1 Interactive v2 Workload

This chapter is based on the TPCTC 2023 paper "The LDBC Social Network Benchmark Interactive Workload v2: A Transactional Graph Query Benchmark with Deep Delete Operations" [7], co-authored by several members of the SNB task force.

Work-in-Progress

The Interactive v2 workload is currently work-in-progress. As of January 2024, commissioning audits for this workload is not yet possible.

Related Software Components

- Datagen (Spark-based): `https://github.com/ldbc/ldbc_snb_datagen_spark`
- Driver: `https://github.com/ldbc/ldbc_snb_interactive_v2_driver`
- Reference implementations: `https://github.com/ldbc/ldbc_snb_interactive_v2_impls`

1.1 Overview

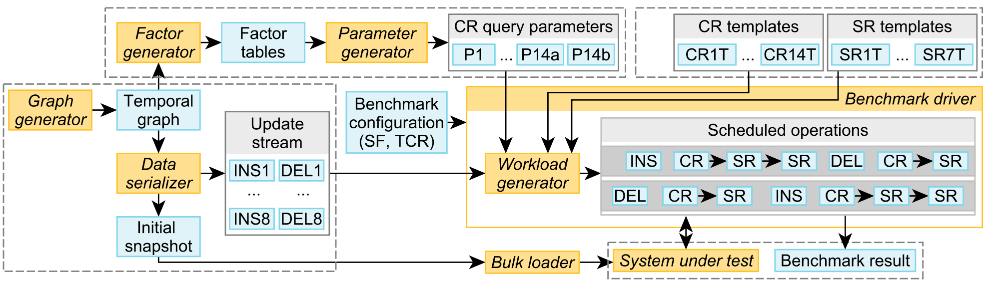


Figure 1.1: Components and workflow of the Interactive v2 workload. The corresponding sections are shown in green circles §. Legend: *Software component* Data artifact

1.2 Operations

The LDBC SNB Interactive v2 workload uses four types of operations. There are 14 complex and 7 short read queries. Update operations include 8 inserts and, newly introduced in the Interactive v2 workload, 8 deletes. The workload mix consists of approximately 8% complex read, 72% short read, 20% insert, and 0.2% delete operations. The complex reads and the short reads are identical to the ones in Interactive v1, except for query 14, which was replaced to cover the *Cheapest path-finding* choke point.[1]

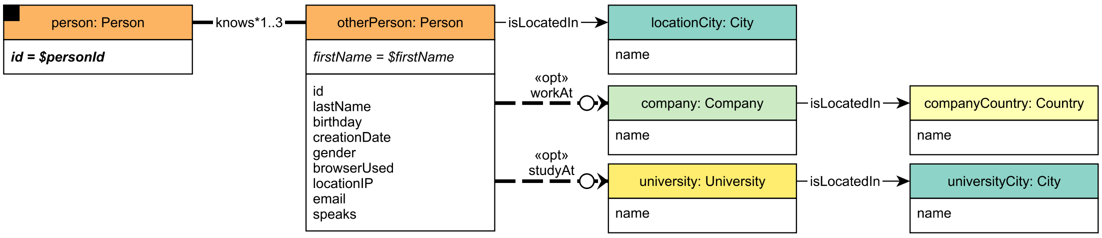
Cheapest path-finding While we strived to keep the changes to the queries minimal, we replaced Q14 due to two reasons. First, we found the original query in Interactive v1 to be ill-suited to the workload as it required the enumeration of *all shortest paths* between two Persons, which can be prohibitively expensive on large scale factors. Second, we introduced a new choke point, CP-7.6 *Cheapest path-finding,* a key computational kernel and a language opportunity for GQL [2]. Therefore, we changed Q14 to use *cheapest paths* instead of *all shortest paths*.

[1] The term *shortest paths* refers to the problem of finding *unweighted shortest paths*, which can be computed with BFS. The term *cheapest paths* refers to the *weighted shortest paths* problem, which can be solved using e.g. Dijkstra's algorithm.

1.2.1 Complex Reads

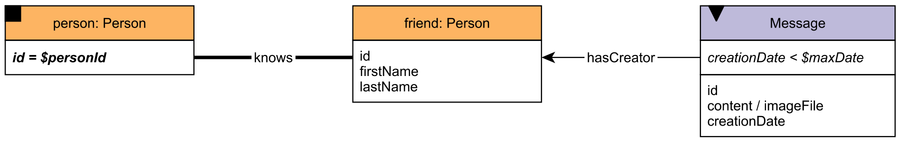
Interactive / complex / 1

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 14v1
IC 14v2

<table>
<tr><td>query</td><td>Interactive / complex / 1</td></tr>
<tr><td>title</td><td>Transitive friends with a certain name</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>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.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$firstName</td><td>String</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>otherPerson.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>otherPerson.lastName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td>distanceFromPerson</td><td>32-bit Integer</td><td>C</td><td></td></tr>
<tr><td>4</td><td>otherPerson.birthday</td><td>Date</td><td>R</td><td></td></tr>
<tr><td>5</td><td>otherPerson.creationDate</td><td>DateTime</td><td>R</td><td></td></tr>
<tr><td>6</td><td>otherPerson.gender</td><td>String</td><td>R</td><td></td></tr>
<tr><td>7</td><td>otherPerson.browserUsed</td><td>String</td><td>R</td><td></td></tr>
<tr><td>8</td><td>otherPerson.locationIP</td><td>String</td><td>R</td><td></td></tr>
<tr><td>9</td><td>otherPerson.email</td><td>{Long String}</td><td>R</td><td></td></tr>
<tr><td>10</td><td>otherPerson.speaks</td><td>{String}</td><td>R</td><td></td></tr>
<tr><td>11</td><td>locationCity.name</td><td>String</td><td>R</td><td></td></tr>
<tr><td>12</td><td>universities</td><td>{<String, 32-bit Integer, String>}</td><td>A</td><td>{<university.name, studyAt.classYear, universityCity.name>}</td></tr>
<tr><td>13</td><td>companies</td><td>{<String, 32-bit Integer, String>}</td><td>A</td><td>{<company.name, workAt.workFrom, companyCountry.name>}</td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>distanceFromPerson</td><td>↑</td><td></td></tr>
<tr><td>2</td><td>otherPerson.lastName</td><td>↑</td><td></td></tr>
<tr><td>3</td><td>otherPerson.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>2.1, 5.3, 8.2</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

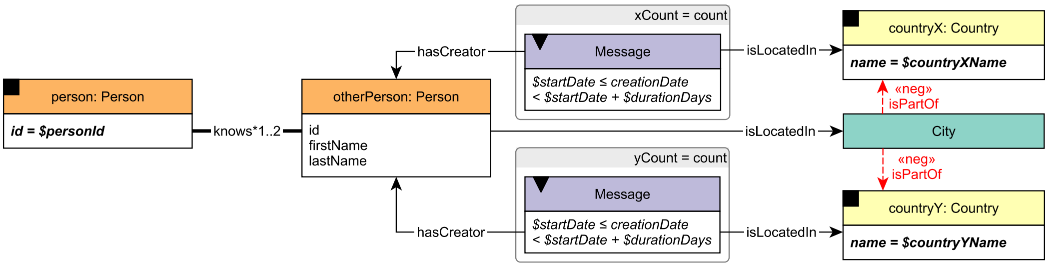
IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

Interactive / complex / 2

<table>
<tr><td>query</td><td>Interactive / complex / 2</td></tr>
<tr><td>title</td><td>Recent messages by your friends</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>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).</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$maxDate</td><td>Date</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>friend.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>friend.firstName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td>friend.lastName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>4</td><td>message.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>5</td><td>message.content or message.imageFile (for photos)</td><td>Text</td><td>R</td><td></td></tr>
<tr><td>6</td><td>message.creationDate</td><td>DateTime</td><td>R</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>message.creationDate</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>message.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>1.1, 2.2, 2.3, 3.2, 8.5</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

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 14v1
IC 14v2

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

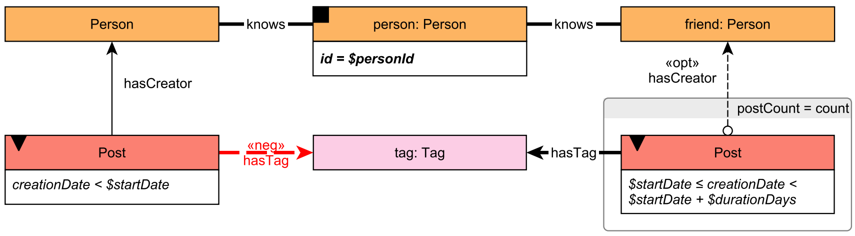
sort

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`.

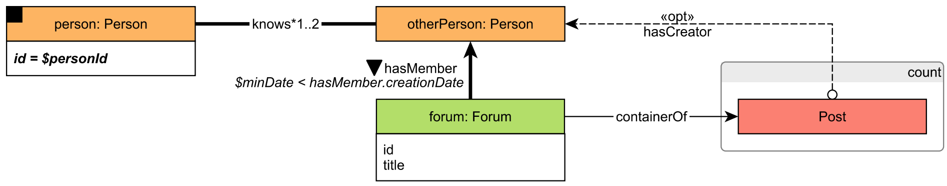
Interactive / complex / 4

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.

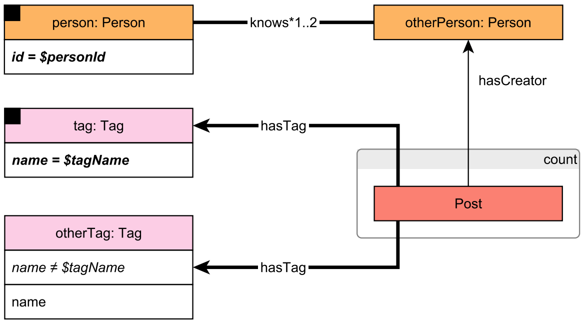
Interactive / complex / 5

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	
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.

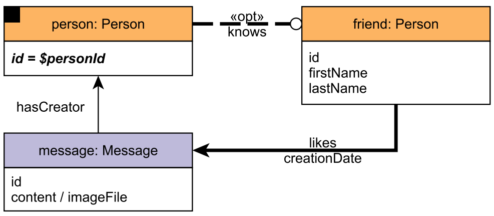
Interactive / complex / 6

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 14v1
IC 14v2

query	Interactive / complex / 6
title	Tag co-occurrence
pattern	
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.

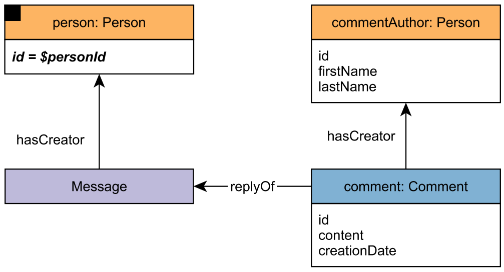
Interactive / complex / 7

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

<table>
<tr><td>query</td><td>Interactive / complex / 7</td></tr>
<tr><td>title</td><td>Recent likers</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Given a start Person with ID $personId, find the most recent likes on any of start Person's Messages. Find Persons that liked (likes edge) any of start Person's Messages, the Messages they liked most recently, the creation date of that like, and the latency in minutes (minutesLatency) between creation of Messages and like. Additionally, for each Person found return a flag indicating (isNew) whether the liker is a friend of start Person. In case that a Person liked multiple Messages at the same time, return the Message with lowest identifier.
Validation rule: Depending on whether the system-under-test supports leap seconds or uses UTC-SLS (UTC with Smoothed Leap Seconds), a difference of 1 minute can occur between the minutesLatency results of two correct implementations when the time interval includes June 30, 2012, when there was a leap second. Therefore, the minutesLatency value is validated using a tolerance of 1 minute.</td></tr>
<tr><td>params</td><td><table><tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr></table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>friend.id</td><td>ID</td><td>R</td><td>friend.id = personId is allowed</td></tr>
<tr><td>2</td><td>friend.firstName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td>friend.lastName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>4</td><td>likes.creationDate</td><td>DateTime</td><td>R</td><td></td></tr>
<tr><td>5</td><td>message.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>6</td><td>message.content or message.imageFile (for photos)</td><td>Text</td><td>R</td><td></td></tr>
<tr><td>7</td><td>minutesLatency</td><td>32-bit Integer</td><td>C</td><td>Duration between the creation of the Message and the creation of the like, in minutes.</td></tr>
<tr><td>8</td><td>isNew</td><td>Boolean</td><td>C</td><td>False if person and friend know each other, True otherwise</td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>likes.creationDate</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>friend.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>2.2, 2.3, 3.3, 5.1, 8.1, 8.3</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

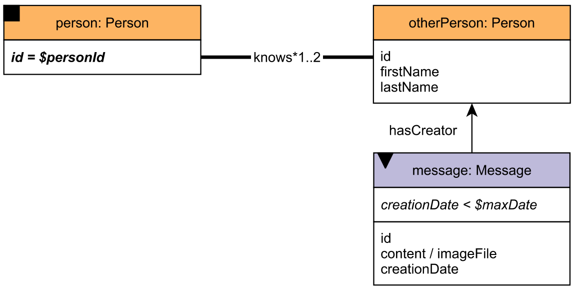
Interactive / complex / 8

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	
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.

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 / 9

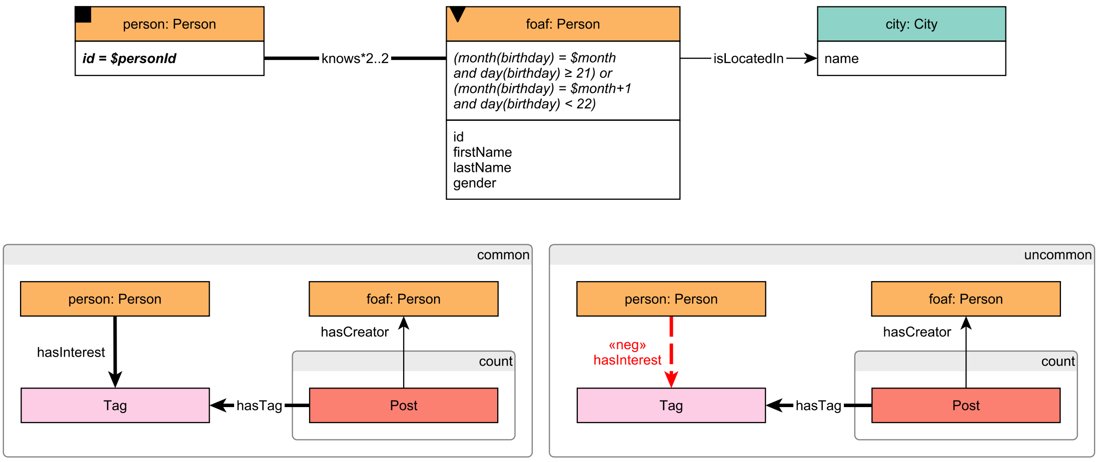
<table>
<tr><td>query</td><td>Interactive / complex / 9</td></tr>
<tr><td>title</td><td>Recent messages by friends or friends of friends</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Given a start Person with ID $personId, find the most recent Messages created by that Person's friends or friends of friends (excluding the start Person). Only consider Messages created before the given $maxDate (excluding that day).</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$maxDate</td><td>Date</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>otherPerson.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>otherPerson.firstName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td>otherPerson.lastName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>4</td><td>message.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>5</td><td>message.content or message.imageFile (for photos)</td><td>Text</td><td>R</td><td></td></tr>
<tr><td>6</td><td>message.creationDate</td><td>DateTime</td><td>R</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>message.creationDate</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>message.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>1.1, 1.2, 2.2, 2.3, 3.2, 3.3, 8.5</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

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	

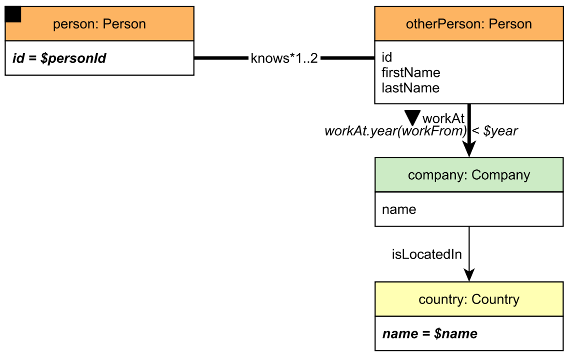
sort

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.

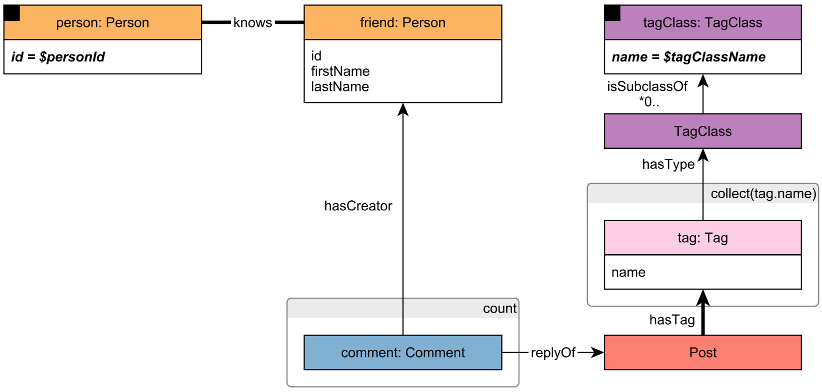
Interactive / complex / 11

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

<table>
<tr><td>query</td><td>Interactive / complex / 11</td></tr>
<tr><td>title</td><td>Job referral</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Given a start Person with ID <code>$personId</code>, find that Person's friends and friends of friends (excluding start Person) who started working in some Company in a given Country with name <code>$countryName</code>, before a given date (<code>$workFromYear</code>).</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td><code>$personId</code></td><td>ID</td><td></td></tr>
<tr><td>2</td><td><code>$countryName</code></td><td>String</td><td></td></tr>
<tr><td>3</td><td><code>$workFromYear</code></td><td>32-bit Integer</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td><code>otherPerson.id</code></td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td><code>otherPerson.firstName</code></td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td><code>otherPerson.lastName</code></td><td>String</td><td>R</td><td></td></tr>
<tr><td>4</td><td><code>company.name</code></td><td>String</td><td>R</td><td></td></tr>
<tr><td>5</td><td><code>workAt.workFrom</code></td><td>32-bit Integer</td><td>R</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td><code>workAt.workFrom</code></td><td>↑</td><td></td></tr>
<tr><td>2</td><td><code>otherPerson.id</code></td><td>↑</td><td></td></tr>
<tr><td>3</td><td><code>company.name</code></td><td>↓</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>10</td></tr>
<tr><td>CPs</td><td>1.3, 2.3, 2.4, 3.3, 4.2</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

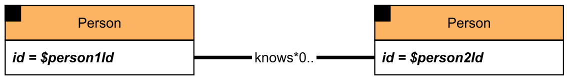
IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14v1
IC 14v2

Interactive / complex / 12

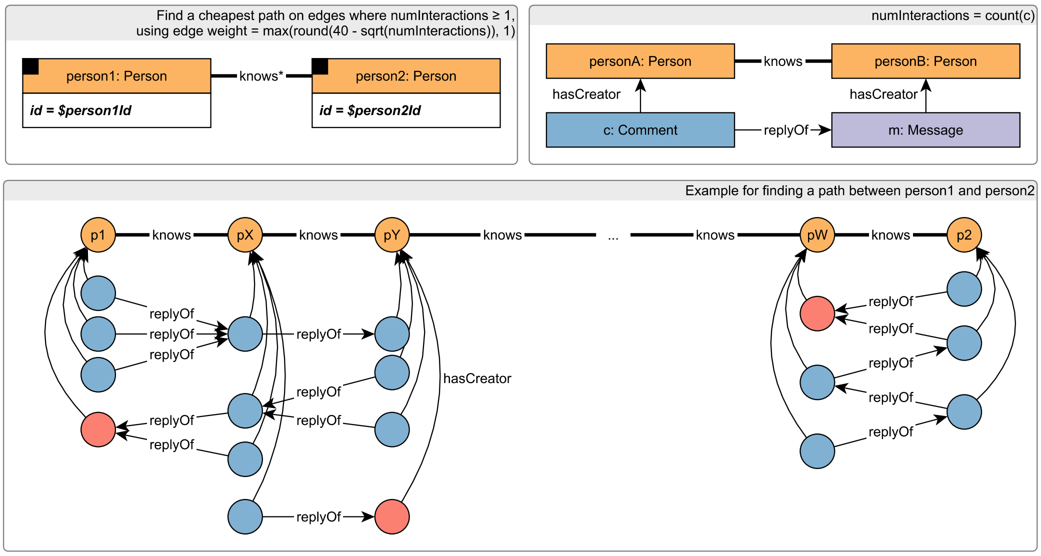
<table>
<tr><td>query</td><td>Interactive / complex / 12</td></tr>
<tr><td>title</td><td>Expert search</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Given a start Person with ID $personId, find the Comments that this Person's friends made in reply to Posts, considering only 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 $tagClassName or in a descendent of that TagClass. Count the number of these reply Comments, and collect the Tags that were attached to the Posts they replied to, but only collect Tags with the given TagClass or with a descendant of that TagClass. Return Persons with at least one reply, the reply count, and the collection of Tags.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$tagClassName</td><td>Long String</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td>friend.id</td><td>ID</td><td>R</td><td></td></tr>
<tr><td>2</td><td>friend.firstName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>3</td><td>friend.lastName</td><td>String</td><td>R</td><td></td></tr>
<tr><td>4</td><td>tagNames</td><td>{Long String}</td><td>A</td><td></td></tr>
<tr><td>5</td><td>replyCount</td><td>32-bit Integer</td><td>A</td><td></td></tr>
</table></td></tr>
<tr><td>sort</td><td><table>
<tr><td>1</td><td>replyCount</td><td>↓</td><td></td></tr>
<tr><td>2</td><td>friend.id</td><td>↑</td><td></td></tr>
</table></td></tr>
<tr><td>limit</td><td>20</td></tr>
<tr><td>CPs</td><td>3.3, 7.2, 7.3, 8.2</td></tr>
<tr><td>relevance</td><td>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.</td></tr>
</table>

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	
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.

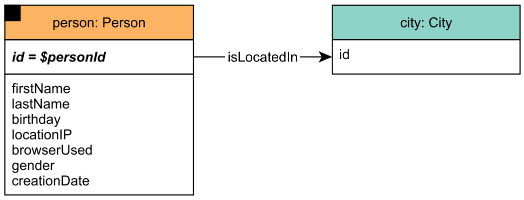
Interactive / complex / 14v2

<table>
<tr><td>query</td><td>Interactive / complex / 14v2</td></tr>
<tr><td>title</td><td>Trusted connection paths (v2)</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>This query is used in SNB Interactive v2.
Find a cheapest path between two given Persons with IDs <code>$person1Id</code> and <code>$person2Id</code> in the interaction subgraph. If there are multiple cheapest paths, any of them can be returned. Do not return any rows if there is no path between the Persons. The interaction subgraph is based on a projection of the Person-knows-Person graph. In this projection, only those knows edges are kept whose endpoint Persons have at least one interaction between them. An interaction is defined as a direct reply Comment (by one of the Persons) to a Message (by the other Person). The weights are defined as: $\max(\text{round}(40 - \sqrt{numInteractions}), 1)$
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-deterministic 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.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td><code>$person1Id</code></td><td>ID</td><td><code>(b)</code> There are no paths between the two Persons
<code>(b)</code> There is a 4-hop path between the two Persons</td></tr>
<tr><td>2</td><td><code>$person2Id</code></td><td>ID</td><td></td></tr>
</table></td></tr>
<tr><td>result</td><td><table>
<tr><td>1</td><td><code>personIdsInPath</code></td><td>[ID]</td><td>C</td><td>Identifiers representing an ordered sequence of the Persons in the path</td></tr>
<tr><td>2</td><td><code>pathWeight</code></td><td>64-bit Integer</td><td>C</td><td></td></tr>
</table></td></tr>
<tr><td>CPs</td><td>3.3, 5.3, 7.6, 7.7, 7.8, 8.1, 8.2, 8.3, 8.6</td></tr>
<tr><td>relevance</td><td>This query tests the performance of cheapest path (weighted shortest path) computation.</td></tr>
</table>

1.2.2 Short Reads

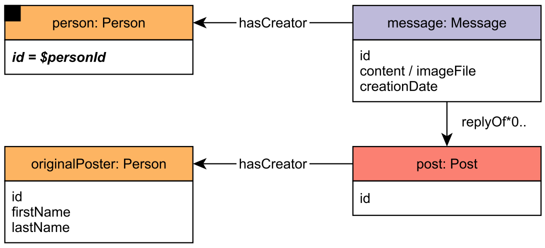
Interactive / short / 1

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

query	Interactive / short / 1			
title	Profile of a person			
pattern				
description	Given a start Person with ID `$personId`, retrieve their first name, last name, birthday, IP address, browser, and city of residence.			
params	1	`$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

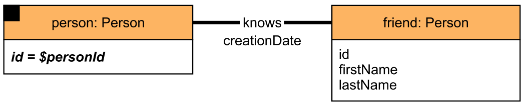
IS 1
IS 2
IS 3
IS 4
IS 5
IS 6
IS 7

Interactive / short / 2

query	Interactive / short / 2
title	Recent messages of a person
pattern	
description	Given a start Person with ID `$personId`, retrieve the last 10 Messages created by that user. For each Message, return that Message, the original Post in its conversation (`post`), and the author of that Post (`originalPoster`). If any of the Messages is a Post, then the original Post (`post`) will be the same Message, i.e. that Message will appear twice in that result.
params	1 | `$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	
description	Given a start Person with ID `$personId`, retrieve all of their friends, and the date at which they became friends.

params

1	$personId	ID	

result

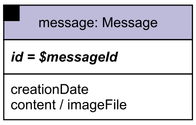
1	friend.id	ID	R	
2	friend.firstName	String	R	
3	friend.lastName	String	R	
4	knows.creationDate	DateTime	R	

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	
description	Given a Message with ID `$messageId`, retrieve its content and creation date.

params

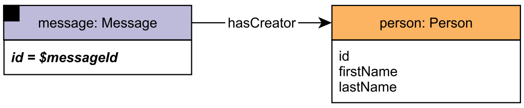
1	$messageId	ID	

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	
description	Given a Message with ID `$messageId`, retrieve its author.

params

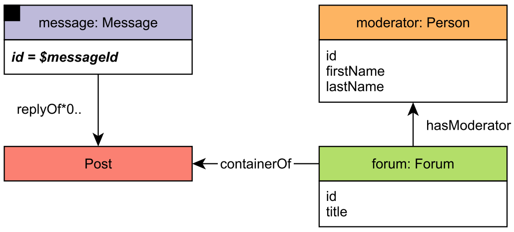
1	`$messageId`	ID

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	
description	Given a Message with ID `$messageId`, 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

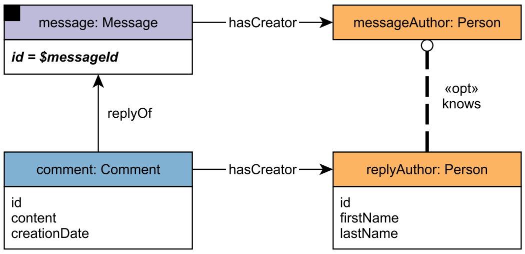
1	`$messageId`	ID

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	
description	Given a `Message` with ID `$messageId`, 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)

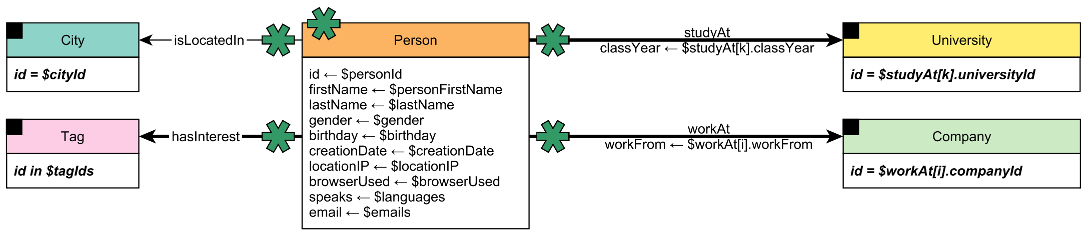
sort

1	`comment.creationDate`	↓
2	`replyAuthor.id`	↑

1.2.3 Insert Operations

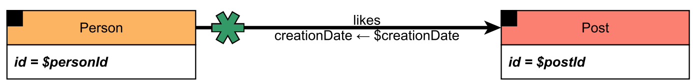
Updates / insert / 1

INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 1</td></tr>
<tr><td>title</td><td>Add person</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a Person node, connected to the network by 4 possible edge types.</td></tr>
<tr><td>params</td><td>
<table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$personFirstName</td><td>String</td><td></td></tr>
<tr><td>3</td><td>$personLastName</td><td>String</td><td></td></tr>
<tr><td>4</td><td>$gender</td><td>String</td><td></td></tr>
<tr><td>5</td><td>$birthday</td><td>Date</td><td></td></tr>
<tr><td>6</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
<tr><td>7</td><td>$locationIP</td><td>String</td><td></td></tr>
<tr><td>8</td><td>$browserUsed</td><td>String</td><td></td></tr>
<tr><td>9</td><td>$cityId</td><td>ID</td><td></td></tr>
<tr><td>10</td><td>$languages</td><td>{String}</td><td></td></tr>
<tr><td>11</td><td>$emails</td><td>{Long String}</td><td></td></tr>
<tr><td>12</td><td>$tagIds</td><td>{ID}</td><td></td></tr>
<tr><td>13</td><td>$studyAt</td><td>{<ID, 32-bit Integer>}</td><td>{<universityId, classYear>}</td></tr>
<tr><td>14</td><td>$workAt</td><td>{<ID, 32-bit Integer>}</td><td>{<companyId, workFrom>}</td></tr>
</table>
</td></tr>
<tr><td>CPs</td><td>9.1, 9.2</td></tr>
</table>

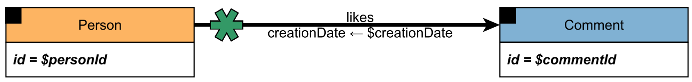
Updates / insert / 2

INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 2</td></tr>
<tr><td>title</td><td>Add like to post</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a likes edge to a Post.</td></tr>
<tr><td>params</td><td>
<table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$postId</td><td>ID</td><td></td></tr>
<tr><td>3</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
</table>
</td></tr>
<tr><td>CPs</td><td>9.2</td></tr>
</table>

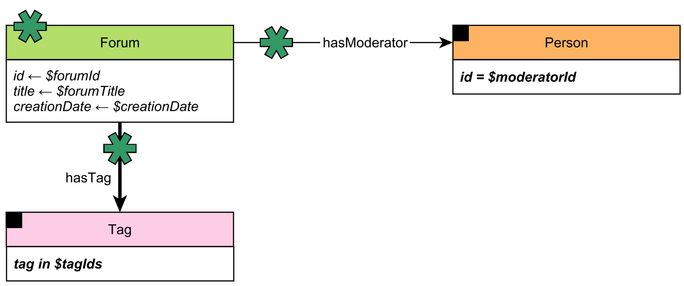
Updates / insert / 3

INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 3</td></tr>
<tr><td>title</td><td>Add like to comment</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a likes edge to a Comment.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$personId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$commentId</td><td>ID</td><td></td></tr>
<tr><td>3</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
</table></td></tr>
<tr><td>CPs</td><td>9.2</td></tr>
</table>

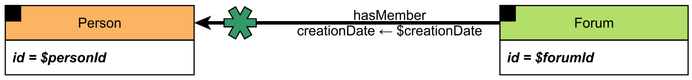
Updates / insert / 4

INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 4</td></tr>
<tr><td>title</td><td>Add forum</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a Forum node, connected to the network by 2 possible edge types.</td></tr>
<tr><td>params</td><td><table>
<tr><td>1</td><td>$forumId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$forumTitle</td><td>Long String</td><td></td></tr>
<tr><td>3</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
<tr><td>4</td><td>$moderatorId</td><td>ID</td><td></td></tr>
<tr><td>5</td><td>$tagIds</td><td>{ID}</td><td></td></tr>
</table></td></tr>
<tr><td>CPs</td><td>9.1, 9.2</td></tr>
</table>

Updates / insert / 5

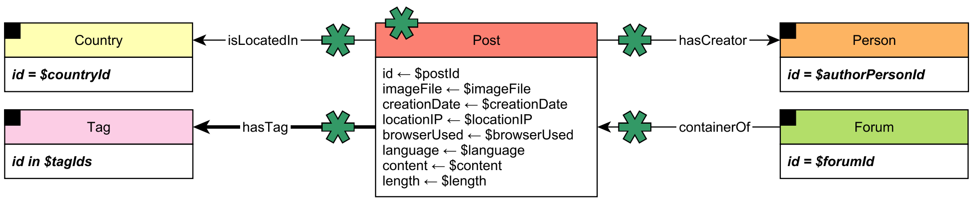
INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

query	Updates / insert / 5
title	Add forum membership
pattern	
description	Add a Forum membership *edge* (hasMember) to a Person.
params	see below
CPs	9.1, 9.2

1	`$personId`	ID	
2	`$forumId`	ID	
3	`$creationDate`	DateTime	

Updates / insert / 6

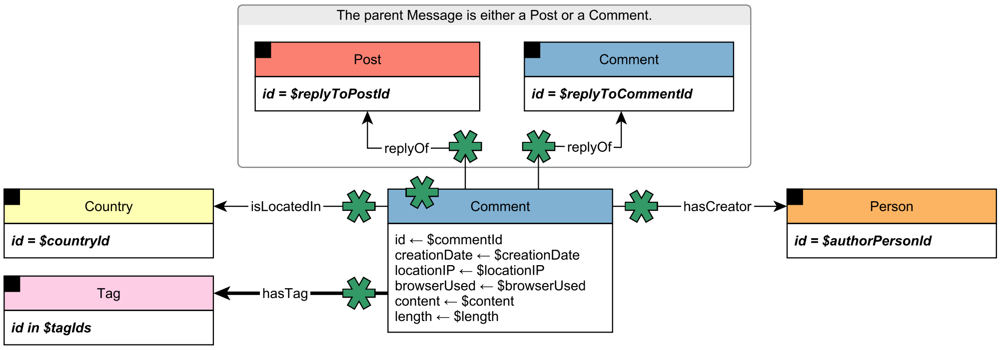
INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

query	Updates / insert / 6
title	Add post
pattern	
description	Add a Post *node* connected to the network by 4 possible *edge* types (hasCreator, containerOf, isLocatedIn, hasTag).
params	see below
CPs	9.1, 9.2

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}	

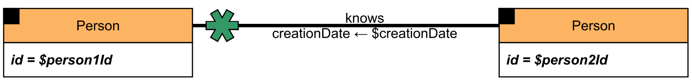
Updates / insert / 7

INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 7</td></tr>
<tr><td>title</td><td>Add comment</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a Comment node replying to a Post/Comment, connected to the network by 4 possible edge types (replyOf, hasCreator, isLocatedIn, hasTag).</td></tr>
<tr><td>params</td><td>
<table>
<tr><td>1</td><td>$commentId</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
<tr><td>3</td><td>$locationIP</td><td>String</td><td></td></tr>
<tr><td>4</td><td>$browserUsed</td><td>String</td><td></td></tr>
<tr><td>5</td><td>$content</td><td>Text</td><td></td></tr>
<tr><td>6</td><td>$length</td><td>32-bit Integer</td><td></td></tr>
<tr><td>7</td><td>$authorPersonId</td><td>ID</td><td></td></tr>
<tr><td>8</td><td>$countryId</td><td>ID</td><td></td></tr>
<tr><td>9</td><td>$replyToPostId</td><td>ID</td><td>old version: -1 if the Comment is a reply of a Comment; new version: null if the Comment is a reply of a Post</td></tr>
<tr><td>10</td><td>$replyToCommentId</td><td>ID</td><td>old version: -1 if the Comment is a reply of a Post; new version: null if the Comment is a reply of a Post</td></tr>
<tr><td>11</td><td>$tagIds</td><td>{ID}</td><td></td></tr>
</table>
</td></tr>
<tr><td>CPs</td><td>9.1, 9.2</td></tr>
</table>

Updates / insert / 8

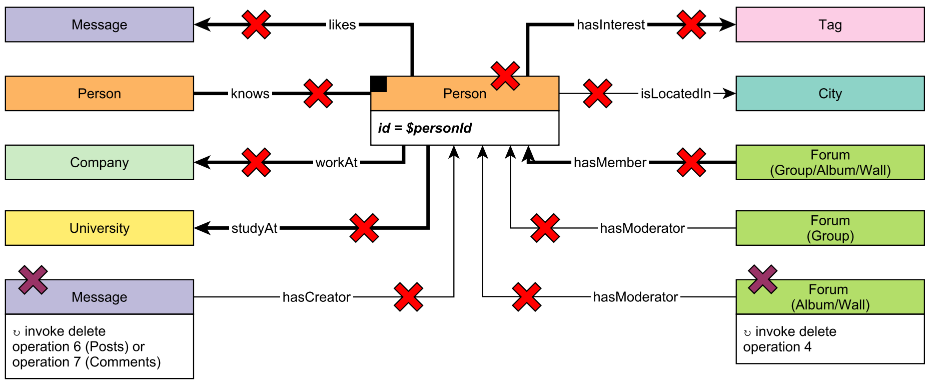
INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

<table>
<tr><td>query</td><td>Updates / insert / 8</td></tr>
<tr><td>title</td><td>Add friendship</td></tr>
<tr><td>pattern</td><td>
</td></tr>
<tr><td>description</td><td>Add a friendship edge (knows) between two Persons.</td></tr>
<tr><td>params</td><td>
<table>
<tr><td>1</td><td>$person1Id</td><td>ID</td><td></td></tr>
<tr><td>2</td><td>$person2Id</td><td>ID</td><td></td></tr>
<tr><td>3</td><td>$creationDate</td><td>DateTime</td><td></td></tr>
</table>
</td></tr>
<tr><td>CPs</td><td>9.2</td></tr>
</table>

1.2.4 Delete Operations

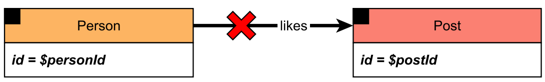
Updates / delete / 1

DEL 1
DEL 2
DEL 3
DEL 4
DEL 5
DEL 6
DEL 7
DEL 8

query	Updates / delete / 1
title	Remove person and its personal forums and message (sub)threads
pattern	
description	Remove a Person with ID $personId and its edges (isLocatedIn, studyAt, workAt, hasInterest, likes, knows, hasMember, hasModerator, hasCreator). Additionally, remove the Album and Wall Forums whose moderator is the Person and remove all Messages the Person has created in the rest of the Forums (Groups).
params	1 $personId ID
CPs	9.3, 9.4, 9.5
relevance	• Removal of a Person removes Forums of type "Walls" and "Albums" but not "Groups", which can continue if even the founder has left the network. For Groups, the hasModerator edge is deleted. We have discussed various approaches to appoint a new moderator, e.g. 1. choose member at random from the set of existing group members or 2. the member with the oldest group join date becomes the moderator. However, to keep the generator and the workload simple, currently no moderator is selected, leaving the group without a moderator. • Removal of a Person removes all Posts/Comments they are creator of this could result in the removal of a Comment in the middle of a thread.

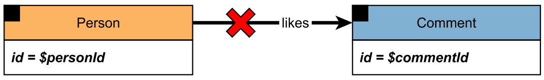
Updates / delete / 2

DEL 1
DEL 2
DEL 3
DEL 4
DEL 5
DEL 6
DEL 7
DEL 8

query	Updates / delete / 2
title	Remove post like
pattern	
description	Given a Person with ID $personId and a Post with ID $postId, remove the likes edge between them.
params	1 $personId ID 2 $postId ID
CPs	9.4
relevance	Removal of a likes edge is a rare event, e.g. people accidently liking a Post, this can be reflected by the relative frequency of the operation.

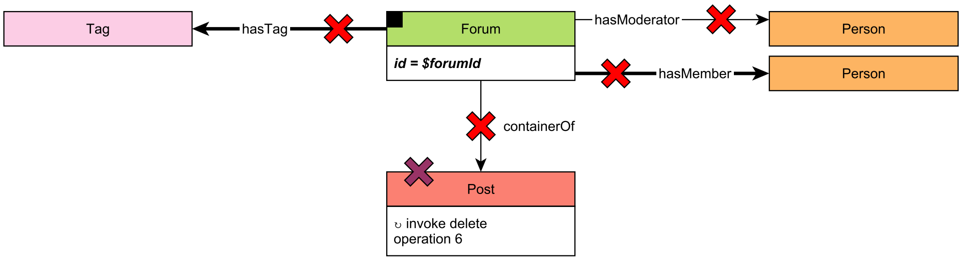
Updates / delete / 3

DEL 1
DEL 2
DEL 3
DEL 4
DEL 5
DEL 6
DEL 7
DEL 8

query	Updates / delete / 3
title	Remove comment like
pattern	
description	Given a Person with ID `$personId` and a Comment with ID `$commentId`, remove the likes edge between them.
params	1 | `$personId` \| ID 2 \| `$commentId` | ID
CPs	9.4
relevance	Removal of a likes edge is a rare event, e.g. people accidently liking a Comment, this can be reflected by the relative frequency of the operation.

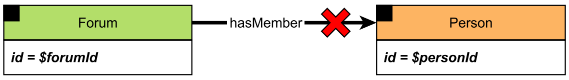
Updates / delete / 4

DEL 1
DEL 2
DEL 3
DEL 4
DEL 5
DEL 6
DEL 7
DEL 8

query	Updates / delete / 4
title	Remove forum and its content
pattern	
description	Remove a Forum with ID `$forumId` and its edges (hasModerator, hasMember, hasTag) and all Posts in the Forum (connected by containerOf edges) and their direct and transitive Comments.
params	1 | `$forumId` | ID
CPs	9.3, 9.4, 9.5
relevance	n/a

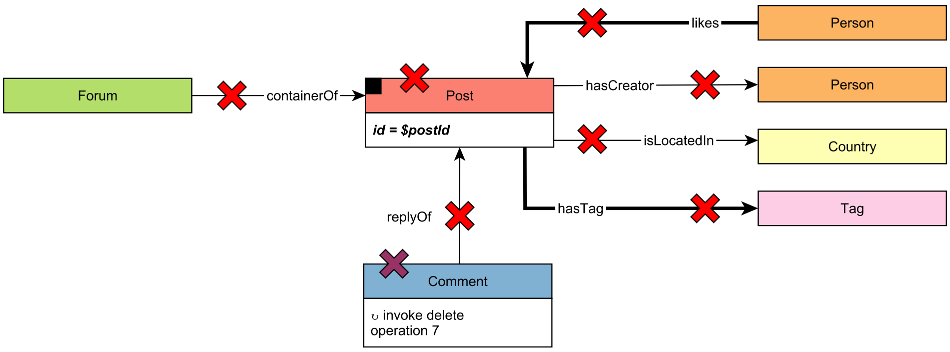
Updates / delete / 5

DEL 1 DEL 2 DEL 3 DEL 4 DEL 5 DEL 6 DEL 7 DEL 8

query	Updates / delete / 5
title	Remove forum membership
pattern	
description	Given a Forum with ID `$forumId` and a Person with ID `$personId`, remove the hasMember edge between them.
params	1 `$forumId` ID 2 `$personId` ID
CPs	9.4
relevance	n/a

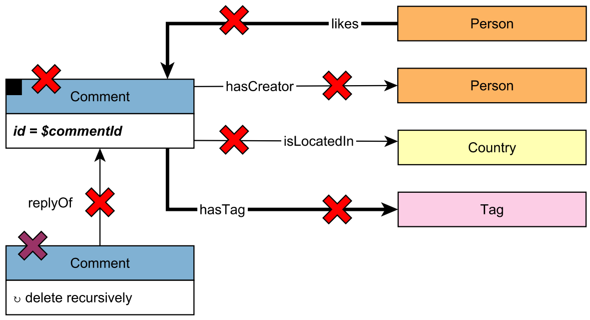
Updates / delete / 6

DEL 1 DEL 2 DEL 3 DEL 4 DEL 5 DEL 6 DEL 7 DEL 8

query	Updates / delete / 6
title	Remove post thread
pattern	
description	Remove a Post node with ID `$postId` and its edges (isLocatedIn, likes, hasCreator, hasTag, containerOf). Remove all replies to the Post and the connecting replyOf edges. In addition, remove all transitive reply Comments to the Post and their edges.
params	1 `$postId` ID
CPs	9.3, 9.4, 9.5
relevance	n/a

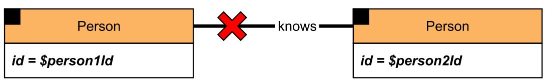
Updates / delete / 7

DEL 1 DEL 2 DEL 3 DEL 4 DEL 5 DEL 6 DEL 7 DEL 8

query	Updates / delete / 7
title	Remove comment subthread
pattern	
description	Remove a Comment node with ID $commentId and its *edges* (isLocatedIn, likes, hasCreator, hasTag). In addition, remove all replies to the Comment connected by replyOf and their *edges*.
params	1 | $commentId | ID
CPs	9.3, 9.4, 9.5
relevance	n/a

Updates / delete / 8

DEL 1 DEL 2 DEL 3 DEL 4 DEL 5 DEL 6 DEL 7 DEL 8

query	Updates / delete / 8
title	Remove friendship
pattern	
description	Given two Person nodes with IDs $person1Id and $person2Id, remove the knows edge between them.
params	1 | $person1Id | ID 2 | $person2Id | ID
CPs	9.4
relevance	n/a

1.3 Parameter Curation

To prevent caching query results, the SNB Interactive v2 driver instantiates the parameterized complex read (IC) query templates with different *substitution parameters* (a.k.a. parameter bindings). However, the naïve approach (using a uniform random sampling of parameters and ignoring updates) leads to unstable runtimes, which compromise both the benchmark's understandability and reproducibility. To ensure stable runtimes, LDBC invented *parameter curation* techniques, which select parameters that produce query runtimes with a unimodal (preferably Gaussian) distribution [5, 10].

1.3.1 Building Blocks for Parameter Curation

Temporal bucketing To ensure that operations are always executable, i.e. they avoid targeting nodes that are yet to be inserted or ones that are already deleted, the parameter curation process in Interactive v2 employs *temporal bucketing*. Namely, we create a parameter bucket for *each day in the simulation time of the update streams*, i.e. each day in the simulation time has its own distinct set of parameters. This is a novel feature in Interactive v2 – previous SNB benchmarks lacked this feature and only selected parameters from the *initial snapshot*.

Factor tables As shown in Figure 1.1, the parameter generation is a two-step process. The *factor generator* produces *factor tables*, which contain data cube-like summary statistics [4] of the temporal graph such as the number of Messages for friends. The factor generator is executed in a distributed setup using Spark as this computation includes expensive joins over large tables, e.g. knows(`person`, `friend`) ⋈ hasCreator(`person`, `comment`).

1.3.2 Parameter Curation for Relational Queries

For relational queries (without path-finding), we based our parameter generation on two techniques.

(1) Selecting windows To select the parameters that are expected to yield similar runtimes, we look for windows with the smallest variance for a given value using SQL window functions. The parameters are first sorted and grouped together based on their difference in frequency. Groups that are smaller than a given minimum threshold are discarded to select a group of parameters large enough to generate a sufficient amount of parameters. From the latter, we select the group with the smallest standard deviation.

(2) Selecting distributions For queries where we want to select parameters that are correlated or anti-correlated, we use factor tables encoding possible combinations (e.g. `countryPairsNumFriends` for `IC 3`): we select values near a high percentile for the correlated and a low percentile for the anti-correlated case.

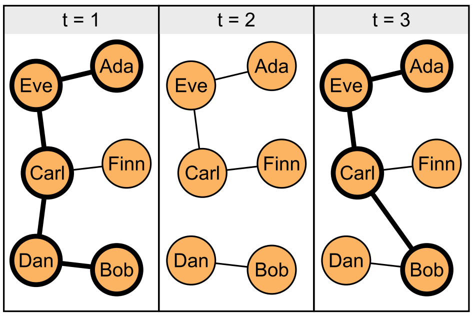
Generating the parameters The parameter candidates discovered by the previous approaches are stored in temporary tables. The parameter generation step uses these tables to select parameters for each day in the update stream.

1.3.3 Parameter Curation for Path-Finding Queries

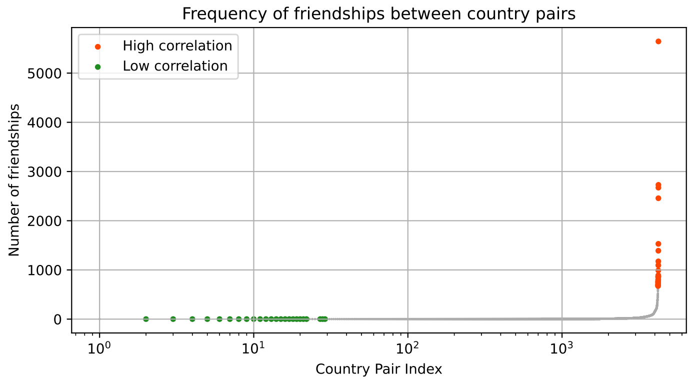
The effect of deletes A key distinguishing feature of graph data management systems is their first-class support for path queries [1]. We demonstrate why ensuring stable query runtimes for path queries is particularly challenging through the example of Figure 1.2a, where we query for the (unweighted) shortest path between *Ada* and *Bob* over a dynamic graph. Initially, at $t = 1$, the length of the shortest path is 4 hops. Then, the edge between *Carl* and *Dan* is deleted, making *Ada* and *Bob* unreachable from each other at $t = 2$. Finally, a new edge is inserted between *Carl* and *Bob*, yielding a shortest path of length 3 at $t = 3$. This illustrates how a given input parameter (a pair of Persons) can oscillate between being reachable and being in disjoint connected components over a short period. To ensure stable query runtimes for path queries in the presence of inserts and deletes, Interactive v2 introduces a novel *path curation* algorithm, which produces pairs of Person nodes whose shortest path length from each other is guaranteed to be exactly k hops at any point during a given day.

Graph construction The parameter curation algorithm builds two variants of the Person–knows–Person subgraph for each day based on the *temporal graph*: graph G_1 has the inserts applied until the beginning of the day and the deletes applied until the end of the day, while G_2 has the deletes applied until the beginning of the day and the inserts applied until the end of the day. For a given pair of Person nodes, their shortest path length in G_1 is an upper bound k_{upper} on their shortest path length at any point in the day – when the inserts during the day are gradually applied, the shortest path length can only become shorter. Conversely, G_2 gives a lower bound k_{lower} for the shortest path – the deletes can only make the shortest path length become longer.

Parameter selection The bounds provided by G_1 and G_2 guarantee for the shortest path length k that $k_{\text{lower}} \le k \le k_{\text{upper}}$ will hold at any point during the day. We can ensure that k will stay constant during the day by selecting Person pairs where $k_{\text{lower}} = k_{\text{upper}}$ holds. To this end, we select pairs who are exactly 4 hops apart in both G_1 and G_2, hence they will be always 4 hops apart during the given day. Unreachable pairs of nodes can be generated by calculating the connected components of G_2 and selecting nodes from disjoint components. The path curation for both the reachable and the unreachable cases is implemented using the NetworKit graph algorithm library [9].



(a) Shortest path (denoted with thick lines) between *Ada* and *Bob* in the presence of updates.



(b) Pairs of Countries in the `countryPairsNumFriends` factor table representing the number of friendships between both Countries.

Figure 1.2: Example graph and distribution for path curation.

1.3.4 Query Variants

The new workload introduces variants for three queries: `IC 3`, `IC 13`, `IC 14v2`.

Complex read 3: Correlated vs. anti-correlated Countries For `IC 3`, variant `IC 3(a)` starts from Countries that have a high correlation in the friendship network, while variant `IC 3(b)` starts from Countries that have a low correlation of friendships between. To generate these inputs, we use the `countryPairsNumFriends` factor table visualized in Figure 1.2b and select values at percentile 1.00 for variant `(a)` and percentile 0.01 for variant `(b)`.

Complex reads 13 and 14: Reachable vs. unreachable Persons Path queries are expected to have different runtimes if there is a path vs. when there is no path. While the performance characteristics vary highly between systems, in principle, the "no path" case should be simpler in the SNB graph, where one of the nodes is always in a small connected component. To distinguish between these cases, we have two variants for the two path queries `IC 13` and `IC 14v2`. For variants `(a)` we select Person pairs which *do not have a path*, and for variants `(b)` we select pairs which *have* a path of length 4.

1.3.5 Parameter Generator Implementation

The parameter generator is implemented in Python using NetworKit [9] and SQL queries executed by DuckDB [8]. Based on our experiments in [6, Figure 4.3], the new parameter generator is scalable. Even with the significant extra work performed for temporal bucketing, it outperforms the old parameter generator by more than 100× on SF1 000, and finishes in less than 1.5 hours on SF10 000.

1.4 Workload Scheduling and Benchmark Driver

In this section, we explain how operations are scheduled in the SNB Interactive workload, how the driver operates, and how the final *throughput* metric is determined. In all cases, we assume that the system-under-test has been populated with the *initial snapshot* using a *bulk loader* before the driver runs the operations.

1.4.1 Scheduling Operations

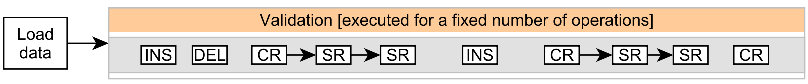
TCR (total compression ratio) The scheduling follows the *simulation time* of the temporal social network graph. The user-provided *total compression ratio* (TCR) value controls the speed at which the simulation is replayed. For example, a TCR value of 0.02 means that the simulation is replayed 50× faster, i.e. for every 20 milliseconds in wall clock time, 1 second passes in the simulation time.

Update operations The driver replays the update operations starting from the cutoff date, Nov 29, 2012. The operations are scheduled according to the distance of their start time from this date, adjusted by the TCR. They are then used to set the cadence of the schedule for the complex reads and, in turn, the short read queries, as we will explain momentarily.

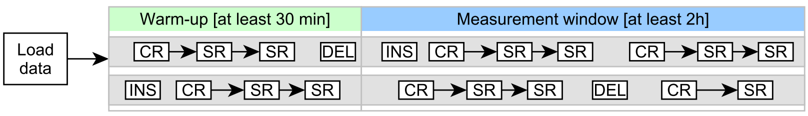
Complex read queries The *complex read queries* differ significantly in their expected runtimes as they touch on different amounts of data. As each query instance contributes equally to the output metric,[2] we balance them such that each query type is expected to take the same amount of time to execute. For example, `IC 14 (new)` is expected to be more difficult than `IC 13`, therefore it is scheduled less frequently. Frequencies vary based on the SF as the relative difficulties of queries change with the data size (e.g. three-hop neighbourhood queries grow faster on larger SFs than one-hop ones).

Short read queries Short read queries are triggered by complex read queries and other short read queries, and use their output as their input. For example, both `IC 3` and `IC 14` trigger `IS 2`, which also triggers itself. This mimics the real-life scenario of a user retrieving more information about Person profiles based on the result of the earlier queries. To see which short read queries are potentially triggered after given short read and complex read queries, see **??**.

1.4.2 Driver



(a) Validation workflow running on a single thread.



(b) Benchmark workflow using multiple threads.

Figure 1.3: Workflow of driver modes in SNB Interactive v2.

Driver modes The SNB driver has two key modes of operation. In *cross-validation mode* (Figure 1.3a)m the driver tests an implementation against the output of another implementation. To ensure deterministic results, operations in this mode are executed sequentially with no overlap between queries and updates. In *benchmark mode* (Figure 1.3b), the driver performs a benchmark run where queries and updates are issued concurrently from multiple threads. The run starts with a 30-minute warm-up period, followed by a 2-hour *measurement window*. This mode does not perform validation as query results may differ (slightly) due to concurrent updates.

[2] Unlike in TPC-H [11] and SNB BI [10], which use *geometric mean* in their metrics.

Dependency tracking To ensure that updates are executable, concurrent threads must be synchronized so that an operation is only executed when its dependencies exist in the network (e.g. two Persons can only become friends if both of them already exist). This is achieved via maintaining a global clock in the driver and performing *dependency tracking* for the updates [3]: each update operation has a timestamp denoting the creation time of the last operation it depends on. The data generator calculates these timestamp during generation and ensures that there is a minimum time separation, T_{safe}, between dependent entities to reduce synchronization overhead in the driver when executing operations. The driver then only needs to check every T_{safe} time whether a given update operation can be executed. By default, T_{safe} is set to 10 seconds in the simulation time.

Latency requirements The workload simulates a highly transactional scenario where operations are subject to (soft) latency requirements. To incorporate this in the workload, it prescribes the *95% on-time requirement*: for a benchmark run to be successful, 95% of the operations must start *on-time*, i.e. within 1 second of their scheduled start time. Benchmark runs where the system-under-test falls behind too much from the schedule are considered invalid.

Throughput The throughput of a run is the total number of operations (`IC` , `IS` , `INS`, `DEL`) executed per second. A lower TCR value implies a higher throughput.

Individual execution times To facilitate deeper analyis, the benchmark driver also collects all individual query execution times. Based on these, the benchmark reports must include statics for each operation type (min, max, mean, P_{50}, P_{90}, P_{95}, and P_{99} of the execution times).

Driver implementation in v2 The Interactive v2 is implemented in Java 17. It consists of 26 500 lines of code for the core project and an additional 18 000 lines of test code. The new version contains several patches including bug fixes, usability improvements, and performance optimizations.

Bibliography

[1] Renzo Angles et al. "Foundations of Modern Query Languages for Graph Databases". In: *ACM Comput. Surv.* 50.5 (2017), 68:1–68:40. DOI: 10.1145/3104031.

[2] Alin Deutsch et al. "Graph Pattern Matching in GQL and SQL/PGQ". In: *SIGMOD*. ACM, 2022, pp. 2246–2258. DOI: 10.1145/3514221.3526057.

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

[4] Jim Gray et al. "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals". In: *Data Min. Knowl. Discov.* 1.1 (1997), pp. 29–53. DOI: 10.1023/A:1009726021843.

[5] Andrey Gubichev and Peter A. Boncz. "Parameter Curation for Benchmark Queries". In: *TPCTC*. Vol. 8904. Lecture Notes in Computer Science. Springer, 2014, pp. 113–129.

[6] David Püroja. "LDBC Social Network Benchmark Interactive v2". https://ldbcouncil.org/docs/papers/msc-thesis-david-puroja-snb-interactive-v2-2023.pdf. Master's thesis. Universiteit van Amsterdam, 2023.

[7] David Püroja et al. "The LDBC Social Network Benchmark Interactive workload v2: A transactional graph query benchmark with deep delete operations". In: *CoRR* abs/2307.04820 (2023). DOI: 10.48550/arXiv.2307.04820.

[8] Mark Raasveldt and Hannes Mühleisen. "DuckDB: An Embeddable Analytical Database". In: *SIGMOD*. ACM, 2019, pp. 1981–1984. DOI: 10.1145/3299869.3320212.

[9] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. "NetworKit: A tool suite for large-scale complex network analysis". In: *Netw. Sci.* 4.4 (2016), pp. 508–530. DOI: 10.1017/nws.2016.20.

[10] Gábor Szárnyas et al. "The LDBC Social Network Benchmark: Business Intelligence Workload". In: *Proc. VLDB Endow.* 16.4 (2022), pp. 877–890. URL: https://ldbcouncil.org/docs/papers/ldbc-snb-bi-vldb-2022.pdf.

[11] TPC (Transaction Processing Performance Council). "TPC Benchmark H, revision 2.18.0". In: (2017), pp. 1–138. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf.