1 Interactive Workload

This workload consists of a set of relatively complex read-only queries, that touch a significant amount of data – often the two-step friendship neighbourhood and associated messages –, but typically in close proximity to a single node. Hence, the query complexity is sublinear to the dataset size.

The LDBC SNB Interactive workload consists of three query classes:

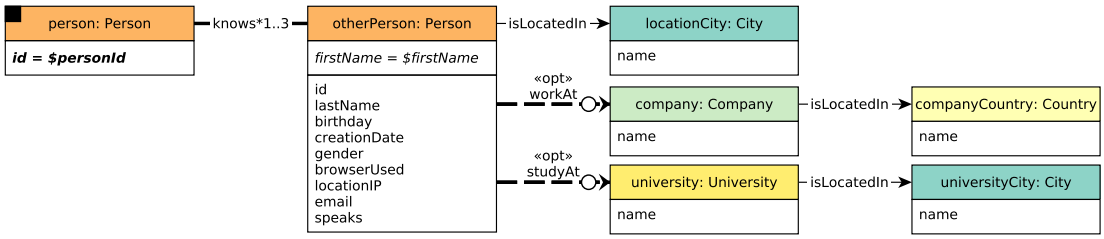
- **Complex read-only queries.** See Section 1.1.
- **Short read-only queries.** See Section 1.2.
- **Transactional update queries inserting new entities.** See Section 1.3.

A detailed description of the workload (covering reads and inserts) is available in the paper published at SIGMOD 2015 [1].

1.1 Complex Reads

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 14

query	Interactive / complex / 1
title	Transitive friends with certain name
pattern	
desc.	Given a start Person, 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›}

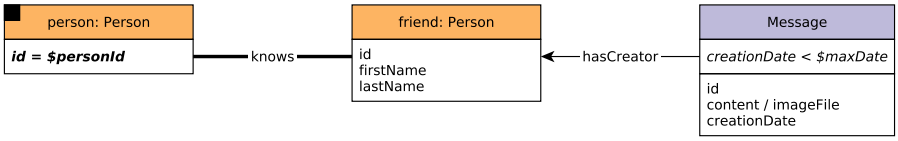
sort

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 looks for paths of length 1..3 through the knows relation, starting from a given Person and ending at a Person with a given first name. 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 14

query	Interactive / complex / 2
title	Recent messages by your friends
pattern	
desc.	Given a start Person (`person`), 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

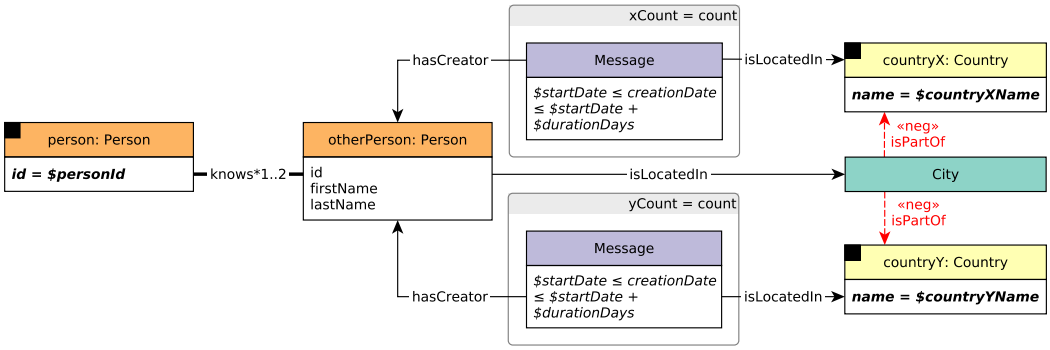
sort

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.

Interactive / complex / 3

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 14

query	Interactive / complex / 3
title	Friends and friends of friends that have been to given countries
pattern	
desc.	Given a start Person, find Persons that are their friends and friends of friends (excluding start Person) that have made Posts / Comments in both of the given Countries, `CountryX` and `CountryY`, within a given period. Only Persons that are foreign to Countries `CountryX` and `CountryY` are considered, that is Persons whose location is neither `CountryX` nor `CountryY`.

params

1	personId	ID	
2	countryXName	String	
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 `CountryX` created by the Person within the given time
5	yCount	32-bit Integer	A	Number of Messages from Country `CountryY` created by the Person within the given time
6	count	32-bit Integer	A	`count = xCount + yCount`

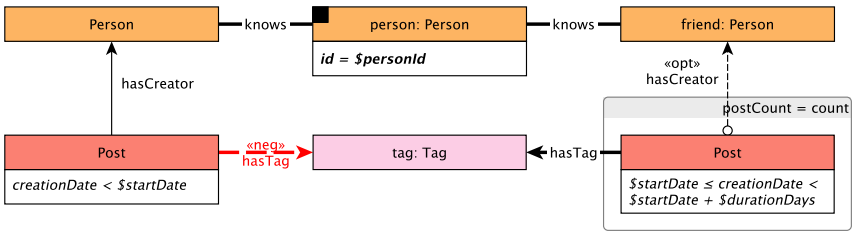
sort

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 `CountryX` and `CountryY`, 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 `CountryX` and `CountryY`.

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 14

query	Interactive / complex / 4
title	New topics
pattern	
desc.	Given a start Person (`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, 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`

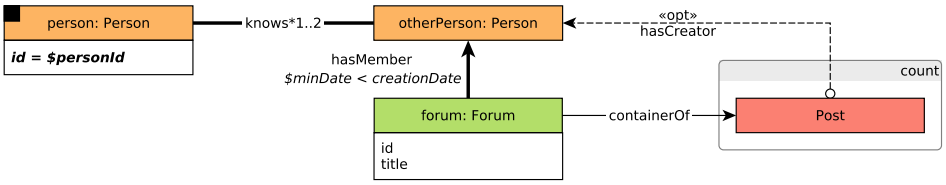
sort

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 14

query	Interactive / complex / 5
title	New groups
pattern	
desc.	Given a start Person, 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`

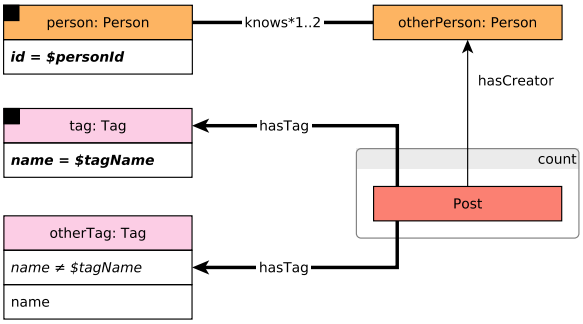
sort

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.

Interactive / complex / 6

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 14

query	Interactive / complex / 6
title	Tag co-occurrence
pattern	
desc.	Given a start Person and some Tag, 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

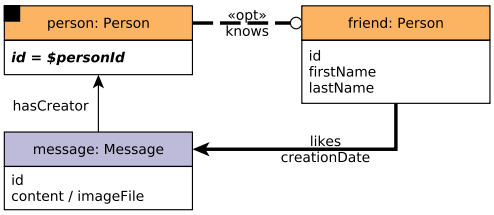
sort

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 14

query	Interactive / complex / 7
title	Recent likers
pattern	
desc.	Given a start Person, 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.

params

1	`personId`	ID

result

1	`friend.id`	ID	R	
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 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

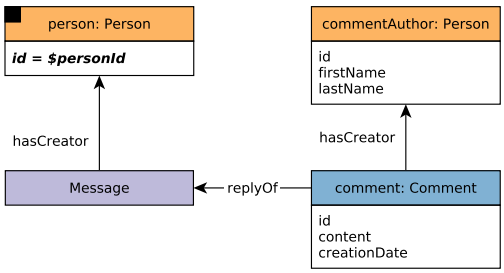
sort

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.

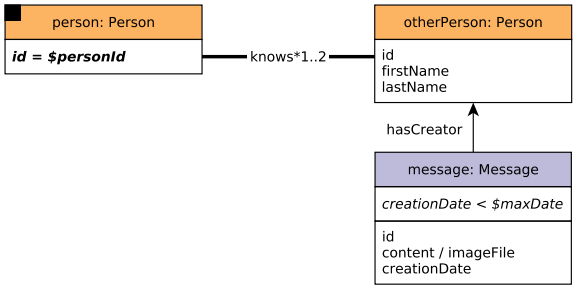
Interactive / complex / 8

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 14

query	Interactive / complex / 8
title	Recent replies
pattern	
desc.	Given a start Person, 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.

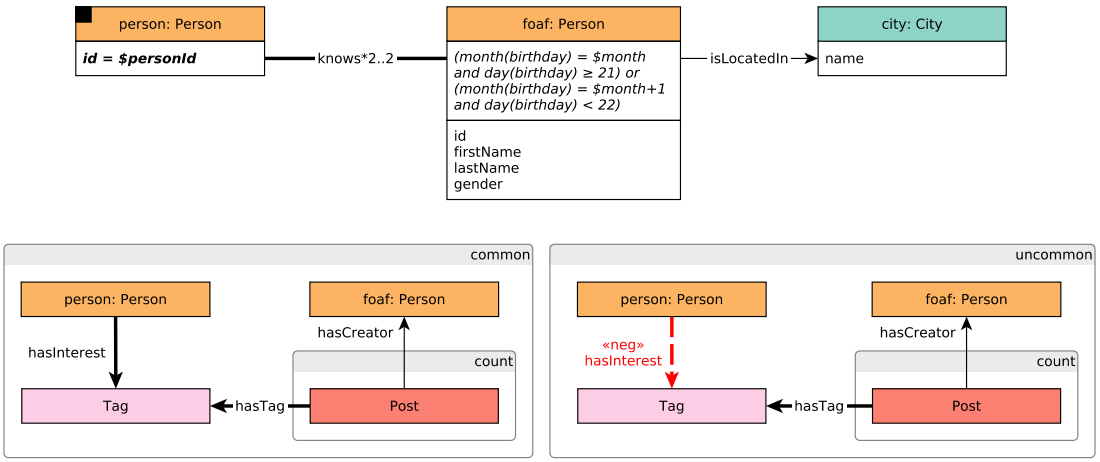
Interactive / complex / 9

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 14

query	Interactive / complex / 9
title	Recent messages by friends or friends of friends
pattern	
desc.	Given a start Person, find the most recent Messages created by that Person's friends or friends of friends (excluding 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 14

query	Interactive / complex / 10
title	Friend recommendation
pattern	
desc.	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	

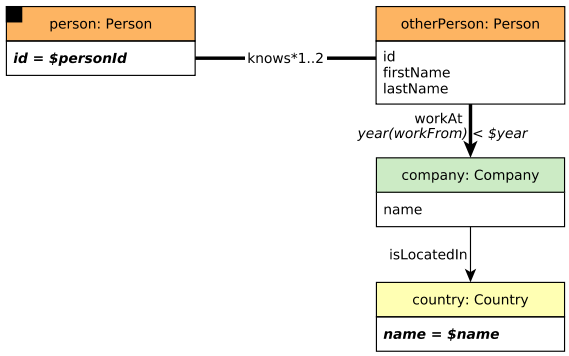
sort

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 14

query	Interactive / complex / 11
title	Job referral
pattern	
desc.	Given a start Person, find that Person's friends and friends of friends (excluding start Person) who started working in some Company in a given Country, before a given date (year).

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	

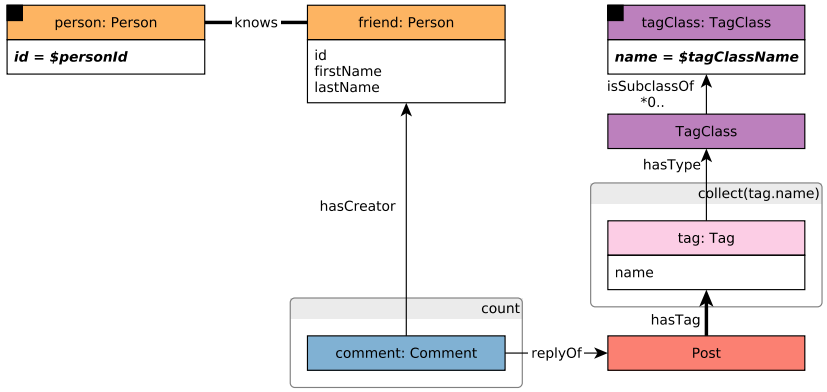
sort

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 14

query	Interactive / complex / 12
title	Expert search
pattern	
desc.	Given a start Person, 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 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

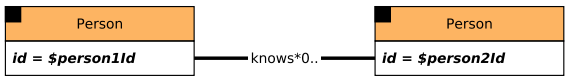
sort

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.

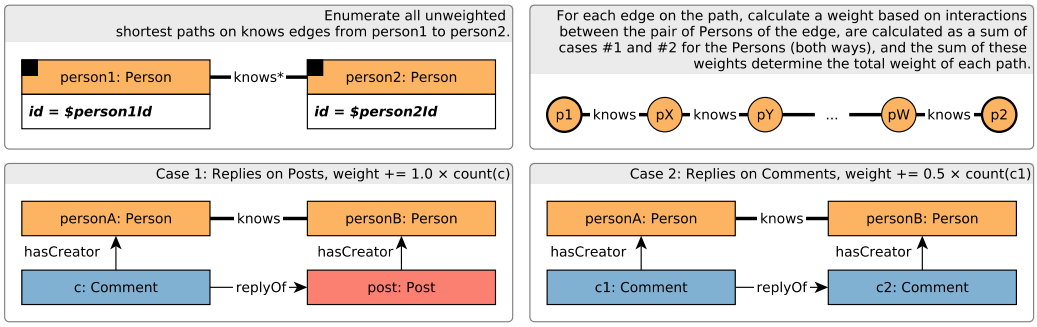
Interactive / complex / 13

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 14

query	Interactive / complex / 13
title	Single shortest path
pattern	
desc.	Given two Persons, find the shortest path between these two Persons in the subgraph induced by the knows relationships. 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 2 person2Id ID
result	1 shortestPathLength 32-bit Integer C
CPs	3.3, 7.2, 7.3, 7.5, 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 / 14

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 14

query	Interactive / complex / 14
title	Trusted connection paths
pattern	
desc.	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. 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

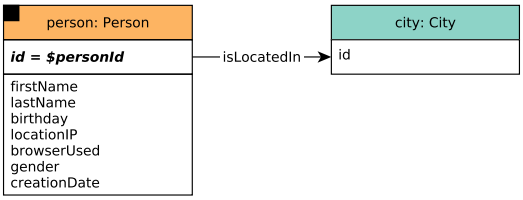
1	pathWeight	↓	The order of paths with the same weight is unspecified

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.

1.2 Short Reads

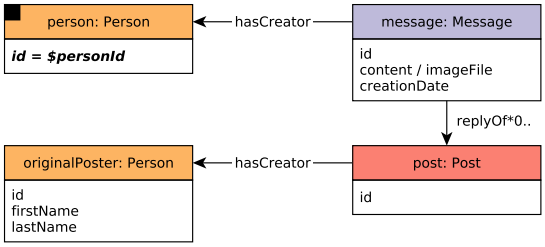
Interactive / short / 1

IS 1
IS 2
IS 3
IS 4
IS 5
IS 6
IS 7

query	Interactive / short / 1
title	Profile of a person
pattern	
desc.	Given a start Person, retrieve their first name, last name, birthday, IP address, browser, and city of residence.
params	1 personId ID
result	1 person.firstName String R 2 person.lastName String R 3 person.birthday Date R 4 person.locationIP String R 5 person.browserUsed String R 6 city.id ID R 7 person.gender String R 8 person.creationDate DateTime R

Interactive / short / 2

IS 1
IS 2
IS 3
IS 4
IS 5
IS 6
IS 7

query	Interactive / short / 2
title	Recent messages of a person
pattern	
desc.	Given a start Person, retrieve the last 10 Messages created by that user. For each Message, return that Message, the original Post in its conversation (`post`), and the author of that Post (`originalPoster`). If any of the Messages is a Post, then the original Post (`post`) will be the same Message, i.e. that Message will appear twice in that result.

params

1	personId	ID	

result

1	message.id	ID	R	
2	message.content or message.imageFile (for photos)	Text	R	
3	message.creationDate	DateTime	R	
4	post.id	ID	R	
5	originalPoster.id	ID	R	
6	originalPoster.firstName	String	R	
7	originalPoster.lastName	String	R	

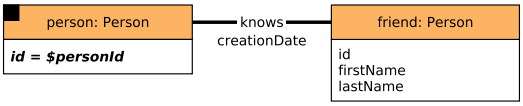
sort

1	message.creationDate	↓	
2	message.id	↓	

limit: 10

Interactive / short / 3

IS 1 IS 2 IS 3 IS 4 IS 5 IS 6 IS 7

query	Interactive / short / 3
title	Friends of a person
pattern	
desc.	Given a start Person, retrieve all of their friends, and the date at which they became friends.

params

1	personId	ID	

result

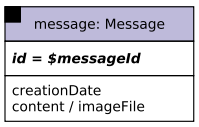
1	friend.id	ID	R	
2	friend.firstName	String	R	
3	friend.lastName	String	R	
4	knows.creationDate	DateTime	R	

sort

1	knows.creationDate	↓	
2	friend.id	↑	

Interactive / short / 4

IS 1 IS 2 IS 3 IS 4 IS 5 IS 6 IS 7

query	Interactive / short / 4
title	Content of a message
pattern	
desc.	Given a Message, retrieve its content and creation date.

params

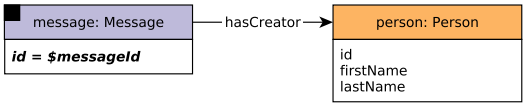
1	messageId	ID	

result

1	message.creationDate	DateTime	R	messageCreationDate
2	message.content or message.imageFile (for photos)	Text	R	messageContent

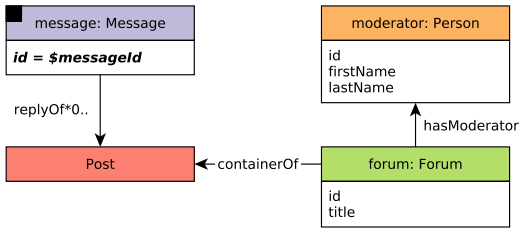
Interactive / short / 5

IS 1 IS 2 IS 3 IS 4 IS 5 IS 6 IS 7

query	Interactive / short / 5
title	Creator of a message
pattern	
desc.	Given a Message, retrieve its author.
params	1 messageId ID
result	1 person.id ID R 2 person.firstName String R 3 person.lastName String R

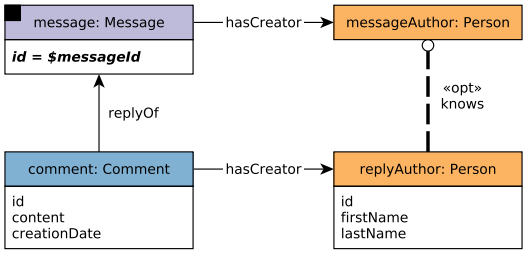
Interactive / short / 6

IS 1 IS 2 IS 3 IS 4 IS 5 IS 6 IS 7

query	Interactive / short / 6
title	Forum of a message
pattern	
desc.	Given a Message, retrieve the Forum that contains it and the Person that moderates that Forum. Since Comments are not directly contained in Forums, for Comments, return the Forum containing the original Post in the thread which the Comment is replying to.
params	1 messageId ID
result	1 forum.id ID R 2 forum.title Long String R 3 moderator.id ID R 4 moderator.firstName String R 5 moderator.lastName String R

Interactive / short / 7

IS 1
IS 2
IS 3
IS 4
IS 5
IS 6
IS 7

query	Interactive / short / 7
title	Replies of a message
pattern	
desc.	Given a Message, retrieve the (1-hop) Comments that reply to it. In addition, return a boolean flag `knows` indicating if the author of the reply (`replyAuthor`) knows the author of the original message (`messageAuthor`). If author is same as original author, return `False` for `knows` flag.

params

1	messageId	ID	

result

1	comment.id	ID	R	
2	comment.content	Text	R	
3	comment.creationDate	DateTime	R	
4	replyAuthor.id	ID	R	
5	replyAuthor.firstName	String	R	
6	replyAuthor.lastName	String	R	
7	knows	Boolean	C	`True` if the knows edge exists between the `replyAuthor` and the `messageAuthor` nodes, `False` otherwise (including the case when the two nodes are the same)

sort

1	comment.creationDate	↓
2	replyAuthor.id	↑

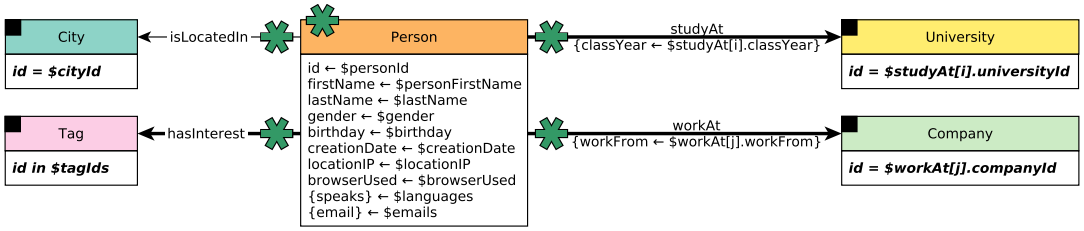
1.3 Inserts (Formerly: Updates)

Each insert query inserts

1. either a single node of a certain type, along with its edges to other existing nodes
2. or a single edge of a certain type between two existing nodes.

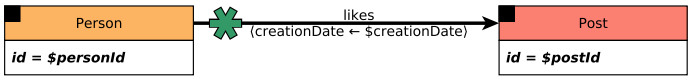
In versions 0.3.x, these operations were called “Interactive updates”. From 0.4.0 onwards, these are called “Interactive inserts”.

Interactive / insert / 1

INS 1	query	Interactive / insert / 1
INS 2	title	Add person
INS 3	pattern	
INS 4	desc.	Add a Person *node*, connected to the network by 4 possible *edge* types.
INS 5	params	
INS 6	CPs	9.1, 9.2

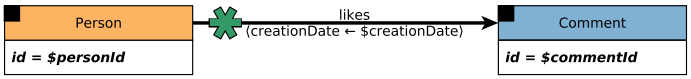
1	personId	ID	
2	personFirstName	String	
3	personLastName	String	
4	gender	String	
5	birthday	Date	
6	creationDate	DateTime	
7	locationIP	String	
8	browserUsed	String	
9	cityId	ID	
10	languages	{String}	
11	emails	{Long String}	
12	tagIds	{ID}	
13	studyAt	{<ID, 32-bit Integer>}	{‹universityId, classYear›}
14	workAt	{<ID, 32-bit Integer>}	{‹companyId, workFrom›}

Interactive / insert / 2

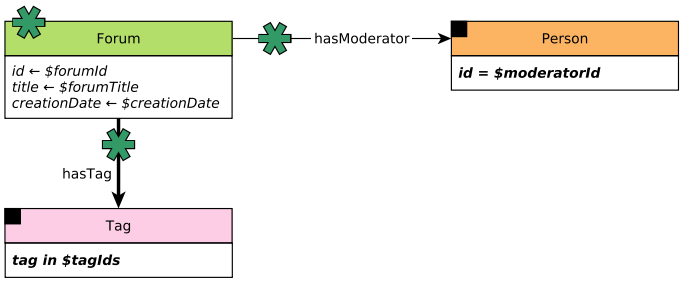
INS 1	query	Interactive / insert / 2
INS 2	title	Add like to post
INS 3	pattern	
INS 4	desc.	Add a likes *edge* to a Post.
INS 5	params	
INS 6	CPs	9.2

1	personId	ID	
2	postId	ID	
3	creationDate	DateTime	

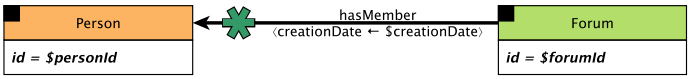
Interactive / insert / 3

INS 1	query	Interactive / insert / 3
INS 2	title	Add like to comment
INS 3	pattern	
INS 4	desc.	Add a likes *edge* to a Comment.
INS 5	params	1 personId ID 2 commentId ID 3 creationDate DateTime
INS 6	CPs	9.2

Interactive / insert / 4

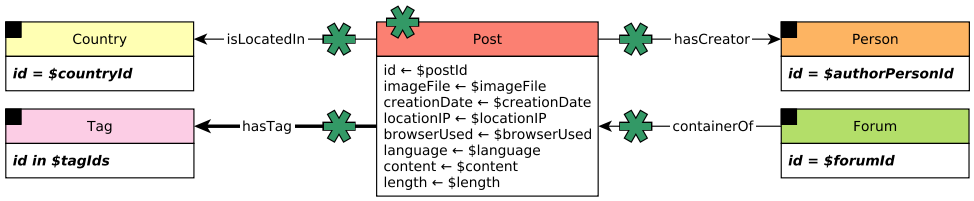
INS 1	query	Interactive / insert / 4
INS 2	title	Add forum
INS 3	pattern	
INS 4	desc.	Add a Forum *node*, connected to the network by 2 possible *edge* types.
INS 5	params	1 forumId ID 2 forumTitle Long String 3 creationDate DateTime 4 moderatorId ID 5 tagIds {ID}
INS 6	CPs	9.1, 9.2

Interactive / insert / 5

INS 1	query	Interactive / insert / 5
INS 2	title	Add forum membership
INS 3	pattern	
INS 4	desc.	Add a Forum membership *edge* (hasMember) to a Person.
INS 5	params	
INS 6	CPs	9.1, 9.2
INS 7		
INS 8		

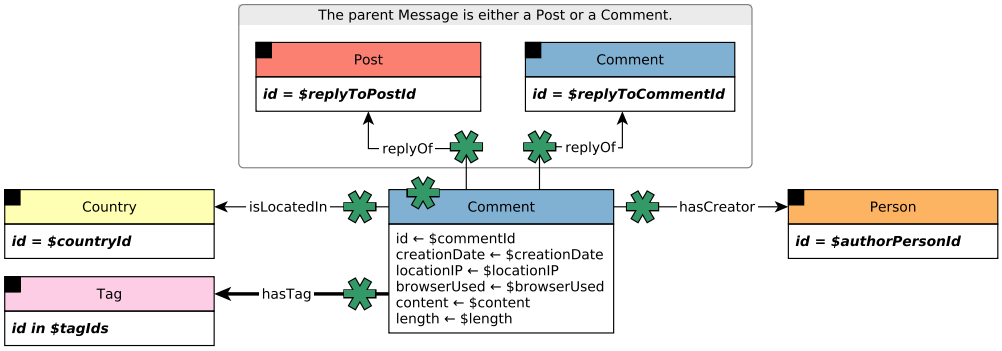
1	personId	ID	
2	forumId	ID	
3	creationDate	DateTime	

Interactive / insert / 6

INS 1	query	Interactive / insert / 6
INS 2	title	Add post
INS 3	pattern	
INS 4	desc.	Add a Post *node* to the social network connected by 4 possible *edge* types (hasCreator, containerOf, isLocatedIn, hasTag).
INS 5	params	
INS 6	CPs	9.1, 9.2
INS 7		
INS 8		

1	postId	ID	
2	imageFile	String	
3	creationDate	DateTime	
4	locationIP	String	
5	browserUsed	String	
6	language	String	
7	content	Text	
8	length	32-bit Integer	
9	authorPersonId	ID	
10	forumId	ID	
11	countryId	ID	
12	tagIds	{ID}	

Interactive / insert / 7

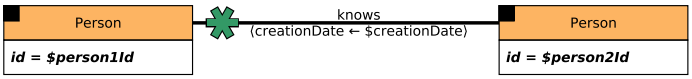
query	Interactive / insert / 7
title	Add comment
pattern	
desc.	Add a Comment *node* replying to a Post/Comment, connected to the network by 4 possible *edge* types (replyOf, hasCreator, isLocatedIn, hasTag).

params

1	commentId	ID	
2	creationDate	DateTime	
3	locationIP	String	
4	browserUsed	String	
5	content	Text	
6	length	32-bit Integer	
7	authorPersonId	ID	
8	countryId	ID	
9	replyToPostId	ID	−1 if the Comment is a reply of a Comment
10	replyToCommentId	ID	−1 if the Comment is a reply of a Post
11	tagIds	{ID}	

CPs	9.1, 9.2

Interactive / insert / 8

query	Interactive / insert / 8
title	Add friendship
pattern	
desc.	Add a friendship *edge* (knows) between two Persons.

params

1	person1Id	ID	
2	person2Id	ID	
3	creationDate	DateTime	

CPs	9.2

Bibliography

[1] Orri Erling et al. “The LDBC Social Network Benchmark: Interactive Workload”. In: *SIGMOD*. 2015, pp. 619–630. DOI: 10.1145/2723372.2742786.