School of Computer Science, McGill University

COMP-421 Database Systems, Winter 2018

Non-Graded Assignment : MapReduce, Transactions & Graph Databases

Due Date : Never !!

This is not a graded assignment. It is to help you practice the last topics in the class for final exam. Try to do it on your own before checking the solutions.

Ex. 1 — Pig Latin

Assume the following relations:

```
Person(pid,pname)
Place(name, province, population, mayorid)
mayorid FOREIGN KEY to Person(pid)
Property(pid, name, province)
pid FOREIGN KEY to Person(pid), name,province FOREIGN KEY to Place(name,province)
```

The relations describe a small database of people and where they live. A person has an identifying pid and a name. A place has a name, is in a province, and has a certain population. Each place name occurs only once in any given provide. A place has (at most) one mayor. People have properties in places. Each person can have property at several places, and obviously, there are many properties in a place.

Formulate the following queries in Pig Latin. Ignore the load and store (your last created relation is the output). Ignore any flattening that you might have to do.

1. Return the name of the mayor of Montreal.
2. Return the pids of persons that have properties at more than one place.
3. Give the names of the mayors that have property at the place where they are mayor.
4. Return for each province the place(s) with the largest population. Your result set should have the province name, the place name and the population of that place.

Ex. 2 — Map Reduce

Given relations `R1(a, b, c)`, `R2(a, b, c)` and `Q(c, d, e)`

For the following queries written in SQL, outline an implementation using a single map-reduce phase. In particular, outline mapper and reducer functions, indicating the format of their input and their output, and how they process each of their input records. You can write pseudo-code or provide a description in half-formal English similar to how the implementation of the basic relational operators using map-reduce was discussed in class.

1.
   ```
   SELECT a,b
   FROM R1
   UNION
   SELECT a,b
   FROM R2
   ```

2.
   ```
   SELECT a, e
   FROM R1, Q
   WHERE R1.c = Q.c AND b < 20
   ```

3.
   ```
   SELECT c, MAX(b)
   FROM R1
   GROUP BY c
   HAVING c IN (SELECT c FROM Q)
   ```

Ex. 3 — Schedules and Concurrency Control

Given the following three schedules S1, S2, S3.

S1

T1	T2	T3
		r3(a)
	r2(b)	
w1(c)		
	w2(b)	
		r3(c)
	r2(a)	
	c2	
r1(b)		
w1(b)		
c1		
		w3(b)
		c3

S2

T1	T2	T3
r1(a)		
		r3(c)
	w2(b)	
w1(a)		
	r2(c)	
	w2(c)	
	c2	
w1(c)		
c1		
		r3(a)
		c3

S3

T1	T2	T3
r1(a)		
	w2(a)	
		r3(a)
		w3(b)
		c3
	r2(b)	
w1(c)		
	w2(c)	
	c2	
c1		

1. *For each of the schedules, show the serialization graph. Indicate whether the schedule is serializable or not. If it is serializable, provide an equivalent serial schedule.*

2. Assume now that the figures above does not depict the final schedules but the sequences of operations submitted to the system. Now assume a DBMS that uses strict 2PL for concur- rency control. *Describe how strict 2PL handles each of the sequences above. Add lock $S_i(a)$ when T_i acquires shared lock on a, $X_i(a)$ when T_i acquires exclusive lock on object a, and unlock request $U_i(a)$ when T_i releases any lock on object a to the above sequence.* The DBMS processes actions in the order shown. If a transaction is blocked, assume that all of its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

   Distinguish two variations of the locking scheme. (1) Upon a shared lock request $S_i(a)$ of transaction T_i if there are only shared locks active on a, then grant $S_i(a)$, else $S_i(a)$ must wait; (2) Upon a shared lock request $S_i(a)$ of transaction T_i, if there are only shared locks active on a and no lock is waiting on a, then grant $S_i(a)$, else $S_i(a)$ must wait.

Ex. 4 — Schedules and SQL

Assume relation `Enrolled(sid:INT, cid:CHAR(10), grade:INT)`
with example instance:

sid	cid	grade
4711	COMP-421	92
4711	MATH-240	82

Assume a database that does not have any concurrency control in place. For each of the executions below (starting with the example instance above), indicate

1.the values returned by the SELECT statements and the final value of the database.

2.whether the execution violates the anomalies (i) dirty read, (ii) lost update, (iii) unrepeatable read

(NOTE that the execution could violate all anomalies, or only some or none could hold).

(I)

```
T1: SELECT grade FROM Enrolled WHERE sid = 4711
T2: UPDATE Enrolled SET grade = grade + 5 WHERE cid = 'COMP-421'
T1: UPDATE Enrolled SET grade = 84 WHERE sid = 4711 AND cid = 'COMP-421'
T1: COMMIT
T2: COMMIT
```

(II)

```
T1: SELECT sid FROM Enrolled WHERE grade > 90
T2: UPDATE Enrolled SET grade = grade + 10 WHERE grade < 85 AND cid = 'COMP-421'
T1: SELECT sid FROM Enrolled WHERE grade > 80 AND grade <= 90
T1: COMMIT
T2: COMMIT
```

(III)

```
T1: UPDATE Enrolled SET grade = grade + 5 WHERE cid = 'MATH-240'
T2: UPDATE Enrolled SET grade = grade + 5 WHERE cid = 'COMP-421'
T3: SELECT cid, AVG(grade) FROM Enrolled GROUP BY cid
T2: ABORT
T3: ABORT
T1: COMMIT
```

(IV)

```
T1: SELECT cid, avg(GRADE) FROM Enrolled GROUP BY cid
T2: UPDATE Enrolled SET grade = grade + 5 WHERE cid = 'COMP-421'
T2: COMMIT
T1: SELECT sid, avg(GRADE) FROM Enrolled GROUP BY sid
T1: COMMIT
```

(V)

```
T1: SELECT * FROM Enrolled WHERE sid = 4711
T2: SELECT * FROM Enrolled WHERE sid = 4711
T1: UPDATE Enrolled SET grade = 100 WHERE sid = 4711 AND cid = 'COMP-421'
T2: UPDATE Enrolled SET grade = 100 WHERE sid = 4711 and  cid = 'MATH-240'
T1: COMMIT
T2: COMMIT
```

Ex. 5 — Graph Database Modelling

Consider the following relational tables that capture hereditary disorders of some test subjects and their genealogy. Your first task will be to create a graph database structure for the given relational data (no need to write Cypher to create the graph database, but it is recommended to practice doing it).
In the next task we will write some queries to understand who has potential risk for some of the hereditary disorders.

Subject

subjectid	gender
1	M
2	F
3	M
4	F
5	M
6	F
7	M
8	M
9	F
10	F
11	M
12	M
13	F
14	F
15	M

Genealogy

fatherid	motherid	childid
1	2	7
1	2	8
3	4	9
5	6	10
8	9	12
10	11	13
10	11	14
10	11	15

Disorder

disorderid	name
1	Lebers
2	Huntington

Affected

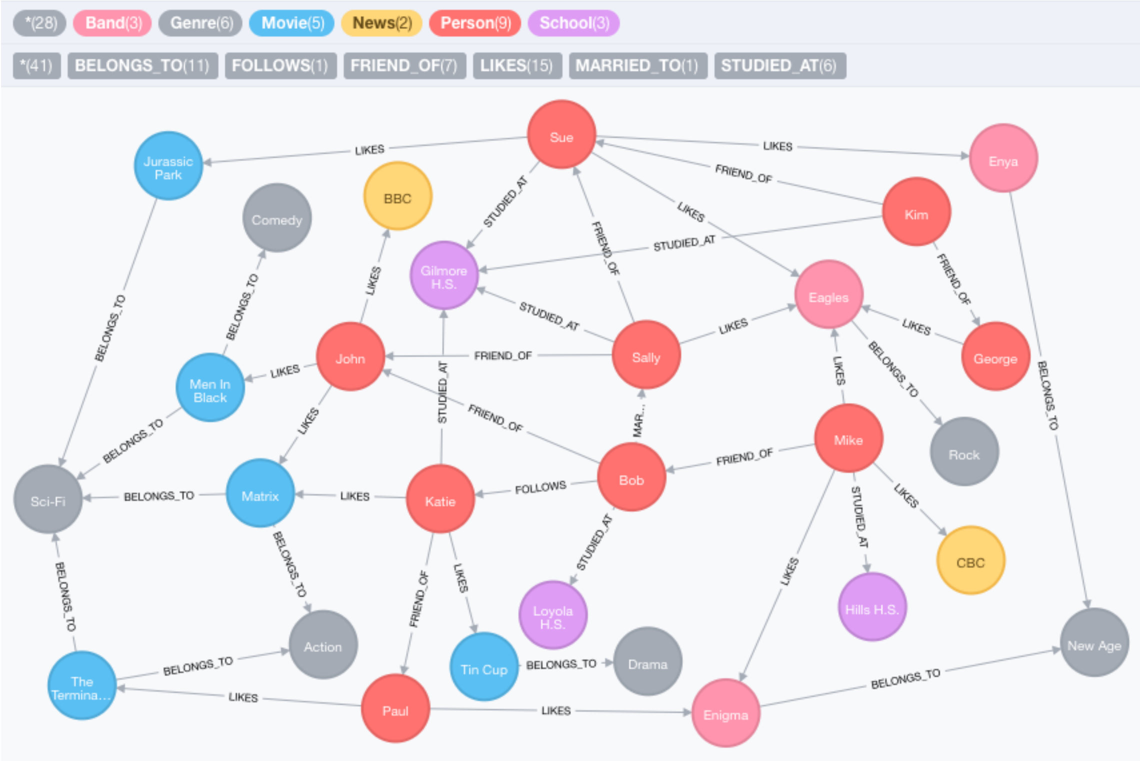
subjectid	disorderid
2	1
3	1
6	1
6	2
11	2

Additionally, we know that the defects for Lebers is passed down only from mother to her children (i.e. down the maternal line of lineage). An affected father cannot transmit it to any of his children.
On the other hand, Huntington can be passed down from any of the parent. This information will help us writing the correct queries against the graph to undestand who has potential risk for which disorder.

1. draw a graph model using the notations we saw in the class to represent this sample dataset. Remember that your model should be scalable (i.e, it should be able to handle more subjects, depth in genealogy (i.e., great grandparents, etc..) and also the fact that new disorders could be added and that a subject can have multiple disorders. You may leave out any artifical keys that you deem to be unnecessary in the graph model, if you chose to do so.
2. Write a Cypher query based on your model that will show all the people who has a potential risk for Huntington.
3. Write a Cypher query based on your model that will show all the people who has a potential risk for Lebers
4. Write a Cypher query based on your model that will show all the people who has a potential risk for Huntington but not Lebers.
5. Identify the siblings (i.e should have father AND mother) and create a sibling relationship between them. Bonus: Can you write a query in such a way that you create only one edge between the siblings ? i.e., for example only $(s7) - [: SIBLING\_OF] - > (s8)$ and does not create $(s8) - [: SIBLING\_OF] - > (s7)$?

Ex. 6 — Cypher

Consider the following graph representing a hypothetical social network.



Specifically, FRIEND_OF and MARRIED_TO are bi-directional relationships in real world, however we will just use one edge in the graph (in any arbitrary direction). Therefore when you write queries using them, you need to make sure that they are agnostic to the direction of the relationship edges for these two relationships (this was shown in class). I.e, for example Mike is a friend of Bob and vice versa, though there is only one edge from Mike to Bob.
You may use the cypher command given below to create the graph in Neo4j

```
CREATE
  (p1:Person {name:'Mike', city:'Montreal'}) ,(p2:Person {name:'Bob', city:'Montreal'})
 ,(p3:Person {name:'Katie'}) ,(p4:Person {name:'Sally', city:'Montreal'})
 ,(p5:Person {name:'John'}) ,(p6:Person {name:'Sue', city:'Ottawa'})
 ,(p7:Person {name:'Kim', city:'Ottawa'}) ,(p8:Person {name:'George', city:'Toronto'})
 ,(p9:Person {name:'Paul'})
 ,(h1:School {name:'Hills H.S.'}) ,(h2:School {name:'Loyola H.S.'})
 ,(h3:School {name:'Gilmore H.S.'})
 ,(b1:Band   {name:'Enigma'}) ,(b2:Band   {name:'Eagles'})
 ,(b3:Band   {name:'Enya'})
 ,(n1:News   {name:'CBC'}) ,(n2:News   {name:'BBC'})
 ,(m1:Movie  {name:'Jurassic Park', year:1993}) ,(m2:Movie  {name:'Men In Black', year:1997})
 ,(m3:Movie  {name:'Matrix', year:1999}) ,(m4:Movie  {name:'The Terminator', year:1984})
 ,(m5:Movie  {name:'Tin Cup', year:1996})
 ,(g1:Genre  {name:'Action'}) ,(g2:Genre  {name:'Sci-Fi'})
 ,(g3:Genre  {name:'Comedy'}) ,(g4:Genre  {name:'New Age'})
 ,(g5:Genre  {name:'Rock'}) ,(g6:Genre  {name:'Drama'})
 ,(p1)-[:FRIEND_OF]->(p2) ,(p2)-[:FRIEND_OF]->(p5)
 ,(p4)-[:FRIEND_OF]->(p5) ,(p4)-[:FRIEND_OF]->(p6)
 ,(p7)-[:FRIEND_OF]->(p6) ,(p7)-[:FRIEND_OF]->(p8)
 ,(p3)-[:FRIEND_OF]->(p9)
 ,(p2)-[:MARRIED_TO]->(p4)
 ,(p2)-[:FOLLOWS]->(p3)
 ,(p1)-[:STUDIED_AT {grad_year:2009}]->(h1) ,(p2)-[:STUDIED_AT {grad_year:2010}]->(h2)
 ,(p6)-[:STUDIED_AT {grad_year:2012}]->(h3) ,(p3)-[:STUDIED_AT {grad_year:2011}]->(h3)
 ,(p4)-[:STUDIED_AT {grad_year:2010}]->(h3) ,(p7)-[:STUDIED_AT {grad_year:2010}]->(h3)
 ,(p1)-[:LIKES]->(b1) ,(p9)-[:LIKES]->(b1)
 ,(p4)-[:LIKES]->(b2) ,(p1)-[:LIKES]->(b2)
 ,(p6)-[:LIKES]->(b2) ,(p8)-[:LIKES]->(b2)
 ,(p6)-[:LIKES]->(b3) ,(p6)-[:LIKES]->(m1)
 ,(p5)-[:LIKES]->(m2) ,(p5)-[:LIKES]->(m3)
 ,(p3)-[:LIKES]->(m3) ,(p3)-[:LIKES]->(m5)
 ,(p9)-[:LIKES]->(m4)
 ,(p1)-[:LIKES]->(n1) ,(p5)-[:LIKES]->(n2)
 ,(m1)-[:BELONGS_TO]->(g2) ,(m3)-[:BELONGS_TO]->(g1)
 ,(m3)-[:BELONGS_TO]->(g2) ,(m2)-[:BELONGS_TO]->(g2)
 ,(m2)-[:BELONGS_TO]->(g3) ,(m4)-[:BELONGS_TO]->(g1)
 ,(m4)-[:BELONGS_TO]->(g2) ,(m5)-[:BELONGS_TO]->(g6)
 ,(b1)-[:BELONGS_TO]->(g4) ,(b2)-[:BELONGS_TO]->(g5)
 ,(b3)-[:BELONGS_TO]->(g4)
;
```

1. List all the Movies in the graph
2. What year was the Movie Jurassic Park released ?
3. List all Sci-Fi Movies ordered by their release year.
4. List all Sci-Fi Movies that are also Comedy.
5. Find all the people who likes the Movie Matrix.
6. Find all the friends of Sally who went to the same school as her.
7. Find all people in Bob's immediate network (i.e. people he is friends with, married to)
8. Find all friends in Bob's network upto an including friends' friends (you can ignore married relationship from consideration)
9. Find all things that John likes.
10. Find a list of movies that Bob's immediate friends like.
11. List all the Band Genres.
12. What year did Mike Graduate ?
13. Find a list of people who went to the same school and graduated the same year, but are not immediate friends.
14. Find the names of all people who are married.
15. Find a list of Movies for John that he has not yet liked but is of the same Genre as the other movies that he has liked.
16. In this queston, we will create a new entity type, FanClub. Each FanClub entity will have one property, city, where the FanClub is based on. Each Fan Club will be ”ASSOCIATED_TO” a specific band. Create Fan Clubs that are located in Montreal and Ottawa for the Eagles band and Enigma (so four fan clubs in total). Additionally create a ”MEMBER_OF” relationship between the Persons located in the same city as that of the fan club and also likes the corresponding bands.
    You may have to do this is question using more than one statements to make it simple.