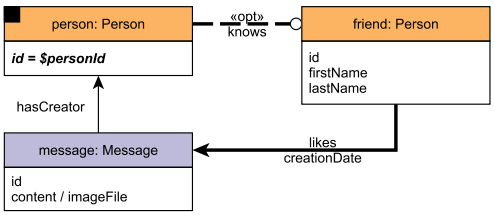
Interactive / complex / 7

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

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

params

1	`$personId`	ID

result

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

sort

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

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