td><td>Zombies in a country</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Find zombies within the given <code>country</code>, and return their zombie scores. A <code>zombie</code> is a Person created before the given <code>endDate</code>, which has created an average of <code>[0, 1)</code> Messages per month, during the time range between profile's <code>creationDate</code> and the given <code>endDate</code>. The number of months spans the time range from the <code>creationDate</code> of the profile to the <code>endDate</code> with partial months on both end counting as one month (e.g. a <code>creationDate</code> of Jan 31 and an <code>endDate</code> of Mar 1 result in 3 months).
For each <code>zombie</code>, calculate the following:

<code>zombieLikeCount</code>: the number of likes received from other zombies.
<code>totalLikeCount</code>: the total number of likes received.
<code>zombieScore</code>: <code>zombieLikeCount</code> / <code>totalLikeCount</code>. If the value of <code>totalLikeCount</code> is 0, the <code>zombieScore</code> of the <code>zombie</code> should be 0.0.

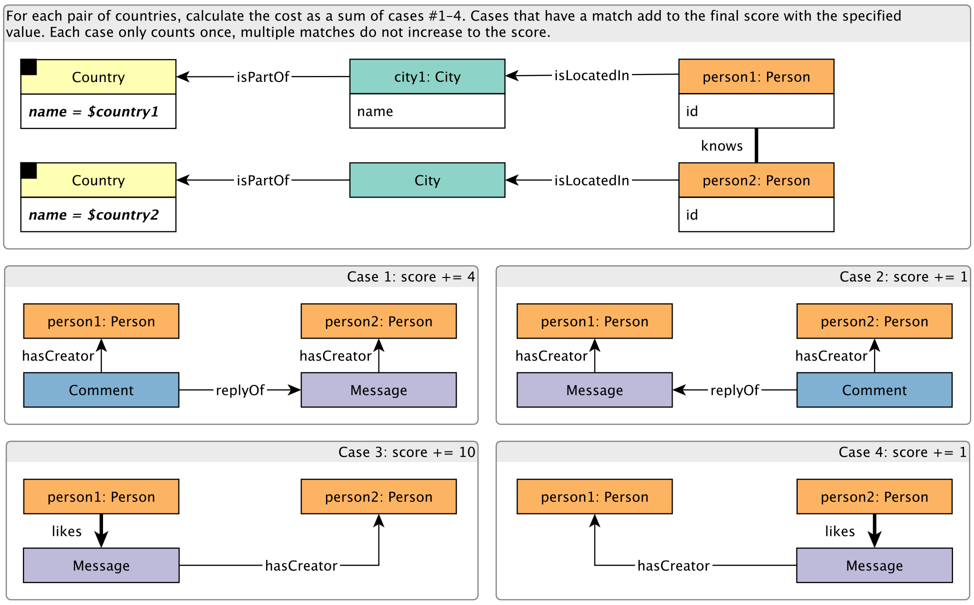
For both <code>zombieLikeCount</code> and <code>totalLikeCount</code>, only consider likes received from profiles that were created before the given <code>endDate</code>.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>country</td><td>Long String</td><td>Selected from the largest Countries (India, China)</td></tr>
<tr><td>2</td><td>endDate</td><td>Date</td><td>Selected from the last days of the initial data set</td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>zombie.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>zombieLikeCount</td><td>32-bit Integer</td><td>A</td><td></td></tr>
<tr><td>3</td><td>totalLikeCount</td><td>32-bit Integer</td><td>A</td><td></td></tr>
<tr><td>4</td><td>zombieScore</td><td>32-bit Float</td><td>A</td><td>Determined as <code>zombieLikeCount</code> / <code>totalLikeCount</code></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>zombieScore</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>zombie.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>100</td></tr>
<tr><td>CPs</td><td>1.2, 2.1, 2.3, 2.4, 3.2, 3.3, 4.2, 5.1, 5.3, 8.2, 8.4, 8.5</td></tr>
</table>

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 14

query	BI / read / 14
title	International dialog

pattern



desc.

Consider all pairs of people (`person1`, `person2`) such that (1) they know each ther, (2) one is located in a City of Country `country1`, and (3) the other is located in a City of Country `country2`. For each City of Country `country1`, return the highest scoring pair. The score of a pair is defined as the sum of the subscores awarded for the following kinds of interaction. The initial value is `score = 0`.

1. `person1` has created a reply Comment to at least one Message by `person2`: `score += 4`
2. `person1` has created at least one Message that `person2` has created a reply to: `score += 1`
3. `person1` liked at least one Message by `person2`: `score += 10`
4. `person1` has created at least one Message that was liked by `person2`: `score += 1`

Consequently, the maximum score a pair can obtain is: `4 + 1 + 10 + 1 = 16`.

params

1	country1	Long String	(A) Correlated with parameter `country2`, i.e. the Countries are close and there are many Persons knowing each other (B) Uncorrelated with parameter `country2`, i.e. the Countries are afar and there are few Persons knowing each other
2	country2	Long String	

result

1	person1.id	ID	R	
2	person2.id	ID	R	
3	city1.name	Long String	R	
4	score	32-bit Integer	C	

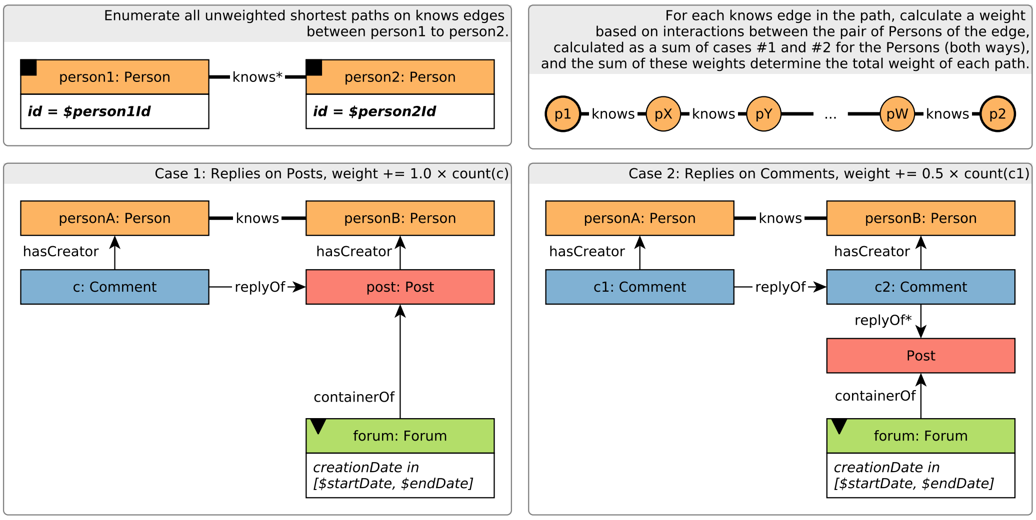
sort

1	score	↓	
2	person1.id	↑	
3	person2.id	↑	

limit	n/a
CPs	1.3, 1.4, 2.1, 3.1, 3.3, 5.1, 5.2, 5.3, 8.3, 8.4

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 15

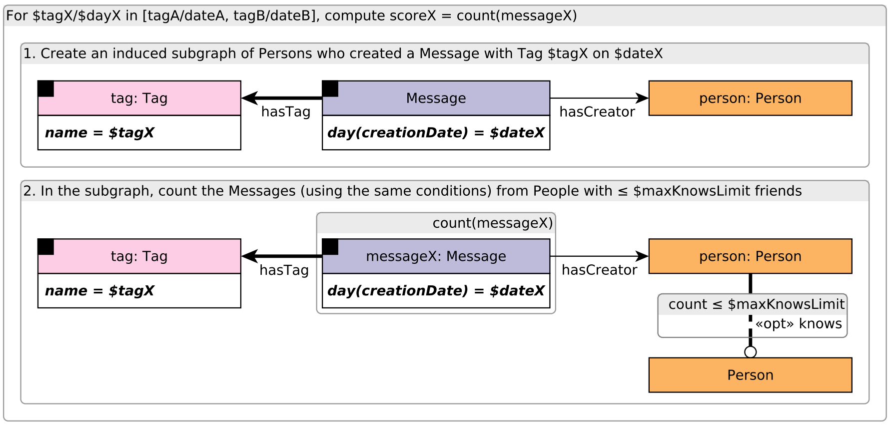
<table>
<tr><td>query</td><td>BI / read / 15</td></tr>
<tr><td>title</td><td>Trusted connection paths through forums created in a given timeframe</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Given two Persons, find all (unweighted) shortest paths between these two Persons, in the subgraph induced by the knows relationship.
Then, for each path calculate a weight. The nodes in the path are Persons, and the weight of a path is the sum of weights between every pair of consecutive Person nodes in the path.
The weight for a pair of Persons is calculated based on their interactions:
• Every direct reply (by one of the Persons) to a Post (by the other Person) contributes 1.0.
• Every direct reply (by one of the Persons) to a Comment (by the other Person) contributes 0.5.
Only consider Messages that were created in a Forum that was created within the timeframe (interval) [startDate, endDate]. Note that for Comments, the containing Forum is that of the Post that the comment (transitively) replies to. Also note that interactions are counted both ways.
Return all paths with the Person IDs ordered by their weights descending.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>person1Id</td><td>ID</td><td>(A) person1Id – person2Id pair with a distance of exactly 4 hops
(B) person1Id – person2Id pair with a distance of exactly 2 hops</td></tr>
<tr><td>2</td><td>person2Id</td><td>ID</td><td></td></tr>
<tr><td>3</td><td>startDate</td><td>Date</td><td>(A) Small interval (approx. one week)
(B) Big interval (approx. one month)</td></tr>
<tr><td>4</td><td>endDate</td><td>Date</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>person.id</td><td>[ID]</td><td>C</td><td>Ordered sequence of the Person IDs in the path</td></tr>
<tr><td>2</td><td>weight</td><td>32-bit Float</td><td>C</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>weight</td><td>↓</td><td>The order of paths with the same weight is unspecified</td></tr>
<tr><td>2</td><td>personIds</td><td>↑</td><td>The IDs in the paths are used for lexicographical sorting</td></tr>
</table></td></tr>
<tr><td>limit</td><td>n/a</td></tr>
<tr><td>CPs</td><td>1.2, 2.1, 2.2, 2.4, 3.3, 5.1, 5.3, 7.2, 7.3, 7.5, 7.7, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6</td></tr>
</table>

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 16

query	BI / read / 16
title	Fake news detection

pattern



desc.

Given two Tag/date pairs (tagA/dateA and tagB/dateB), for each pair tagX/dateX:

- Create an induced subgraph between Persons where for each pair of Persons person1/person2, both have created a Message on the day of dateX with Tag tagX.
- In the induced subgraph, only keep pairs of Persons who have at most maxKnowsLimit friends (in the induced subgraph).
- For these Persons, count the number of Messages created on dateX with Tag tagX.

Return Persons who had at least one Messages for both tagA/dateA and tagB/dateB ranked by their total number of Messages (descending).

params

1	tagA	Long String	(A) tagA–dateA, tagB–dateB are both selected to be a flashmob Tag – date combination (B) tagA–dateA, tagB–dateB are both selected to be a non-flashmob Tag – date combination
2	dateA	Date	
3	tagB	Long String	
4	dateB	Date	
5	maxKnowsLimit	32-bit Integer	Selected between 3 and 6

result

1	person.id	ID	R	
2	messageCountA	32-bit Integer	A	Message count for tagA/dateA
3	messageCountB	32-bit Integer	A	Message count for tagB/dateB

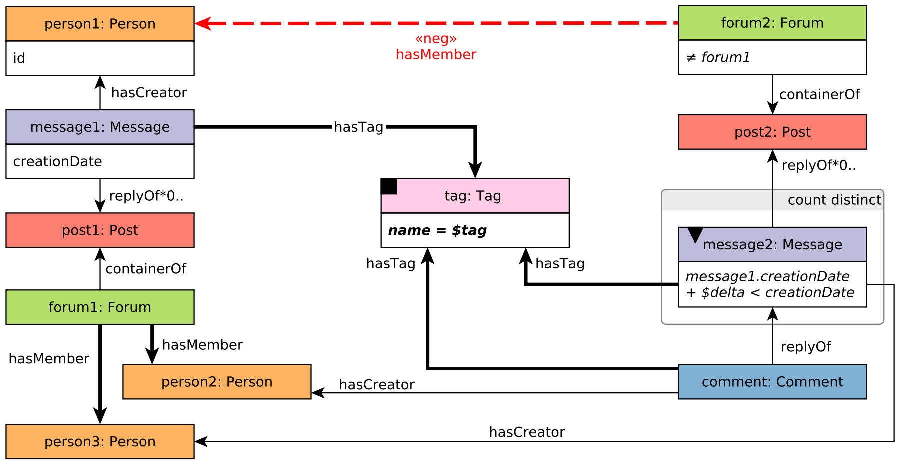
sort

1	messageCountA + messageCountB	↓	
2	person.id	↑	

limit	20
CPs	5.3, 8.4, 8.5
relevance	There are two major ways to compute this query: (1) create the induced subgraph as suggested by the specification (either as a view or in materialized form), or (2) skip creating the induced subgraph and perform on-the-fly check for the number of friends (who also posted at least one Message with the given Tag on the given date). The latter approach is easier to express in systems which do not provide graph views but might result in redundant computations (the query engine might repeatedly check whether a Person has at least one Message that satifies the conditions).

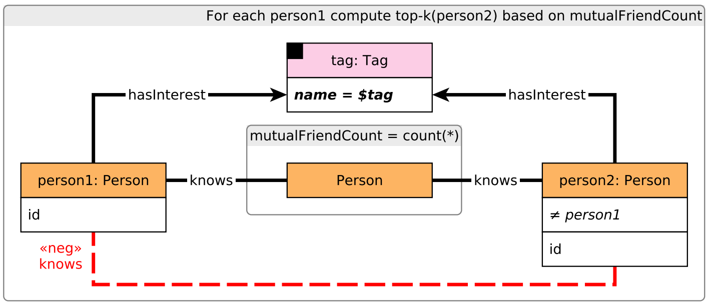
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 17

query	BI / read / 17
title	Information propagation analysis
pattern	
desc.	This query aims to identify instances of "information propagation" when a Person (person1) submits a Message (message1) with a given Tag (tag) to a Forum (forum1). This is read by other members of forum1, Persons person2 and person3. Some time later (specified by the delta parameter), these persons have a discussion with the same tag in a different Forum (forum2) where person1 is not a member. The discussion consists of a Message (message2) by person2 and a direct reply Comment (comment) by person3. Return IDs of person1 with the number of interactions their Messages (might have) caused.
params	1 | tag | Long String | Tags with a similar amount of Messages are selected 2 | delta | 32-bit Integer | Measured in hours, selected to be between 8 and 16 hours.
result	1 | person1.id | ID | R 2 | messageCount | 32-bit Integer | A
sort	1 | messageCount | ↓ 2 | person1.id | ↑
limit	10
CPs	2.1, 2.3, 8.1

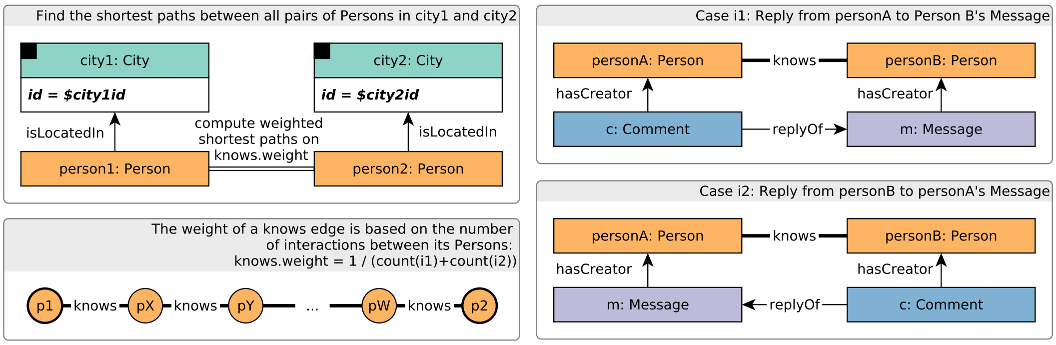
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 18

<table>
<tr><td>query</td><td>BI / read / 18</td></tr>
<tr><td>title</td><td>Friend recommendation</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>For a given Tag (tag), for each person1 interested in tag, recommend new friends (person2) who
do not yet know person1
at least one mutual friend with person1
are also interested in tag.
Rank Persons person2 based on the number of mutual friends with person1.</td></tr>
<tr><td>params</td><td><table><tr><td>1</td><td>tag</td><td>Long String</td><td>Tags with a similar amount of Persons are selected</td></tr></table></td></tr>
<tr><td>result</td><td><table><tr><td>1</td><td>person1.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>2</td><td>person2.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>3</td><td>mutualFriendCount</td><td>32-bit Integer</td><td>A</td><td></td></tr></table></td></tr>
<tr><td>sort</td><td><table><tr><td>1</td><td>mutualFriendCount</td><td>↓</td><td></td></tr><tr><td>2</td><td>person1.id</td><td>↑</td><td></td></tr><tr><td>3</td><td>person2.id</td><td>↑</td><td></td></tr></table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>2.5, 8.1</td></tr>
</table>

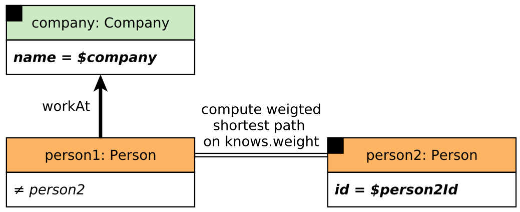
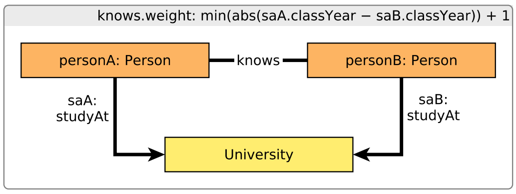
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 19

<table>
<tr><td>query</td><td>BI / read / 19</td></tr>
<tr><td>title</td><td>Interaction path between cities</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>desc.</td><td>Given two Cities city1, city2, find Persons person1, person2 living in these Cities (respectively) with the shortest interaction path between them. If there are multiple pairs of people with shortest paths having the same total weight, return all of them.
The shortest path is computed using a weight between two Persons defined as the reciprocal of the number of interactions (direct reply Comments to a Message by the other Person). Therefore, more interactions imply a smaller weight.
Note: Interactions are counted both ways, i.e. if Alice writes 2 reply Comments to Bob's Messages and Bob writes 3 reply Comments to Alice's Messages, their total number of interactions is 5.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>city1Id</td><td>ID</td><td>(A) Small Cities within the same Country with many direct relationships between their inhabitants
(B) Small Cities from different Countries with only a few direct relationships between their inhabitants</td></tr>
<tr><td>2</td><td>city2Id</td><td>ID</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>person1.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>person2.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>3</td><td>totalWeight</td><td>32-bit Float</td><td>C</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>totalWeight</td><td>↑</td><td></td></tr>
<tr><td>2</td><td>person1.id</td><td>↑</td><td></td></tr>
<tr><td>3</td><td>person2.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>3.3, 7.6, 7.7, 8.4, 8.6</td></tr>
<tr><td>relevance</td><td>Finding shortest paths between pairs of Persons in Cities can be implemented in theory with an all-pairs shortest paths algorithm. However, this needs to be executed on the whole Person-knows-Person graph (with edge weights derived from the number of interactions) so it is expected to be prohibitively expensive. A better approach is using multiple single-source shortest path algorithms (e.g. from the City with fewer inhibitants).
Implementations can either pre-compute edge weights or compute them on-the-fly.</td></tr>
</table>

BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 20

query	BI / read / 20
title	Recruitment
pattern	 
desc.	Given a Company `company` and a Person `person2` (who is not working and has not worked at `company`), find a different Person (`person1`) who works or at some point worked in `company` and is reachable by from `person2` through people who have studied together. On this path, we only consider edges between Persons who know each other and attended the same University and set the weight of the edge to the absolute difference between the year of enrolment plus 1 (`studyAt.classYear + 1`). If the Persons attended multiple universities, we select the smallest (`min`) value. If there are multiple Person `person1` nodes with the same shortest path, return all of them.
params	1 | `company` | Long String | Companies with a similar number of employees (former or current) are selected 2 | `person2Id` | ID | `person2` is selected so that there is no direct (1-hop) path to any `person1` working at `company`
result	1 | `person1.id` | ID | R 2 | `totalWeight` | 64-bit Integer | C
sort	1 | `totalWeight` | ↑ 2 | `person1.id` | ↑
limit	20
CPs	3.3, 7.6, 7.7, 8.4, 8.6
relevance	Implementations can either pre-compute edge weights or compute them on-the-fly. To find the (weighted) shortest path efficiently, can use e.g. a bidirectional Dijkstra algorithm.