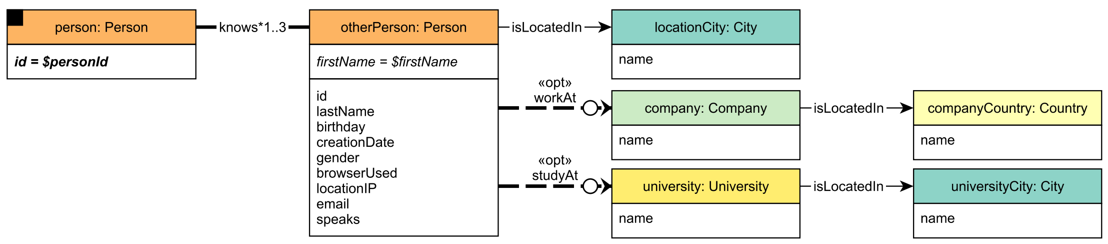
Interactive / complex / 1

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 1
title	Transitive friends with a certain name
pattern	 person: Person id = $personId knows*1..3 otherPerson: Person firstName = $firstName id lastName birthday creationDate gender browserUsed locationIP email speaks isLocatedIn locationCity: City name «opt» workAt company: Company name isLocatedIn companyCountry: Country name «opt» studyAt university: University name isLocatedIn universityCity: City name
description	Given a start Person with ID `$personId`, find Persons with a given first name (`$firstName`) that the start Person is connected to (excluding start Person) by at most 3 steps via the knows relationships. Return Persons, including the distance (1..3), summaries of the Persons workplaces and places of study.

params			
1	`$personId`	ID	
2	`$firstName`	String	

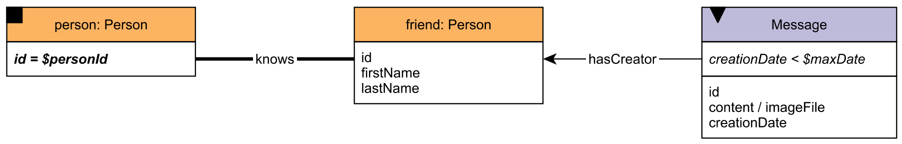
result				
1	`otherPerson.id`	ID	R	
2	`otherPerson.lastName`	String	R	
3	`distanceFromPerson`	32-bit Integer	C	
4	`otherPerson.birthday`	Date	R	
5	`otherPerson.creationDate`	DateTime	R	
6	`otherPerson.gender`	String	R	
7	`otherPerson.browserUsed`	String	R	
8	`otherPerson.locationIP`	String	R	
9	`otherPerson.email`	{Long String}	R	
10	`otherPerson.speaks`	{String}	R	
11	`locationCity.name`	String	R	
12	`universities`	{<String, 32-bit Integer, String>}	A	`{<university.name, studyAt.classYear, universityCity.name>}`
13	`companies`	{<String, 32-bit Integer, String>}	A	`{<company.name, workAt.workFrom, companyCountry.name>}`

sort			
1	`distanceFromPerson`	↑	
2	`otherPerson.lastName`	↑	
3	`otherPerson.id`	↑	

limit	20
CPs	2.1, 5.3, 8.2
relevance	This query is a representative of a simple navigational query. It is interesting for several aspects. (1) It requires for a complex aggregation for returning the concatenation of universities, companies, languages and email information of the Person. (2) It tests the ability of the optimizer to move the evaluation of sub-queries functionally dependant on the Person, after the evaluation of the top-k. (3) Its performance is highly sensitive to properly estimating the cardinalities in each transitive path, and paying attention not to explore already visited Persons.

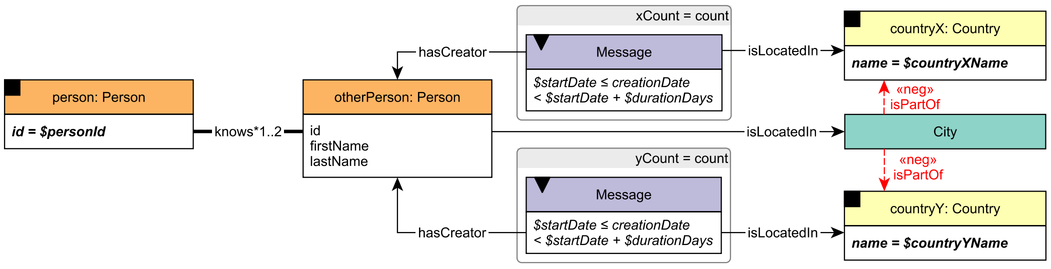
Interactive / complex / 2

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 2
title	Recent messages by your friends
pattern	 person: Person id = $personId knows friend: Person id firstName lastName hasCreator Message creationDate < $maxDate id content / imageFile creationDate
description	Given a start Person with ID `$personId`, find the most recent Messages from all of that Person's friends (`friend` nodes). Only consider Messages created before the given `$maxDate` (excluding that day).
params	1 | `$personId` \| ID 2 \| `$maxDate` | Date
result	1 | `friend.id` | ID | R 2 | `friend.firstName` | String | R 3 | `friend.lastName` | String | R 4 | `message.id` | ID | R 5 | `message.content` or `message.imageFile` (for photos) | Text | R 6 | `message.creationDate` | DateTime | R
sort	1 | `message.creationDate` | ↓ 2 | `message.id` | ↑
limit	20
CPs	1.1, 2.2, 2.3, 3.2, 8.5
relevance	This is a navigational query looking for paths of length two, starting from a given Person, going to their friends and from them, moving to their published Posts and Comments. This query exercices both the optimizer and how data is stored. It tests the ability to create execution plans taking advantage of the orderings induced by some operators to avoid performing expensive sorts. This query requires selecting Posts and Comments based on their creation date, which might be correlated with their identifier and therefore, having intermediate results with interesting orders. Also, messages could be stored in an order correlated with their creation date to improve data access locality. Finally, as many of the attributes required in the projection are not needed for the execution of the query, it is expected that the query optimizer will move the projection to the end.

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

Interactive / complex / 3

query	Interactive / complex / 3
title	Friends and friends of friends that have been to given countries
pattern	
description	Given a start Person with ID `$personId`, find Persons that are their friends and friends of friends (excluding the start Person) that have made Posts / Comments in both of the given Countries (named `$countryXName` and `$countryYName`), within `[$startDate, $startDate + $durationDays)` (closed-open interval). Only Persons that are foreign to these Countries are considered, that is Persons whose location Country is neither named `$countryXName` nor `$countryYName`.

params

1	`$personId`	ID	
2	`$countryXName`	String	In SNB Interactive v2, this query has two variants: (a) Correlated Countries (b) Anti-correlated Countries
3	`$countryYName`	String	
4	`$startDate`	Date	Beginning of requested period
5	`$durationDays`	32-bit Integer	Duration of requested period, in days. The interval `[$startDate, $startDate + $durationDays)` is closed-open

result

1	`otherPerson.id`	ID	R	
2	`otherPerson.firstName`	String	R	
3	`otherPerson.lastName`	String	R	
4	`xCount`	32-bit Integer	A	Number of Messages from Country named `$countryXName` created by the Person within the given time
5	`yCount`	32-bit Integer	A	Number of Messages from Country named `$countryYName` created by the Person within the given time
6	`count`	32-bit Integer	A	`count` = `xCount` + `yCount`

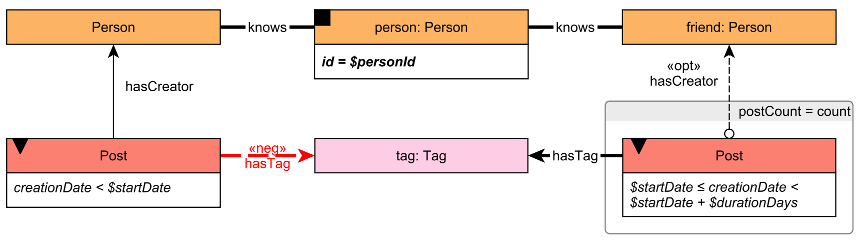
sort

1	`count`	↓	
2	`otherPerson.id`	↑	

limit	20
CPs	2.1, 3.1, 5.1, 8.2, 8.5
relevance	This query looks for paths of length two and three, starting from a Person, going to friends or friends of friends, and then moving to Messages. This query tests the ability of the query optimizer to select the most efficient join ordering, which will depend on the cardinalities of the intermediate results. Many friends of friends can be duplicate, then it is expected to eliminate duplicates and those people prior to access the Post and Comments, as well as eliminate those friends from Countries named `$countryXName` and `$countryYName`, as the size of the intermediate results can be severely affected. A possible structural optimization could be to materialize the number of Posts and Comments created by a Person, and progressively filter those people that could not even fall in the top 20 even having all their posts in the Countries named `$countryXName` and `$countryYName`.

Interactive / complex / 4

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 4
title	New topics
pattern	
description	Given a start Person with ID `$personId`, find Tags that are attached to Posts that were created by that Person's friends. Only include Tags that were attached to friends' Posts created within a given time interval `[$startDate, $startDate + $durationDays)` (closed-open) and that were never attached to friends' Posts created before this interval.

params				
	1	`$personId`	ID	
	2	`$startDate`	Date	
	3	`$durationDays`	32-bit Integer	Duration of requested period, in days. The interval `[$startDate, $startDate + $durationDays)` is closed-open

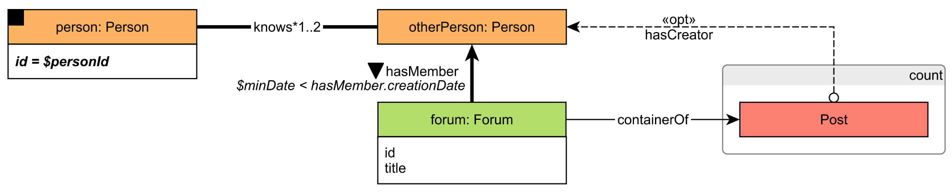
result					
	1	`tag.name`	Long String	R	
	2	`postCount`	32-bit Integer	A	Number of Posts made within the given time interval that have `tag`

sort				
	1	`postCount`	↓	
	2	`tag.name`	↑	

limit	10
CPs	2.3, 8.2, 8.5
relevance	This query looks for paths of length two, starting from a given Person, moving to Posts and then to Tags. It tests the ability of the query optimizer to properly select the usage of hash joins or index based joins, depending on the cardinality of the intermediate results. These cardinalities are clearly affected by the input Person, the number of friends, the variety of Tags, the time interval and the number of Posts.

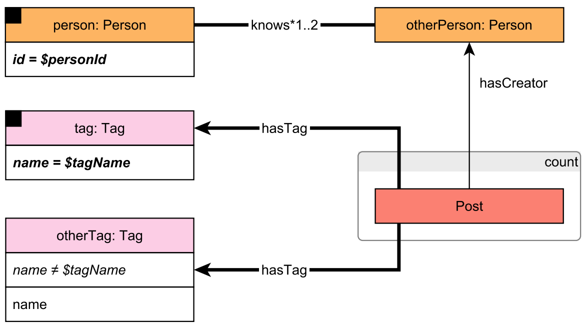
Interactive / complex / 5

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 5				
title	New groups				
pattern	 person: Person id = $personId knows*1..2 otherPerson: Person «opt» hasCreator hasMember $minDate < hasMember.creationDate count forum: Forum id title containerOf Post				
description	Given a start Person with ID `$personId`, denote their friends and friends of friends (excluding the start Person) as `otherPerson`. Find Forums that any Person `otherPerson` became a member of after a given date (`$minDate`). For each of those Forums, count the number of Posts that were created by the Person `otherPerson`.				
params	1	`$personId`	ID		
	2	`$minDate`	Date		
result	1	`forum.title`	Long String	R	
	2	`postCount`	32-bit Integer	A	Number of Posts made in `forum` that were created by the Person `otherPerson`
sort	1	`postCount`	↓		
	2	`forum.id`	↑		
limit	20				
CPs	2.3, 3.3, 8.2, 8.5				
relevance	This query looks for paths of length two and three, starting from a given Person, moving to friends and friends of friends, and then getting the Forums they are members of. Besides testing the ability of the query optimizer to select the proper join operator, it rewards the usage of indices, but their accesses will be presumably scattered due to the two/three-hop search space of the query, leading to unpredictable and scattered index accesses. Having efficient implementations of such indices will be highly beneficial.				

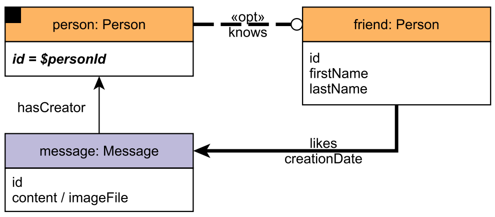
IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

Interactive / complex / 6

query	Interactive / complex / 6
title	Tag co-occurrence
pattern	 person: Person id = $personId knows*1..2 otherPerson: Person hasCreator tag: Tag name = $tagName hasTag count Post otherTag: Tag name ≠ $tagName name hasTag
description	Given a start Person with ID `$personId` and a Tag with name `$tagName`, find the other Tags that occur together with this Tag on Posts that were created by start Person's friends and friends of friends (excluding start Person). Return top 10 Tags, and the count of Posts that were created by these Persons, which contain both this Tag and the given Tag.
params	1 `$personId` ID 2 `$tagName` Long String
result	1 `otherTag.name` Long String R 2 `postCount` 32-bit Integer A Number of Posts that were created by friends and friends of friends, which have the Tag `otherTag`
sort	1 `postCount` ↓ 2 `otherTag.name` ↑
limit	10
CPs	5.1, 8.2
relevance	This query looks for paths of lengths three or four, starting from a given Person, moving to friends or friends of friends, then to Posts and finally ending at a given Tag.

Interactive / complex / 7

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 7
title	Recent likers
pattern	 person: Person id = $personId «opt» knows friend: Person id firstName lastName hasCreator message: Message id content / imageFile likes creationDate
description	Given a start Person with ID `$personId`, find the most recent `likes` on any of start Person's Messages. Find Persons that liked (`likes` edge) any of start Person's Messages, the Messages they liked most recently, the creation date of that like, and the latency in minutes (`minutesLatency`) between creation of Messages and like. Additionally, for each Person found return a flag indicating (`isNew`) whether the liker is a friend of start Person. In case that a Person liked multiple Messages at the same time, return the Message with lowest identifier. *Validation rule:* Depending on whether the system-under-test supports leap seconds or uses UTC-SLS (UTC with Smoothed Leap Seconds), a difference of 1 minute can occur between the `minutesLatency` results of two correct implementations when the time interval includes June 30, 2012, when there was a leap second. Therefore, the `minutesLatency` value is validated using a tolerance of 1 minute.

params

1	`$personId`	ID

result

1	`friend.id`	ID	R	friend.id = personId is allowed
2	`friend.firstName`	String	R	
3	`friend.lastName`	String	R	
4	`likes.creationDate`	DateTime	R	
5	`message.id`	ID	R	
6	`message.content or message.imageFile (for photos)`	Text	R	
7	`minutesLatency`	32-bit Integer	C	Duration between the creation of the Message and the creation of the like, in minutes.
8	`isNew`	Boolean	C	`False` if `person` and `friend` know each other, `True` otherwise

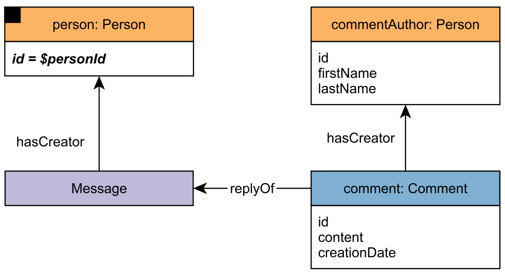
sort

1	`likes.creationDate`	↓
2	`friend.id`	↑

limit	20
CPs	2.2, 2.3, 3.3, 5.1, 8.1, 8.3
relevance	This query looks for paths of length two, starting from a given Person, moving to its published messages and then to Persons who liked them. It tests several aspects related to join optimization, both at query optimization plan level and execution engine level. On the one hand, many of the columns needed for the projection are only needed in the last stages of the query, so the optimizer is expected to delay the projection until the end. This query implies accessing two-hop data, and as a consequence, index accesses are expected to be scattered. We expect to observe variate cardinalities, depending on the characteristics of the input parameter, so properly selecting the join operators will be crucial. This query has a lot of correlated sub-queries, so it is testing the ability to flatten the query execution plans.

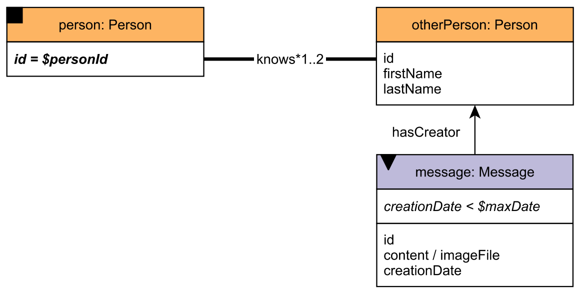
Interactive / complex / 8

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 8
title	Recent replies
pattern	 person: Person id = $personId commentAuthor: Person id firstName lastName hasCreator hasCreator Message replyOf comment: Comment id content creationDate
description	Given a start Person with ID $personId, find the most recent Comments that are replies to Messages of the start Person. Only consider direct (single-hop) replies, not the transitive (multi-hop) ones. Return the reply Comments, and the Person that created each reply Comment.
params	1 | `$personId` | ID
result	1 | `commentAuthor.id` | ID | R 2 | `commentAuthor.firstName` | String | R 3 | `commentAuthor.lastName` | String | R 4 | `comment.creationDate` | DateTime | R 5 | `comment.id` | ID | R 6 | `comment.content` | Text | R
sort	1 | `comment.creationDate` | ↓ 2 | `comment.id` | ↑
limit	20
CPs	2.4, 3.3, 5.3
relevance	This query looks for paths of length two, starting from a given Person, going through its created Messages and finishing at their replies. In this query there is temporal locality between the replies being accessed. Thus the top-k order by this can interact with the selection, i.e. do not consider older Posts than the 20th oldest seen so far.

Interactive / complex / 9

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

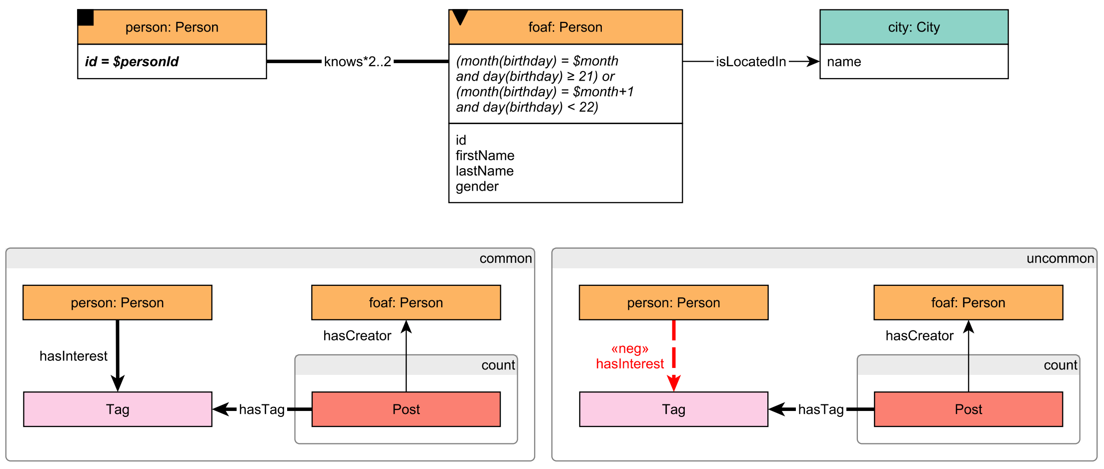
query	Interactive / complex / 9
title	Recent messages by friends or friends of friends
pattern	 person: Person id = $personId knows*1..2 otherPerson: Person id firstName lastName hasCreator message: Message creationDate < $maxDate id content / imageFile creationDate
description	Given a start Person with ID `$personId`, find the most recent Messages created by that Person's friends or friends of friends (excluding the start Person). Only consider Messages created before the given `$maxDate` (excluding that day).
params	1 `$personId` ID 2 `$maxDate` Date
result	1 `otherPerson.id` ID R 2 `otherPerson.firstName` String R 3 `otherPerson.lastName` String R 4 `message.id` ID R 5 `message.content or message.imageFile (for photos)` Text R 6 `message.creationDate` DateTime R
sort	1 `message.creationDate` ↓ 2 `message.id` ↑
limit	20
CPs	1.1, 1.2, 2.2, 2.3, 3.2, 3.3, 8.5
relevance	This query looks for paths of length two or three, starting from a given Person, moving to its friends and friends of friends, and ending at their created Messages. This is one of the most complex queries, as the list of choke points indicates. This query is expected to touch variable amounts of data with entities of different characteristics, and therefore, properly estimating cardinalities and selecting the proper operators will be crucial.

Interactive / complex / 10

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 10
title	Friend recommendation

pattern



description

Given a start Person with ID `$personId`, find that Person's friends of friends (`foaf`) – excluding the start Person and his/her immediate friends –, who were born on or after the 21st of a given `$month` (in any year) and before the 22nd of the following month. Calculate the similarity between each `friend` and the start `person`, where `commonInterestScore` is defined as follows:

- `common` = number of Posts created by `friend`, such that the Post has a Tag that the start `person` is interested in
- `uncommon` = number of Posts created by `friend`, such that the Post has no Tag that the start `person` is interested in
- `commonInterestScore` = `common` - `uncommon`

params

1	`$personId`	ID	
2	`$month`	32-bit Integer	Between 1 and 12. Implementations may also pass the next month as an additional `$nextMonth` parameter

result

1	`foaf.id`	ID	R	
2	`foaf.firstName`	String	R	
3	`foaf.lastName`	String	R	
4	`commonInterestScore`	32-bit Integer	A	
5	`foaf.gender`	String	R	
6	`city.name`	String	R	

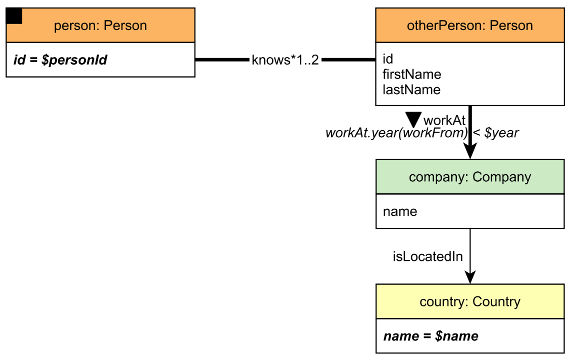
sort

1	`commonInterestScore`	↓	
2	`foaf.id`	↑	

limit	10
CPs	2.3, 3.3, 4.1, 4.2, 5.1, 5.2, 6.1, 7.1, 8.6
relevance	This query looks for paths of length two, starting from a Person and ending at the friends of their friends. It does widely scattered graph traversal, and one expects no locality of in friends of friends, as these have been acquired over a long time and have widely scattered identifiers. The join order is simple but one must see that the anti-join for "not in my friends" is better with hash. Also the last pattern in the scalar sub-queries joining or anti-joining the Tags of the candidate's Posts to interests of self should be by hash.

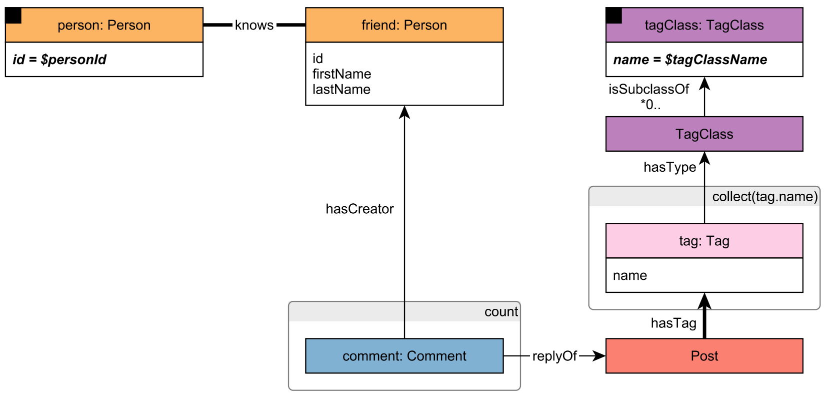
Interactive / complex / 11

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 11			
title	Job referral			
pattern				
description	Given a start Person with ID `$personId`, find that Person's friends and friends of friends (excluding start Person) who started working in some Company in a given Country with name `$countryName`, before a given date (`$workFromYear`).			
params	1	`$personId`	ID	
	2	`$countryName`	String	
	3	`$workFromYear`	32-bit Integer	
result	1	`otherPerson.id`	ID	R
	2	`otherPerson.firstName`	String	R
	3	`otherPerson.lastName`	String	R
	4	`company.name`	String	R
	5	`workAt.workFrom`	32-bit Integer	R
sort	1	`workAt.workFrom`	↑	
	2	`otherPerson.id`	↑	
	3	`company.name`	↓	
limit	10			
CPs	1.3, 2.3, 2.4, 3.3, 4.2			
relevance	This query looks for paths of length two or three, starting from a Person, moving to friends or friends of friends, and ending at a Company. In this query, there are selective joins and a top-k order by that can be exploited for optimizations.			

Interactive / complex / 12

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 12
title	Expert search
pattern	 person: Person id = $personId knows friend: Person id firstName lastName tagClass: TagClass name = $tagClassName isSubclassOf *0.. TagClass hasType collect(tag.name) tag: Tag name hasCreator count hasTag comment: Comment replyOf Post
description	Given a start Person with ID `$personId`, find the Comments that this Person's friends made in reply to Posts, considering only those Comments that are direct (single-hop) replies to Posts, not the transitive (multi-hop) ones. Only consider Posts with a Tag in a given TagClass with name `$tagClassName` or in a descendent of that TagClass. Count the number of these reply Comments, and collect the Tags that were attached to the Posts they replied to, but only collect Tags with the given TagClass or with a descendant of that TagClass. Return Persons with at least one reply, the reply count, and the collection of Tags.

params

1	`$personId`	ID
2	`$tagClassName`	Long String

result

1	`friend.id`	ID	R
2	`friend.firstName`	String	R
3	`friend.lastName`	String	R
4	`tagNames`	{Long String}	A
5	`replyCount`	32-bit Integer	A

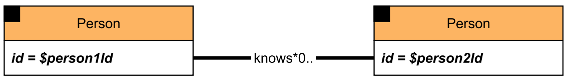
sort

1	`replyCount`	↓
2	`friend.id`	↑

limit	20
CPs	3.3, 7.2, 7.3, 8.2
relevance	This query starts at a Person, moves to its friends, and the to their Comments and their root Posts. Then, it gets the Tag of each Post and checks whether it (directly or transitively) belongs to the specified TagClass. This can be thought of a bidirectional search between the Person and the TagClass. The difficulty of this query is determining the optimal direction of this traversal.

Interactive / complex / 13

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

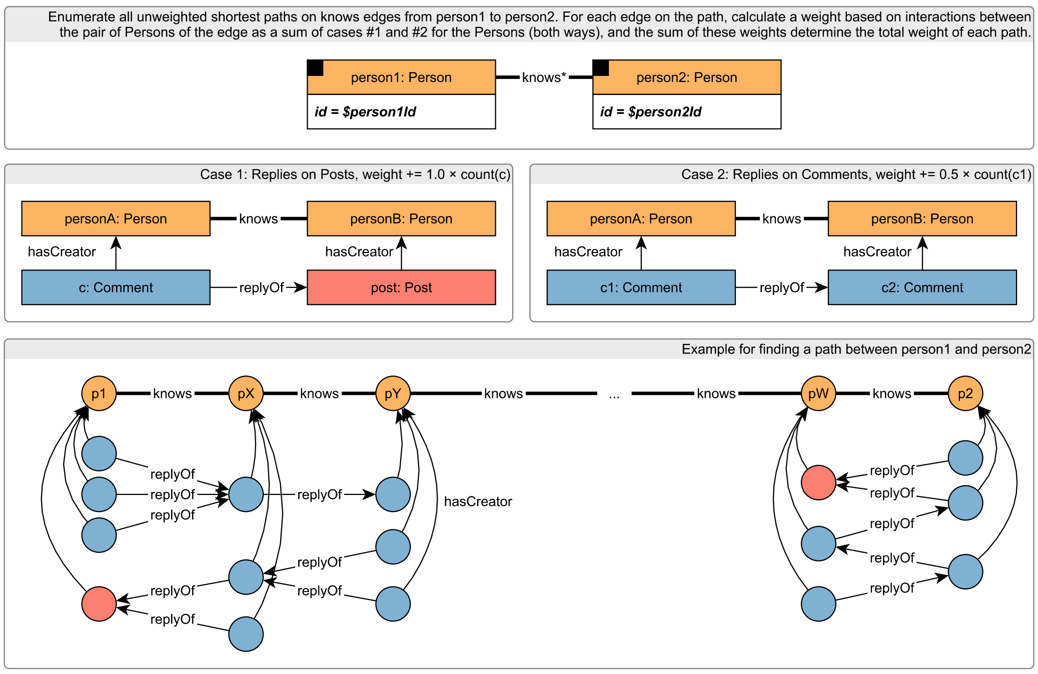
query	Interactive / complex / 13
title	Single shortest path
pattern	 Person id = $person1Id knows*0.. Person id = $person2Id
description	Given two Persons with IDs `$person1Id` and `$person2Id`, find the shortest path between these two Persons in the subgraph induced by the knows edges. Return the length of this path: • -1: no path found • 0: start person = end person • > 0: path found (start person ≠ end person)
params	1 | `$person1Id` \| ID \| In SNB Interactive v2, this query has two variants: (b) Guaranteed that there is no path between the two Persons (b) Guaranteed that there is a 4-hop path between the two Persons 2 \| `$person2Id` | ID
result	1 | `shortestPathLength` | 32-bit Integer | C
CPs	3.3, 7.2, 7.3, 7.5, 7.8, 8.1, 8.6
relevance	This query looks for a variable length path, starting at a given Person and finishing at an another given Person. Proper cardinality estimation and search space pruning, will be crucial. This query also allows for possible parallel implementations.

Interactive / complex / 14v1

IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

query	Interactive / complex / 14v1
title	Trusted connection paths (v1)

pattern


Enumerate all unweighted shortest paths on knows edges from person1 to person2. For each edge on the path, calculate a weight based on interactions between the pair of Persons of the edge as a sum of cases #1 and #2 for the Persons (both ways), and the sum of these weights determine the total weight of each path.

description

This query is used in SNB Interactive v1.
Given two Persons with IDs `$person1Id` and `$person2Id`, 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) is 1.0.
- Every direct reply (by one of the Persons) to a Comment (by the other Person) is 0.5.

Note that interactions are counted both ways (e.g. if Alice writes 2 Post replies and 1 Comment reply to Bob, while Bob writes 3 Post replies and 4 Comment replies to Alice, their interaction score is $2 \times 1.0 + 1 \times 0.5 + 3 \times 1.0 + 4 \times 0.5 = 7.5$).
Return all the paths with shortest length and their weights. Do not return any rows if there is no path between the two Persons.

params

1	`$person1Id`	ID	
2	`$person2Id`	ID	

result

1	`personIdsInPath`	[ID]	C	Identifiers representing an ordered sequence of the Persons in the path
2	`pathWeight`	64-bit Float	C	

sort

1	`pathWeight`	↓	The order of paths with the same weight is unspecified

CPs	3.3, 5.3, 7.2, 7.3, 7.5, 7.7, 8.1, 8.2, 8.3, 8.6
relevance	This query looks for a variable length path, starting at a given Person and finishing at an another given Person. This is a more complex query as it not only requires computing the path length, but returning it and computing a weight. To compute this weight one must look for smaller sub-queries with paths of length three, formed by the two Persons at each step, a Post and a Comment.