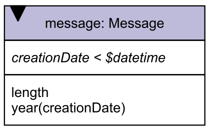
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 1

query	BI / read / 1
title	Posting summary
pattern	 message: Message creationDate < $datetime length year(creationDate)
description	Given a `$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 ≤ length < 40 (short) • `1`: 40 ≤ length < 80 (one liner) • `2`: 80 ≤ length < 160 (tweet) • `3`: 160 ≤ length (long)

params

1	`$datetime`	DateTime	

result

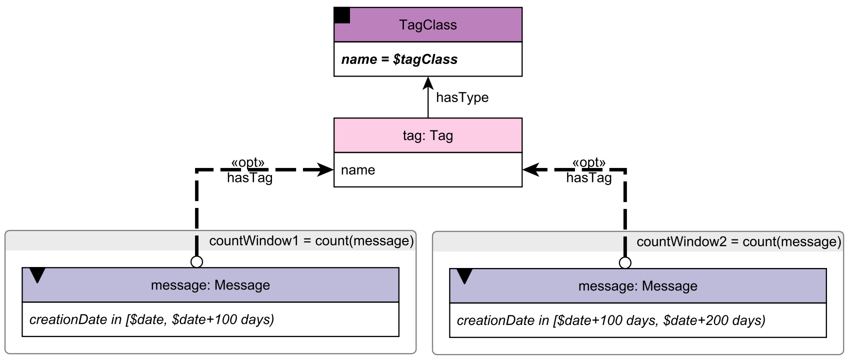
1	`year`	32-bit Integer	R	`year(message.creationDate)`
2	`isComment`	Boolean	M	`True` for Comments, `False` for Posts
3	`lengthCategory`	32-bit Integer	C	`0` for short, `1` for one-liner, `2` for tweet, `3` for long
4	`messageCount`	64-bit Integer	A	Total number of Messages in that group
5	`averageMessageLength`	32-bit Float	A	Average length of the Message content in that group
6	`sumMessageLength`	64-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 / read / 2

query	BI / read / 2
title	Tag evolution
pattern	 TagClass name = $tagClass hasType tag: Tag name «opt» hasTag «opt» hasTag countWindow1 = count(message) message: Message creationDate in [$date, $date+100 days) countWindow2 = count(message) message: Message creationDate in [$date+100 days, $date+200 days)
description	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.

params				
	1	$date	Date	Based on the creation day – TagClass – number of Messages factor table: (a) A flashmob date (b) A non-flashmob date
	2	$tagClass	Long String	For both (a) and (b), TagClasses with a similar amount of Messages are selected

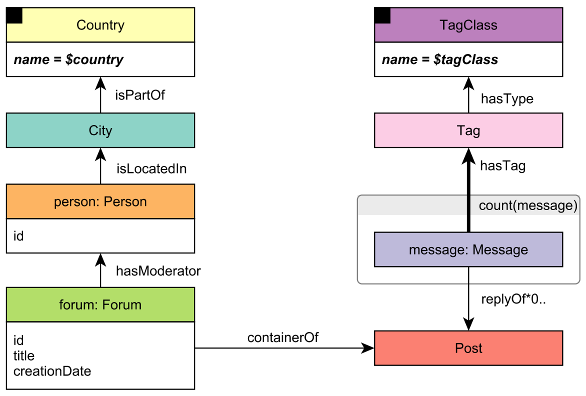
result					
	1	tag.name	Long String	R	
	2	countWindow1	32-bit Integer	A	Occurrences of the tag during the first time window
	3	countWindow2	32-bit Integer	A	Occurrences of the tag during the second time window
	4	diff	32-bit Integer	A	Absolute difference of countWindow1 and countWindow2

sort				
	1	diff	↓	
	2	tag.name	↑	

limit	100
CPs	2.4, 3.1, 3.2, 4.1, 4.2, 4.3, 5.3, 6.1, 8.2, 8.5

BI / read / 3

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 / 3
title	Popular topics in a country
pattern	
description	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

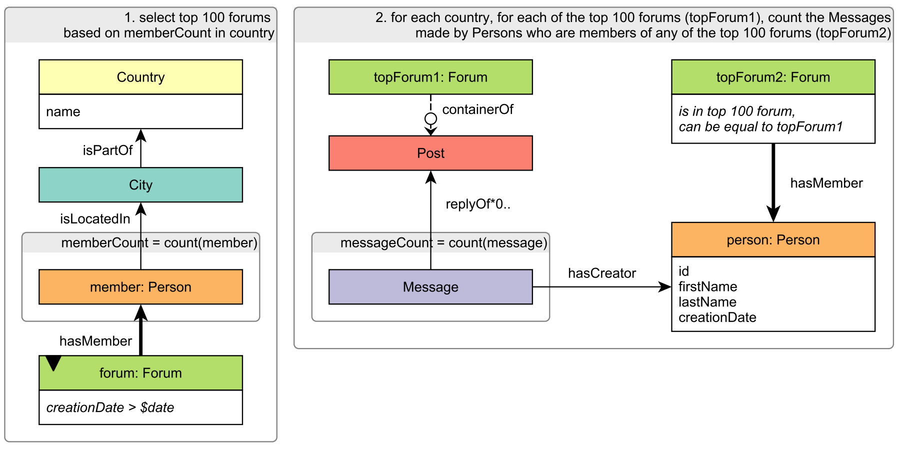
result			
1	`forum.id`	ID	R
2	`forum.title`	Long String	R
3	`forum.creationDate`	DateTime	R
4	`person.id`	ID	R
5	`messageCount`	32-bit Integer	A

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 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 / 4

query	BI / read / 4
title	Top message creators by country
pattern	
description	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 and the Forum was created after a given `$date`. 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 with the smaller id value 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).

1. select top 100 forums based on memberCount in country
Country
name
isPartOf
City
isLocatedIn
memberCount = count(member)
member: Person
hasMember
forum: Forum
creationDate > $date
2. for each country, for each of the top 100 forums (topForum1), count the Messages made by Persons who are members of any of the top 100 forums (topForum2)
topForum1: Forum
containerOf
Post
replyOf*0..
messageCount = count(message)
Message
hasCreator
topForum2: Forum
is in top 100 forum, can be equal to topForum1
hasMember
person: Person
id
firstName
lastName
creationDate

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	

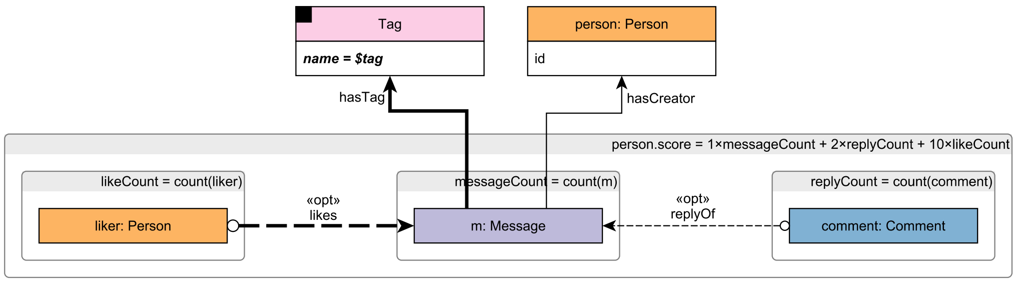
sort

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

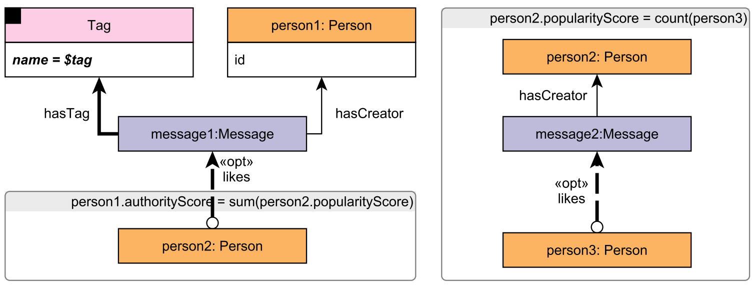
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

BI / read / 5

query	BI / read / 5
title	Most active posters of a given topic
pattern	
description	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 its 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, 2.6, 8.2

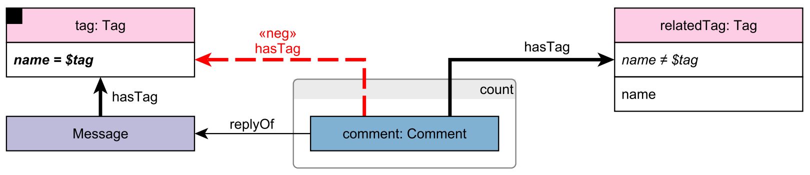
BI / read / 6

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 / 6
title	Most authoritative users on a given topic
pattern	 Tag name = $tag person1: Person id hasTag message1:Message hasCreator «opt» likes person1.authorityScore = sum(person2.popularityScore) person2: Person person2.popularityScore = count(person3) person2: Person hasCreator message2:Message «opt» likes person3: Person
description	Given a `$tag`, find all Persons (`person1`) that ever created a Message with the `$tag`. For each of these Persons (`person1`) compute their "authority score" as follows: • The "authority score" is the sum of "popularity scores" of the Persons (`person2`) that liked any of that Person's Messages with the given `$tag` (same criterion as for `message1`). • A Person's (`person2`) "popularity score" is defined as the total number of likes (by any Person `person3`) on any of their Messages (`message2`).
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 Q7.
result	1 | `person1.id` | ID | R 2 | `authorityScore` | 32-bit Integer | A
sort	1 | `authorityScore` | ↓ 2 | `person1.id` | ↑
limit	100
CPs	1.2, 2.3, 2.6, 3.3, 6.1, 8.2
relevance	Computing the authority scores might involve computing the popularity score for the same Person multiple times. Implementations are advised to avoid such redundant computations.

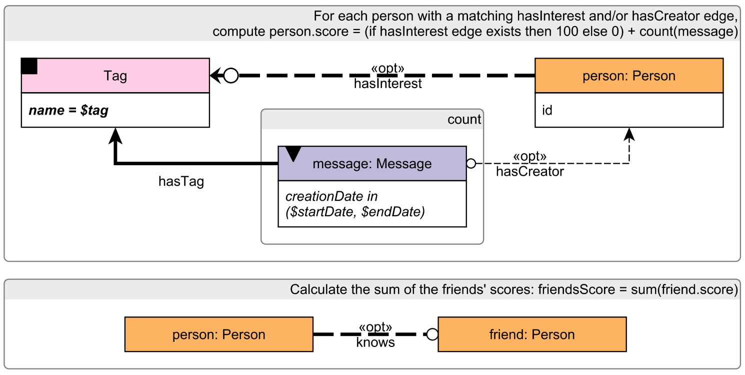
BI / read / 7

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 / 7
title	Related topics
pattern	 tag: Tag name = $tag «neg» hasTag hasTag relatedTag: Tag name ≠ $tag name count hasTag Message replyOf comment: Comment
description	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 related 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 / read / 8

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 / 8
title	Central person for a tag
pattern	
description	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 `$startDate` and that has a given `$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`).

For each person with a matching hasInterest and/or hasCreator edge, compute person.score = (if hasInterest edge exists then 100 else 0) + count(message)
Tag
name = $tag
«opt»
hasInterest
person: Person
id
count
message: Message
creationDate in ($startDate, $endDate)
hasTag
«opt»
hasCreator
Calculate the sum of the friends' scores: friendsScore = sum(friend.score)
person: Person
«opt»
knows
friend: Person

params	1	`$tag`	Long String	Tags with a similar amount of Messages are selected
	2	`$startDate`	Date	`(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)
	3	`$endDate`	Date	

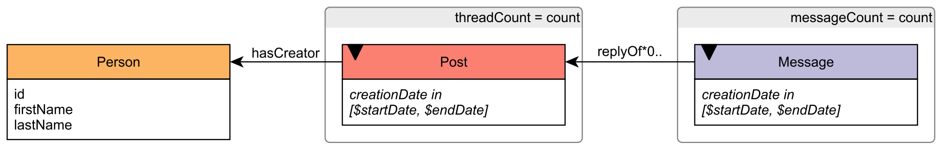
result	1	`person.id`	ID	R	
	2	`score`	32-bit Integer	A	
	3	`friendsScore`	32-bit Integer	A	The sum of the score of the `person`'s friends

sort	1	`score + friendsScore`	↓	
	2	`person.id`	↑	

limit	100
CPs	1.2, 2.1, 2.3, 3.2, 5.3, 8.2, 8.4, 8.5
relevance	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.

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	 threadCount = count messageCount = count Person id firstName lastName hasCreator Post creationDate in [$startDate, $endDate] replyOf*0.. Message creationDate in [$startDate, $endDate]
description	For each Person, count the number of Posts they created in the time interval `[$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`

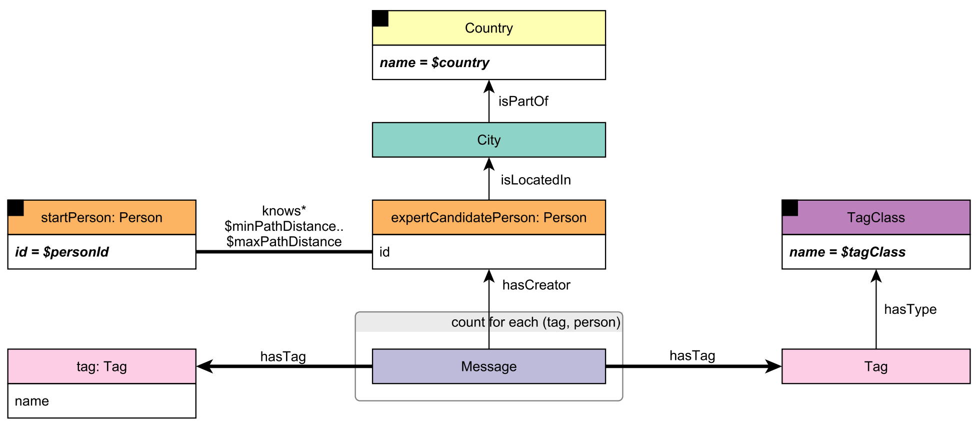
result					
	1	`person.id`	ID	R	
	2	`person.firstName`	String	R	
	3	`person.lastName`	String	R	
	4	`threadCount`	32-bit Integer	A	The number of Posts created by that Person (the number of threads initiated)
	5	`messageCount`	32-bit Integer	A	The number of Messages created in all the threads this Person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

sort				
	1	`messageCount`	↓	
	2	`person.id`	↑	

limit	100
CPs	1.2, 2.2, 2.3, 2.6, 3.2, 7.2, 7.3, 7.4, 8.1, 8.5

BI / read / 10

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 / 10
title	Experts in social circle
pattern	 Country name = $country isPartOf City isLocatedIn startPerson: Person id = $personId knows* $minPathDistance.. $maxPathDistance expertCandidatePerson: Person id TagClass name = $tagClass hasCreator hasType count for each (tag, person) tag: Tag name hasTag Message hasTag Tag
description	Given a Person startPerson with ID $personID, find all other Persons (expertCandidatePerson) that live in a given $country and are connected to the startPerson on a *shortest path* with length in range [$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.

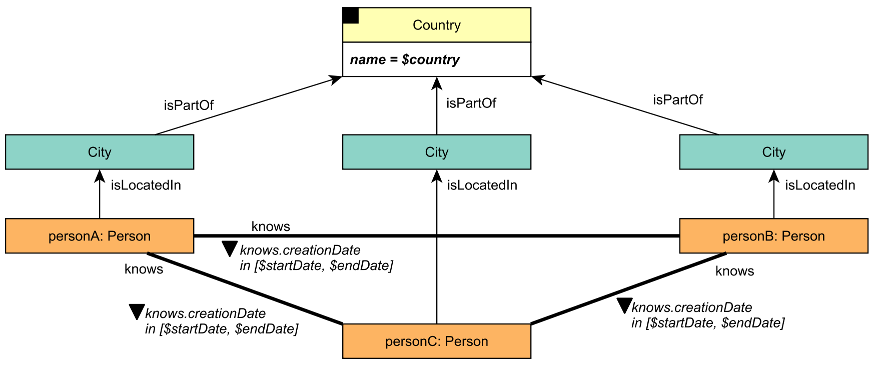
params			
1	$personId	ID	(a) Persons with an average degree of knows edges are selected (b) Persons who have only one friend and that Person has two friends in total (including the original Person)
2	$country	String	Select mid-sized Countries
3	$tagClass	Long String	TagClasses with a similar degree of 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	↑	

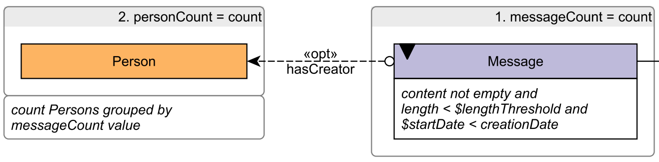
limit	100
CPs	1.2, 1.3, 2.3, 2.4, 2.6, 3.3, 5.3, 7.1, 7.2, 7.3, 8.1, 8.6

BI / read / 11

query	BI / read / 11
title	Friend triangles
pattern	 Country name = $country isPartOf isPartOf isPartOf City City City isLocatedIn isLocatedIn isLocatedIn personA: Person knows knows.creationDate in [$startDate, $endDate] personB: Person knows knows.creationDate in [$startDate, $endDate] knows knows.creationDate in [$startDate, $endDate] personC: Person
description	For a given `$country`, count all the distinct triples of Persons such that: • `personA` is friend of `personB`, • `personB` is friend of `personC`, • `personC` is friend of `personA`, and these friendships were created in the range `[$startDate, $endDate]`. Distinct means that given a triple t_1 in the result set R of all qualified triples, there is no triple t_2 in R such that t_1 and t_2 have the same set of elements.
params	1 | `$country` \| Long String \| Selected from the largest Countries (India, China) 2 \| `$startDate` | Date | Selected from a 30-day interval towards the end of the simulation time 3 | `$endDate` | Date | Selected to yield around a 100-day interval
result	1 | count | 64-bit Integer | A
limit	n/a
CPs	2.3, 2.5, 3.2

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	 2. personCount = count Person count Persons grouped by messageCount value «opt» hasCreator 1. messageCount = count Message content not empty and length < $lengthThreshold and $startDate < creationDate replyOf*0.. Post language in $languages
description	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 creationDate is after $startDate (exclusive, equality is not allowed). • Its length is below the $lengthThreshold (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

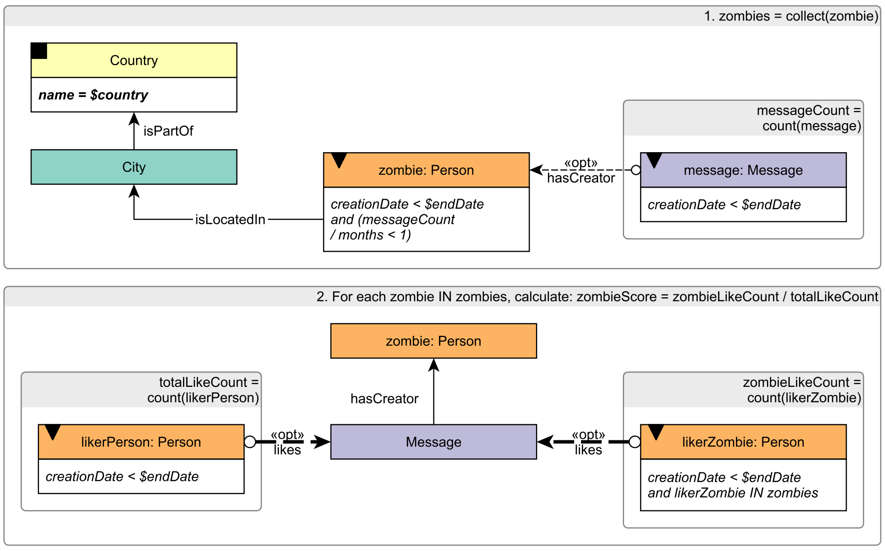
result	1	messageCount	32-bit Integer	A	Number of Messages created
	2	personCount	32-bit Integer	A	Number of Persons with messageCount Messages

sort	1	personCount	↓
	2	messageCount	↓

limit	n/a
CPs	1.1, 1.2, 1.4, 2.6, 3.2, 4.2, 4.3, 8.1, 8.2, 8.3, 8.4, 8.5

BI / read / 13

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 / 13
title	Zombies in a country
pattern	 1. zombies = collect(zombie) Country name = $country isPartOf City isLocatedIn zombie: Person creationDate < $endDate and (messageCount / months < 1) «opt» hasCreator messageCount = count(message) message: Message creationDate < $endDate 2. For each zombie IN zombies, calculate: zombieScore = zombieLikeCount / totalLikeCount zombie: Person hasCreator totalLikeCount = count(likerPerson) likerPerson: Person creationDate < $endDate «opt» likes Message «opt» likes zombieLikeCount = count(likerZombie) likerZombie: Person creationDate < $endDate and likerZombie IN zombies
description	Find zombies within the given `$country`, and return their zombie scores. A `zombie` is a Person created before the given `$endDate`, which has created an average of `[0, 1)` Messages per month, during the time range between profile's `creationDate` and the given `$endDate`. The number of months spans the time range from the `creationDate` of the profile to the `$endDate` with partial months on both end counting as one month (e.g. a `creationDate` of Jan 31 and an `$endDate` of Mar 1 result in 3 months). For each `zombie`, calculate the following: • `zombieLikeCount`: the number of likes received from other zombies. • `totalLikeCount`: the total number of likes received. • `zombieScore`: `zombieLikeCount` / `totalLikeCount`. If the value of `totalLikeCount` is 0, the `zombieScore` of the `zombie` should be 0.0. For both `zombieLikeCount` and `totalLikeCount`, only consider likes received from profiles that were created before the given `$endDate`.

params

1	`$country`	Long String	Selected from the largest Countries (India, China)
2	`$endDate`	Date	Selected from the last days of the initial data set

result

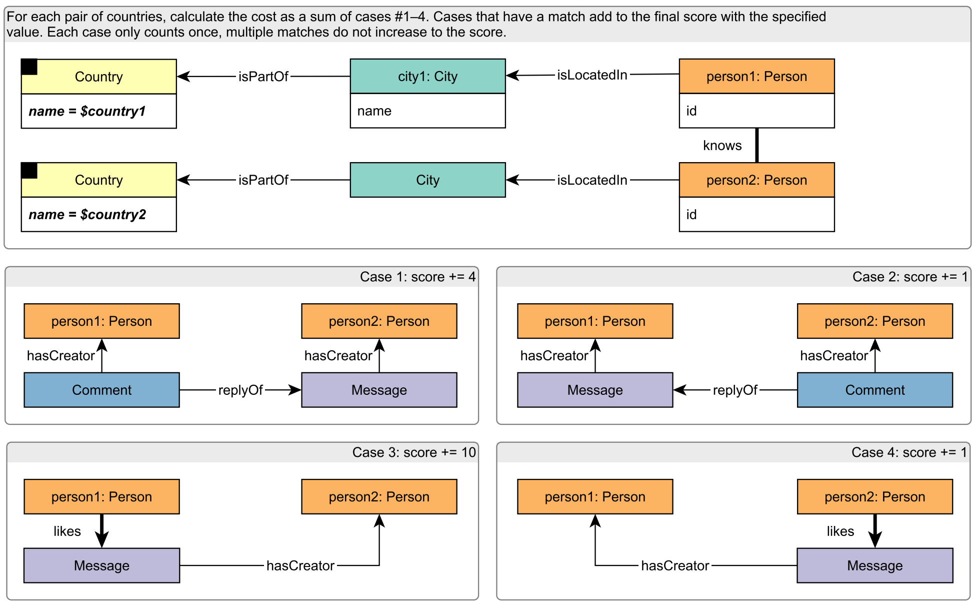
1	`zombie.id`	ID	R	
2	`zombieLikeCount`	32-bit Integer	A	
3	`totalLikeCount`	32-bit Integer	A	
4	`zombieScore`	32-bit Float	A	Determined as `zombieLikeCount` / `totalLikeCount`

sort

1	`zombieScore`	↓	
2	`zombie.id`	↑	

limit	100
CPs	1.2, 2.1, 2.3, 2.4, 2.6, 3.2, 3.3, 4.2, 5.1, 5.3, 8.2, 8.4, 8.5

BI / read / 14

query	BI / read / 14
title	International dialog
pattern	
description	Consider all pairs of people `(person1, person2)` such that (1) they know each other, (2) one is located in a City of `$country1`, and (3) the other is located in a City of `$country2`. For each City of `$country1`, return the highest scoring pair. If there are multiple top-scoring pairs in a city, return the pair with the lowest (`person1.id`, `person2.id`) using a lexicographical ordering. 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	

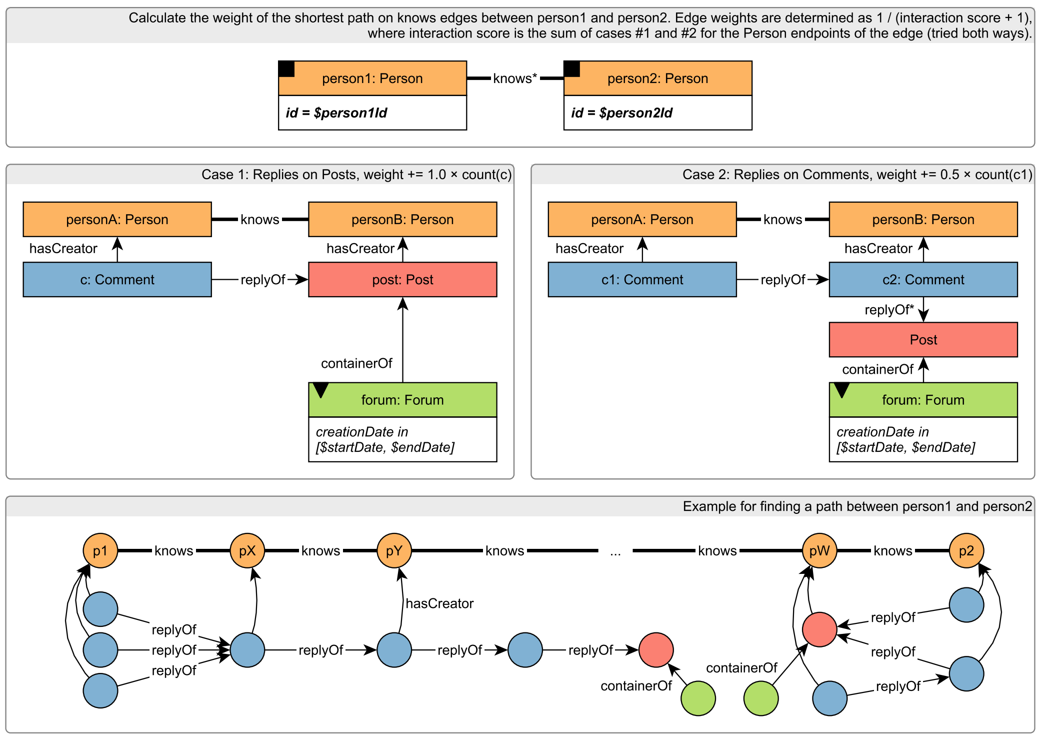
sort

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

limit	100
CPs	1.3, 1.4, 2.1, 3.1, 3.3, 5.1, 5.2, 5.3, 8.3, 8.4

BI / read / 15

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 / 15
title	Trusted connection paths through forums created in a given timeframe
pattern	 Calculate the weight of the shortest path on knows edges between person1 and person2. Edge weights are determined as 1 / (interaction score + 1), where interaction score is the sum of cases #1 and #2 for the Person endpoints of the edge (tried both ways). Case 1: Replies on Posts, weight += 1.0 × count(c) Case 2: Replies on Comments, weight += 0.5 × count(c1) Example for finding a path between person1 and person2 person1: Person id = $person1Id knows* person2: Person id = $person2Id personA: Person knows personB: Person hasCreator c: Comment replyOf post: Post containerOf forum: Forum creationDate in [$startDate, $endDate] c1: Comment c2: Comment replyOf* Post p1 pX pY pW p2 ...
description	Given two Persons with IDs `$person1Id` and `$person2Id`, calculate the cost of the weighted shortest path between these two Persons, in the subgraph induced by the knows relationship. The interaction score of a knows edge is calculated based on the interactions of its Person endpoints: • Every direct reply (by one of the Persons) to a Post (by the other Person) is 1.0 point. • Every direct reply (by one of the Persons) to a Comment (by the other Person) is 0.5 points. Only consider Messages that were created in a Forum that was created within the timeframe (interval) `[$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. The weight for the shortest path algorithm is determined as $\frac{1}{\mathit{interaction\ score}+1}$. The result of the query is a single number, the cost of the weighted shortest path. If no such path exists, the query should return -1.0.

params

1	`$person1Id`	ID	(a) `$person1Id` – `$person2Id` pair with a distance of 4 hops (b) `$person1Id` – `$person2Id` pair with a distance of 2 hops
2	`$person2Id`	ID	
3	`$startDate`	Date	(a) Small interval (approx. one week) (b) Big interval (approx. one month)
4	`$endDate`	Date	

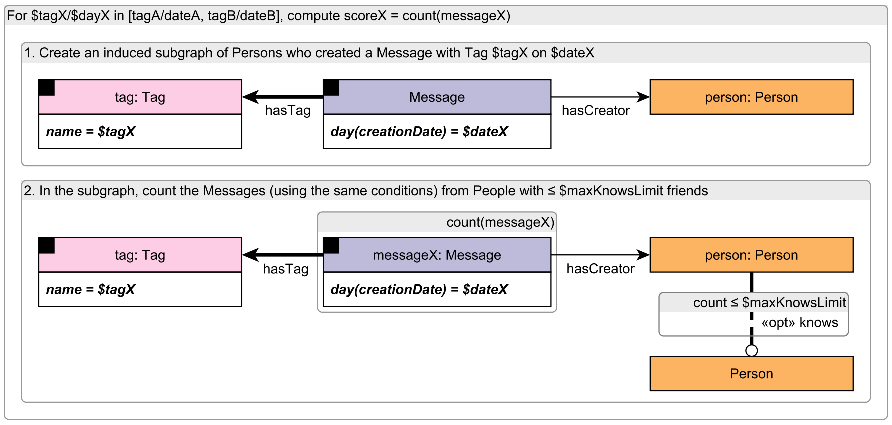
result

1	`weight`	32-bit Float	C	

limit	n/a
CPs	1.2, 2.1, 2.2, 2.4, 3.3, 5.1, 5.3, 7.2, 7.3, 7.6, 7.7, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6

BI / read / 16

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 / 16
title	Fake news detection
pattern	 For $tagX/$dayX in [tagA/dateA, tagB/dateB], compute scoreX = count(messageX) 1. Create an induced subgraph of Persons who created a Message with Tag $tagX on $dateX tag: Tag name = $tagX hasTag Message day(creationDate) = $dateX hasCreator person: Person 2. In the subgraph, count the Messages (using the same conditions) from People with ≤ $maxKnowsLimit friends tag: Tag name = $tagX hasTag count(messageX) messageX: Message day(creationDate) = $dateX hasCreator person: Person count ≤ $maxKnowsLimit «opt» knows Person
description	Given two Tag/date pairs (`$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

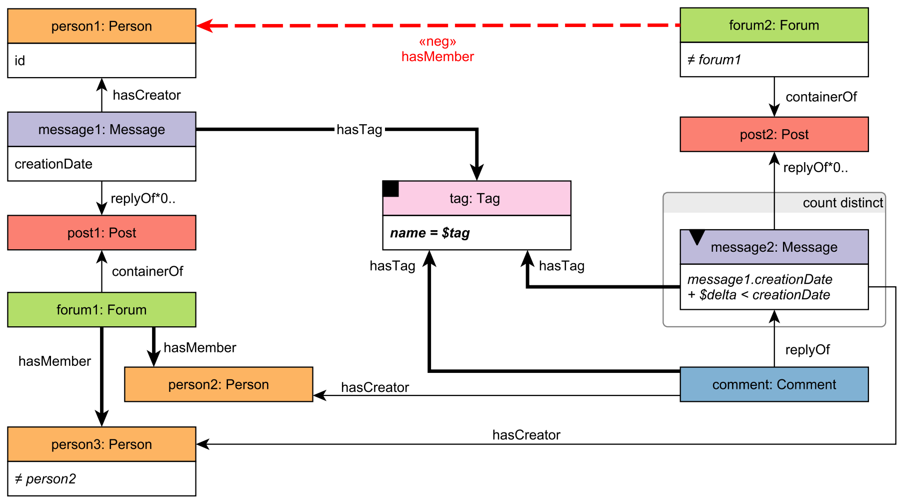
1	`person.id`	ID	R	
2	`messageCountA`	32-bit Integer	A	Message count for `$tagA`/`$dateA`
3	`messageCountB`	32-bit Integer	A	Message count for `$tagB`/`$dateB`

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 / read / 17

query	BI / read / 17
title	Information propagation analysis
pattern	 person1: Person id «neg» hasMember forum2: Forum ≠ forum1 hasCreator containerOf message1: Message creationDate hasTag post2: Post replyOf*0.. tag: Tag name = $tag count distinct replyOf*0.. post1: Post message2: Message message1.creationDate + $delta < creationDate hasTag hasTag containerOf forum1: Forum hasMember replyOf hasMember person2: Person hasCreator comment: Comment person3: Person ≠ person2 hasCreator
description	This query aims to identify instances of "information propagation" when a Person (person1) submits a Message (message1) with a given $tag to a Forum (forum1). This is read by other members of forum1, Persons person2 and person3 (who must be different Persons). 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.

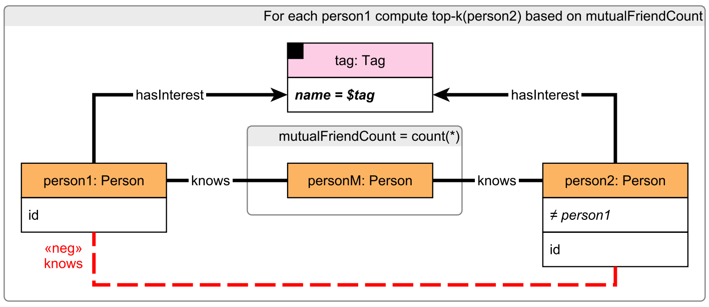
params	1	$tag	Long String	Tags with a similar amount of Messages are selected
	2	$delta	32-bit Integer	Measured in hours, selected to be between 8 and 16 hours.

result	1	person1.id	ID	R	
	2	messageCount	32-bit Integer	A	

sort	1	messageCount	↓	
	2	person1.id	↑	

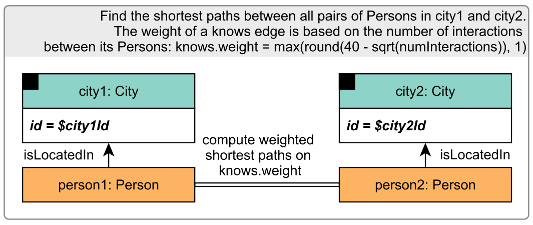
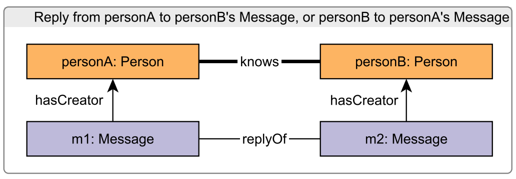
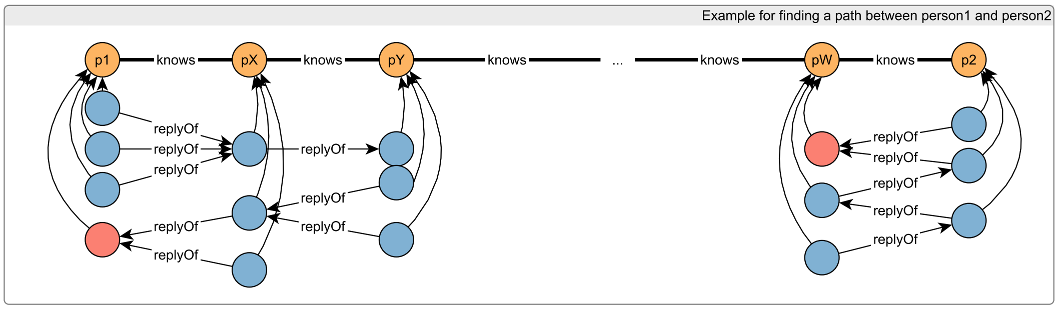
limit	10
CPs	2.1, 2.3, 2.5, 2.6, 8.1

BI / read / 18

query	BI / read / 18
title	Friend recommendation
pattern	 For each person1 compute top-k(person2) based on mutualFriendCount tag: Tag name = $tag hasInterest hasInterest mutualFriendCount = count(*) person1: Person id knows personM: Person knows person2: Person ≠ person1 id «neg» knows
description	For a given `$tag`, for each `person1` interested in `$tag`, recommend new friends (`person2`) who • do not yet know `person1` • have at least one mutual friend with `person1` • are also interested in `$tag`. Rank Persons `person2` based on the number of mutual friends with `person1`.
params	1 | `$tag` | Long String | Tags with a similar amount of Persons are selected
result	1 | `person1.id` | ID | R 2 | `person2.id` | ID | R 3 | `mutualFriendCount` | 32-bit Integer | A
sort	1 | `mutualFriendCount` | ↓ 2 | `person1.id` | ↑ 3 | `person2.id` | ↑
limit	20
CPs	2.5, 2.6, 8.1

BI / read / 19

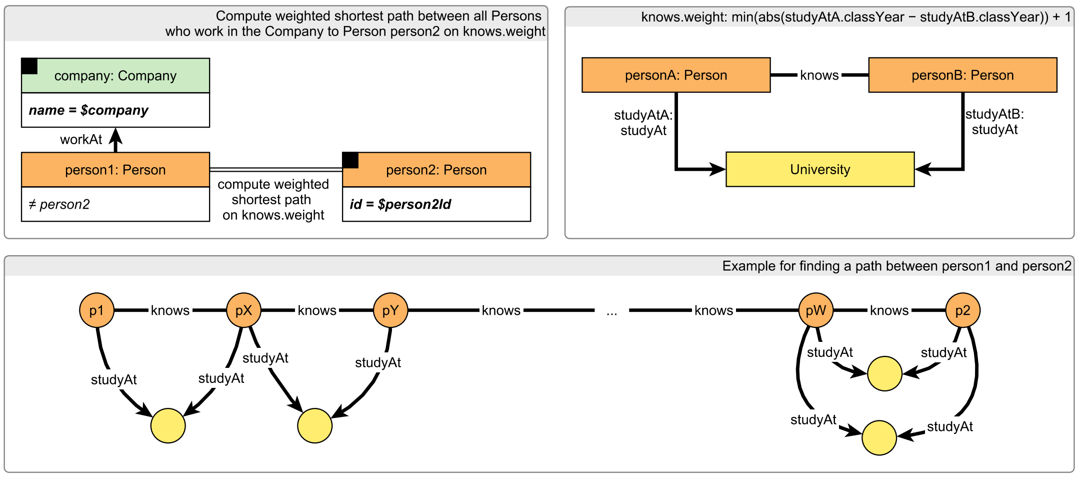
BI 1
BI 2
BI 3
BI 4
BI 5
BI 6
BI 7
BI 8
BI 9
BI 10
BI 11
BI 12
BI 13
BI 14
BI 15
BI 16
BI 17
BI 18
BI 19
BI 20

query	BI / read / 19
title	Interaction path between cities
pattern	  
description	Given two Cities with IDs `$city1Id`, `$city2Id`, find Persons `person1`, `person2` living in these Cities (respectively) with the *cheapest* interaction path between them. The cheapest path is equivalent to the *weighted shortest* path. It is computed on a subgraph of the Person-knows-Person graph with the edge weights based on the number of interactions. An *interaction* is a direct reply Comments from one Person to Messages by the other Person. Only knows edges with at least one interaction between their endpoint Persons are considered. For these, the weight of a knows edge is defined as: $\max(\mathrm{round}(40 - \sqrt{numInteractions}), 1)$ If there are multiple pairs of people with cheapest paths that have the same total weight, return all of them. *Note:* Interactions are counted both ways, e.g. if Alice knows Bob, 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 and the weight of the knows edge is 38. *Remark:* Determinism is ensured by using square root followed by rounding. For all integers between 1 and 100 000, the square root's fractional part is more than 10e-5 from 0.5, where the rounding could be non-determinstic based on floating point inaccuracies. As 10e-5 is significantly larger than the machine epsilon of IEEE 754 floats (both 32- and 64-bit), the floating point inaccuracies have no chance to affect the derived integer edge weights.
params	1 | `$city1Id` \| ID \| (a) Small Cities within the same Country (b) Larger Cities from different Countries 2 \| `$city2Id` | ID |
result	1 | `person1.id` | ID | R 2 | `person2.id` | ID | R 3 | `totalWeight` | 32-bit Integer | C
sort	1 | `person1.id` | ↑ 2 | `person2.id` | ↑
limit	n/a
CPs	3.3, 7.6, 7.7, 8.4, 8.6
relevance	To find the weighted shortest paths efficiently, the system can use e.g. a bidirectional Dijkstra algorithm. As the edge weights do not depend on any parameter, systems can pre-compute them (if they do not interleave reads and writes).

Find the shortest paths between all pairs of Persons in city1 and city2.
The weight of a knows edge is based on the number of interactions
between its Persons: knows.weight = max(round(40 - sqrt(numInteractions)), 1)
city1: City
id = $city1Id
isLocatedIn
person1: Person
compute weighted shortest paths on knows.weight
city2: City
id = $city2Id
isLocatedIn
person2: Person

BI / read / 20

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 / 20
title	Recruitment
pattern	 Compute weighted shortest path between all Persons who work in the Company to Person person2 on knows.weight company: Company name = $company workAt person1: Person ≠ person2 compute weighted shortest path on knows.weight person2: Person id = $person2Id knows.weight: min(abs(studyAtA.classYear − studyAtB.classYear)) + 1 personA: Person knows personB: Person studyAtA: studyAt studyAtB: studyAt University Example for finding a path between person1 and person2 p1 knows pX knows pY knows ... knows pW knows p2 studyAt studyAt studyAt studyAt studyAt studyAt studyAt studyAt
description	Consider knows edges where the endpoint Persons attended the same University and set the weight of the edge to the absolute difference between the year of enrolment plus 1. If the Persons attended multiple universities, we select the smallest (min) value. Formally: $$w = \min_{\text{studyAt}_A,\text{studyAt}_B} \lvert\text{studyAt}_A.\text{classYear} - \text{studyAt}_B.\text{classYear}\rvert + 1$$ Given a $company and a Person person2 with ID $person2Id (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 from person2 through people who have studied together through the shortest weighted path. If there are multiple Person person1 nodes with the same shortest path length, return all of them.
params	1 · $company · Long String · Companies with a similar number of employees (former or current) are selected 2 · $person2Id · ID · (a) There is guaranteed to be no path between any person1 working at company and person2 (b) There is guaranteed to be a 2-hop path between at least one person1 working at company and person
result	1 · person1.id · ID · R 2 · totalWeight · 32-bit Integer · C
sort	1 · totalWeight · ↑ 2 · person1.id · ↑
limit	20
CPs	3.3, 7.6, 7.7, 7.8, 8.4, 8.6
relevance	To find the weighted shortest paths efficiently, the system can use e.g. a bidirectional Dijkstra algorithm. As the edge weights do not depend on any parameter, systems can pre-compute them (if they do not interleave reads and writes).