

## Interactive / complex / 1

IC 1	query	Interactive / complex / 1			
IC 2	title	Transitive friends with certain name			
IC 3	pattern				
IC 4	desc.	Given a start Person, find Persons with a given first name ( <i>firstName</i> ) 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.			
IC 5	params	1	personId	ID	
IC 6		2	firstName	String	
IC 7	result	1	otherPerson.id	ID	R
IC 8		2	otherPerson.lastName	String	R
IC 9		3	distanceFromPerson	32-bit Integer	C
IC 10		4	otherPerson.birthday	Date	R
IC 11		5	otherPerson.creationDate	DateTime	R
IC 12		6	otherPerson.gender	String	R
IC 13		7	otherPerson.browserUsed	String	R
IC 14		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	query	Interactive / complex / 2			
IC 2	title	Recent messages by your friends			
IC 3	pattern				
IC 4					
IC 5					
IC 6					
IC 7					
IC 8	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).			
IC 9					
IC 10	params	1	personId	ID	
IC 11		2	maxDate	Date	
IC 12					
IC 13	result	1	friend.id	ID	R
IC 14		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	<p>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 exercises 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.</p>			

## Interactive / complex / 3

query	Interactive / complex / 3																																		
title	Friends and friends of friends that have been to given countries																																		
pattern	<p>The diagram illustrates the query pattern. It starts with a <b>person: Person</b> entity (orange box) with attribute <code>id = \$personId</code>. This person <b>knows*1..2</b> <b>otherPerson: Person</b> entities (orange box) with attributes <code>id</code>, <code>firstName</code>, and <code>lastName</code>. Each <b>otherPerson</b> can be the <b>hasCreator</b> of one or more <b>Message</b> entities (purple box). The <b>Message</b> entities have attributes <code>\$startDate ≤ creationDate ≤ \$startDate + \$durationDays</code>. These messages are <b>isLocatedIn</b> <b>Country</b> entities (yellow box) and <b>City</b> entities (teal box). Specifically, there are two <b>Message</b> boxes: the top one is associated with <code>xCount = count</code> and the bottom one with <code>yCount = count</code>. Both <b>Message</b> boxes are <b>isLocatedIn</b> <b>countryX: Country</b> and <b>countryY: Country</b> entities. The <b>Country</b> entities have attributes <code>name = \$countryXName</code> and <code>name = \$countryYName</code> respectively. There are also <b>City</b> entities <b>cityX: City</b> and <b>cityY: City</b> which are <b>isPartOf</b> <b>countryX: Country</b> and <b>countryY: Country</b> respectively, indicated by red dashed arrows labeled <code>«neg» isPartOf</code>.</p>																																		
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	<table><tr><td>1</td><td>personId</td><td>ID</td><td></td></tr><tr><td>2</td><td>countryXName</td><td>String</td><td></td></tr><tr><td>3</td><td>countryYName</td><td>String</td><td></td></tr><tr><td>4</td><td>startDate</td><td>Date</td><td>Beginning of requested period</td></tr><tr><td>5</td><td>durationDays</td><td>32-bit Integer</td><td>Duration of requested period, in days. The interval [startDate, startDate + durationDays) is closed-open</td></tr></table>					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										
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	<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>xCount</td><td>32-bit Integer</td><td>A</td><td>Number of Messages from Country CountryX created by the Person within the given time</td></tr><tr><td>5</td><td>yCount</td><td>32-bit Integer</td><td>A</td><td>Number of Messages from Country CountryY created by the Person within the given time</td></tr><tr><td>6</td><td>count</td><td>32-bit Integer</td><td>A</td><td>count = xCount + yCount</td></tr></table>					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
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	<table><tr><td>1</td><td>xCount</td><td>↓</td><td></td></tr><tr><td>2</td><td>otherPerson.id</td><td>↑</td><td></td></tr></table>					1	xCount	↓		2	otherPerson.id	↑																							
1	xCount	↓																																	
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	<pre> graph LR     P1[person: Person id = \$personId] -- knows --&gt; P2[Person]     P2 -- hasCreator --&gt; Post1[Post creationDate &lt; \$startDate]     P1 -- knows --&gt; Friend[friend: Person]     Friend -- «opt» hasCreator --&gt; Post2[Post \$startDate ≤ creationDate &lt; \$startDate + \$durationDays]     Post2 -- hasTag --&gt; Tag[tag: Tag]     Post1 -- «neg» hasTag --&gt; Tag </pre>			
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
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	query	Interactive / complex / 5			
IC 2	title	New groups			
IC 3	pattern	 <pre> graph LR     P1[person: Person id = \$personId] -- knows*1..2 --&gt; P2[otherPerson: Person]     P2 -- hasMember \$minDate &lt; creationDate --&gt; F[forum: Forum id title]     F -- containerOf --&gt; Post[Post]     Post -- count --&gt; Count[count]     P2 -.-&gt; «opt» hasCreator  Post           </pre>			
IC 4	desc.	<p>Given a start Person, find the Forums which that Person's friends and friends of friends (excluding start Person) became Members of after a given date. For each Forum find the number of Posts that were created by any of these Persons. For each Forum and consider only those Persons which joined that particular Forum after the given date (minDate).</p>			
IC 5	params	1	personId	ID	
IC 6		2	minDate	Date	
IC 7	result	1	forum.title	Long String	R
IC 8		2	postCount	32-bit Integer	A
IC 9		Number of Posts made in forum that were created by friends			
IC 10	sort	1	postCount	↓	
IC 11		2	forum.id	↑	
IC 12	limit	20			
IC 13	CPs	2.3, 3.3, 8.2, 8.5			
IC 14	relevance	<p>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 indexes, 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 indexes will be highly beneficial.</p>			

## 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	<pre>graph LR     P1[person: Person id = \$personId] -- knows*1..2 --&gt; P2[otherPerson: Person]     P2 -- hasCreator --&gt; Post[Post]     Post -- hasTag --&gt; T1[tag: Tag name = \$tagName]     Post -- hasTag --&gt; T2[otherTag: Tag name != \$tagName name]     Post -- count --&gt; Count[count]</pre>													
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	<table><tr><td>1</td><td>personId</td><td>ID</td><td></td></tr><tr><td>2</td><td>tagName</td><td>Long String</td><td></td></tr></table>					1	personId	ID		2	tagName	Long String		
1	personId	ID												
2	tagName	Long String												
result	<table><tr><td>1</td><td>otherTag.name</td><td>Long String</td><td>R</td><td rowspan="2">Number of Posts that were created by friends and friends of friends, which have the Tag otherTag</td></tr><tr><td>2</td><td>postCount</td><td>32-bit Integer</td><td>A</td></tr></table>					1	otherTag.name	Long String	R	Number of Posts that were created by friends and friends of friends, which have the Tag otherTag	2	postCount	32-bit Integer	A
1	otherTag.name	Long String	R	Number of Posts that were created by friends and friends of friends, which have the Tag otherTag										
2	postCount	32-bit Integer	A											
sort	<table><tr><td>1</td><td>postCount</td><td>↓</td><td></td></tr><tr><td>2</td><td>otherTag.name</td><td>↑</td><td></td></tr></table>					1	postCount	↓		2	otherTag.name	↑		
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

query	Interactive / complex / 7				
title	Recent likers				
pattern					
desc.	<p>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.</p>				
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	<p>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.</p>				

IC 10  
IC 11  
IC 12  
IC 13  
IC 14

## Interactive / complex / 8

IC 1	query	Interactive / complex / 8			
IC 2	title	Recent replies			
IC 3	pattern	 <pre> graph TD     P[person: Person id = \$personId]     M[Message]     C[comment: Comment id content creationDate]     CA[commentAuthor: Person id firstName lastName]      M -- hasCreator --&gt; P     C -- hasCreator --&gt; CA     M -- replyOf --&gt; C           </pre>			
IC 4	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.			
IC 5	params	1	personId	ID	
IC 6	result	1	commentAuthor.id	ID	R
IC 7		2	commentAuthor.firstName	String	R
IC 8		3	commentAuthor.lastName	String	R
IC 9		4	comment.creationDate	DateTime	R
IC 10		5	comment.id	ID	R
IC 11		6	comment.content	Text	R
IC 12	sort	1	comment.creationDate	↓	
IC 13		2	comment.id	↑	
IC 14	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	query	Interactive / complex / 9			
IC 2	title	Recent messages by friends or friends of friends			
IC 3	pattern	 <pre> graph LR     person[person: Person] -- "id = \$personId" --&gt; knows[knows*1..2]     knows --&gt; otherPerson[otherPerson: Person]     otherPerson -- "hasCreator" --&gt; message[message: Message]     message -- "creationDate &lt; \$maxDate" --&gt; filter[filter]     filter --&gt; result[result]             </pre>			
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10					
IC 11					
IC 12	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).			
IC 13	params	1	personId	ID	
IC 14		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	<div><div><div><div>person: Person</div><div><code>id = \$personId</code></div></div><div>knows*2..2</div><div><div>foaf: Person</div><div><code>(month(birthday) = \$month and day(birthday) ≥ 21) or (month(birthday) = \$month+1 and day(birthday) &lt; 22)</code></div><div>id firstName lastName gender</div></div><div>isLocatedIn</div><div><div>city: City</div><div>name</div></div></div><div><div><div>common</div><div><div><div>person: Person</div><div>hasInterest</div><div>Tag</div></div><div><div>foaf: Person</div><div>hasCreator</div><div><div>count</div><div>Post</div></div><div>hasTag</div></div></div></div><div><div>uncommon</div><div><div><div>person: Person</div><div>«neg» hasInterest</div><div>Tag</div></div><div><div>foaf: Person</div><div>hasCreator</div><div><div>count</div><div>Post</div></div><div>hasTag</div></div></div></div></div></div>				
desc.	<p>Given a start Person with id <code>personId</code>, find that Person’s friends of friends (<code>foaf</code>) – 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 <code>commonInterestScore</code> is defined as follows:</p> <ul style="list-style-type: none"><li>• <code>common</code> = number of Posts created by friend, such that the Post has a Tag that the start person is interested in</li><li>• <code>uncommon</code> = number of Posts created by friend, such that the Post has no Tag that the start person is interested in</li><li>• <code>commonInterestScore</code> = <code>common</code> - <code>uncommon</code></li></ul>				
params	<div><div>1</div><div>personId</div><div>ID</div></div> <div><div>2</div><div>month</div><div>32-bit Integer</div></div>	<div>Between 1 and 12. Implementations may also pass the next month as an additional <code>nextMonth</code> parameter</div>			
result	<div><div>1</div><div>foaf.id</div><div>ID</div><div>R</div></div> <div><div>2</div><div>foaf.firstName</div><div>String</div><div>R</div></div> <div><div>3</div><div>foaf.lastName</div><div>String</div><div>R</div></div> <div><div>4</div><div>commonInterestScore</div><div>32-bit Integer</div><div>A</div></div> <div><div>5</div><div>foaf.gender</div><div>String</div><div>R</div></div> <div><div>6</div><div>city.name</div><div>String</div><div>R</div></div>				
sort	<div><div>1</div><div>commonInterestScore</div><div>↓</div></div> <div><div>2</div><div>foaf.id</div><div>↑</div></div>				
limit	10				
CPs	2.3, 3.3, 4.1, 4.2, 5.1, 5.2, 6.1, 7.1, 8.6				
relevance	<p>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.</p>				

## Interactive / complex / 11

IC 1	query	Interactive / complex / 11			
IC 2	title	Job referral			
IC 3	pattern	 <pre> graph TD     person[person: Person] -- "knows*1..2" --&gt; otherPerson[otherPerson: Person]     otherPerson -- "workAt year(workFrom) &lt; \$year" --&gt; company[company: Company]     company -- "isLocatedIn" --&gt; country[country: Country]             </pre> <p>The diagram illustrates a query pattern. It starts with a <b>person: Person</b> entity (orange box) with a variable <b>id = \$personId</b>. This entity is connected via a <b>knows*1..2</b> relationship to an <b>otherPerson: Person</b> entity (orange box). The <b>otherPerson</b> entity has attributes <b>id</b>, <b>firstName</b>, and <b>lastName</b>. From <b>otherPerson</b>, a <b>workAt</b> relationship leads to a <b>company: Company</b> entity (green box), with the condition <b>year(workFrom) &lt; \$year</b>. The <b>company</b> entity has an attribute <b>name</b>. Finally, the <b>company</b> is connected via an <b>isLocatedIn</b> relationship to a <b>country: Country</b> entity (yellow box), which has an attribute <b>name = \$name</b>.</p>			
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10	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).			
IC 11	params	1	personId	ID	
IC 12		2	countryName	String	
IC 13		3	workFromYear	32-bit Integer	
IC 14	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 descendant 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	<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>					1	personId	ID		2	tagClassName	Long String																		
1	personId	ID																												
2	tagClassName	Long String																												
result	<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>					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	
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	<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>					1	replyCount	↓		2	friend.id	↑																		
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	query	Interactive / complex / 13			
IC 2	title	Single shortest path			
IC 3	pattern				
IC 6	desc.	<p>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:</p> <ul style="list-style-type: none"> <li>• -1: no path found</li> <li>• 0: start person = end person</li> <li>• &gt; 0: path found (start person ≠ end person)</li> </ul>			
IC 13	params	1	person1Id	ID	
IC 14		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	query	Interactive / complex / 14		
IC 2	title	Trusted connection paths		
IC 3	pattern	<div> <div> Enumerate all unweighted shortest paths on knows edges from person1 to person2. <pre> graph LR     p1[person1: Person id = \$person1Id] -- knows* --&gt; p2[person2: Person id = \$person2Id] </pre> </div> <div> For each edge on the path, calculate a weight based on interactions between the pair of Persons of the edge, are calculated as a sum of cases #1 and #2 for the Persons (both ways), and the sum of these weights determine the total weight of each path. <pre> graph LR     p1((p1)) -- knows --&gt; pX((pX))     pX -- knows --&gt; pY((pY))     pY -- ... --&gt; pW((pW))     pW -- knows --&gt; p2((p2)) </pre> </div> </div> <div> <div> Case 1: Replies on Posts, weight += 1.0 × count(c) <pre> graph LR     pA[personA: Person] -- knows --&gt; pB[personB: Person]     c[c: Comment] -- hasCreator --&gt; pA     c -- replyOf --&gt; post[post: Post]     post -- hasCreator --&gt; pB </pre> </div> <div> Case 2: Replies on Comments, weight += 0.5 × count(c1) <pre> graph LR     pA[personA: Person] -- knows --&gt; pB[personB: Person]     c1[c1: Comment] -- hasCreator --&gt; pA     c1 -- replyOf --&gt; c2[c2: Comment]     c2 -- hasCreator --&gt; pB </pre> </div> </div>		
IC 4	desc.	<p>Given two Persons, find all (unweighted) shortest paths between these two Persons, in the subgraph induced by the knows relationship.</p> <p>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.</p> <p>The weight for a pair of Persons is calculated based on their interactions:</p> <ul style="list-style-type: none"> <li>• Every direct reply (by one of the Persons) to a Post (by the other Person) contributes 1.0.</li> <li>• Every direct reply (by one of the Persons) to a Comment (by the other Person) contributes 0.5.</li> </ul> <p>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 <math>2 \times 1.0 + 1 \times 0.5 + 3 \times 1.0 + 4 \times 0.5 = 7.5</math>).</p> <p>Return all the paths with shortest length, and their weights. Do not return any rows if there is no path between the two Persons.</p>		
IC 5	params	1	person1Id	ID
IC 6		2	person2Id	ID
IC 7	result	1	personIdsInPath	[ID] C identifiers representing an ordered sequence of the Persons in the path
IC 8		2	pathWeight	64-bit Float C
IC 9	sort	1	pathWeight	↓ The order of paths with the same weight is unspecified
IC 10	CPs	3.3, 5.3, 7.2, 7.3, 7.5, 7.7, 8.1, 8.2, 8.3, 8.6		
IC 11	relevance	<p>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.</p>		
IC 12				
IC 13				
IC 14				