Assignment 6

Jashjeet Singh Madan

April 1, 2020

Part 1

Q.1.

In the translation algorithm from SQL to RA, when we eliminated set predicates, we tacitly assumed that the argument of each set predicate was a (possibly parameterized) SQL query that did not use a UNION, INTERSECT, or an EXCEPT operation. In this problem, you are asked to extend the translation algorithm from SQL to RA such that (possibly parameterized) set predicates [NOT] IN are eliminated that have as an argument a SQL query that uses a UNION, INTERSECT, or EXCEPT operation. More specifically, consider the following types of queries using the [NOT] IN set predicate. (Help taken from Harsha R.S.)

SELECT L(r1,...,rk) FROM R1 r1, ..., Rk rk WHERE C1(r1,...,rk) AND r1.A1 [NOT] IN (SELECT DISTINCT s1.B1 FROM S1 s1,..., S1 sm WHERE C2(s1,...,sm,r1,...,rk) [UNION|INTERSECT|EXCEPT] SELECT DISTINCT t1.C1 FROM T1 t1, ..., Tn tn WHERE C3(t1,...,tn,r1,...,rk))

(a)

Show how such SQL queries can be translated to equivalent RA expressions. Be careful in the translation since you should take into account that projections do not in general distribute over intersections or over set differences.

SELECT L(r1) FROM R1 r1 WHERE C1(r1) AND r1.A1 [NOT] IN (SELECT DISTINCT s1.B1 FROM S1 s1 WHERE C2(s1,r1) [UNION|INTERSECT|EXCEPT] SELECT DISTINCT t1.C1 FROM T1 t1 WHERE C3(t1,r1))

"IN" k=1, m=1 and n=1 case:

Let E represent the following query:

$$\pi_{L(r)}(\sigma_{C1(R1)}(R1)) \bowtie (\pi_{(S1.B1)}(S1 \ltimes_{C2(S1,R2)} R2)[\cup | \cap | -]\pi_{(T1.C1)}(T1 \ltimes_{C3(T1,R3)} R3))$$

Then the final query will be represented by:

$$\pi_{L(r)}(R1 \ltimes E)$$

"IN" General case:

Let E represent the following query:

$$\pi_{L(r1,...,rk)}(\sigma_{C1(R1...Rk)}(R1...Rk)) \bowtie (\pi_{(S1.B1)}(S1...Sm \ltimes_{C2(S1...Sm,R1...Rk)} R1...Rk)[\cup | \cap | -]\pi_{(T1.C1)}(T1...Tn \ltimes_{C3(T1...Tn,R1...Rk)} R1...Rk))$$

Then the final query will be represented by:

$$\pi_{L(r1,...,rk)}(R1...Rk \ltimes E)$$

"NOT IN" k=1, m=1 and n=1 case:

Let E represent the following query:

$$\pi_{L(r)}(\sigma_{C1(R1)}(R1)) - (\pi_{(S1.B1)}(S1 \ltimes_{C2(S1,R2)} R2)[\cup | \cap | -]\pi_{(T1.C1)}(T1 \ltimes_{C3(T1,R3)} R3))$$

Then the final query will be represented by:

$$\pi_{L(r)}(R1 \ltimes E)$$

"NOT IN" General case:

Let E represent the following query:

$\pi_{L(r1,...,rk)}(\sigma_{C1(R1...Rk)}(R1...Rk)) - (\pi_{(S1.B1)}(S1...Sm \ltimes_{C2(S1...Sm,R1...Rk)} R1...Rk)[\cup| \cap |-]\pi_{(T1.C1)}(T1...Tn \ltimes_{C3(T1...Tn,R1...Rk)} R1...Rk))$

Then the final query will be represented by:

$\pi_{L(r1,...,rk)}(R1...Rk \ltimes E)$

(b)

Show how your translation can be improved when the variable “r1” does not occur in the conditions “C2” and “C3” in the subquery. You don’t have to solve this query for the general case, but rather for the SQL query

SELECT L(r) FROM R1 r1 WHERE C1(r1) [NOT] IN r1.A (SELECT DISTINCT s1.B1 FROM S1 s1 WHERE C2(s1) [UNION|INTERSECT|EXCEPT] SELECT DISTINCT t1.C1 FROM T1 t1 WHERE C3(t1))

"IN" k=1, m=1 and n=1 case:

Let E represent the following query:

$\pi_{L(r)}(\sigma_{C1(R1)}(R1)) \bowtie (\pi_{(S1.B1)}(\sigma_{(C2(S1))}S1[\cup| \cap |-]\pi_{(T1.C1)}(T1)))$

Then the final query will remain same and will be represented by:

$\pi_{L(r)}(R1 \ltimes E)$

"NOT IN" k=1, m=1 and n=1 case:

Let E represent the following query:

$\pi_{L(r)}(\sigma_{C1(R1)}(R1)) - (\pi_{(S1.B1)}(S1[\cup| \cap |-]\pi_{(T1.C1)}(T1)))$

Then the final query will remain same and will be represented by:

$\pi_{L(r)}(R1 \ltimes E)$

Q.2.

We have to prove the correctness of the following rewrite rule

$\pi_a(R \bowtie_{(R.a=S.b \wedge R.b=S.a)} S) = \pi_a(\pi_{a,b}(R) \cap \pi_{a,b}(S))$

$\pi_a(R \bowtie_{(R.a=S.b \wedge R.b=S.a)} S) = \{a|(\exists r \exists s(R|a,b,c)(r) \wedge (S|a,b,d)(s) \wedge r.a = s.b \wedge r.b = s.a)\}$

$=\{a|(\exists a \exists b \exists c \exists d(R|a,b,c) \wedge (S|a,b,d) \wedge R.a = S.b \wedge R.b = S.a)\}$

$=\{a|((\exists a \exists b \exists c(R|a,b,c)) \wedge (\exists a \exists b \exists d(S|a,b,d)) \wedge R.a = S.b \wedge R.b = S.a)\}$

$=\{a|(\exists a \exists b((\exists c(R|a,b,c)) \wedge (\exists d(S|a,b,d)) \wedge R.a = S.b \wedge R.b = S.a))\}$

$=\{a|(\exists b(((a,b) \in \pi_{(a,b)}(R)) \wedge ((b,a) \in \pi_{(b,a)}(S)) \wedge R.a = S.b \wedge R.b = S.a))\}$

The equality of ((R.a=S.b) and (R.b=S.a)) which is nothing but ((R.a,R.b)=(S.b,S.a)). Taking the intersection will give the same result.

$=\pi_a(\pi_{(a,b)}(R) \cap \pi_{(b,a)}(S))$

Part 2

Q.3. Part (a)

$\pi_{(S.sid,S.sname)}$
$(\sigma_{(S.sid=M.sid \wedge M.major='CS' \wedge T.sid=S.sid \wedge T.bookno=C.bookno \wedge C.bookno=B1.bookno \wedge C.citedbookno=B2.bookno \wedge B1.price<B2.price)} (S \times M \times T \times C \times B1 \times B2))$

Q.3. Part (b)

Step 1

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge M.major='CS' \wedge T.sid=S.sid \wedge T.bookno=C.bookno \wedge C.bookno=B1.bookno \wedge C.citedbookno=B2.bookno \wedge B1.price<B2.price)}(S \times M \times T \times C \times B1 \times B2))$$

Step 2

Pushing selections and projections inside before joining

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge T.sid=S.sid \wedge T.bookno=C.bookno \wedge C.bookno=B1.bookno \wedge C.citedbookno=B2.bookno \wedge B1.price<B2.price)}(\sigma_{(M.major='CS')}(S \times M \times T \times C \times B1 \times B2)))$$

Step 3

Using natural joins instead of cross joins

$$\pi_{(S.sid,S.sname)}(\pi_{(S.sid,S.sname)}(S) \bowtie_{(S.sid=T.sid)} (\pi_{(T.sid)}(T \bowtie_{(T.bookno=C.bookno)} (C \bowtie_{(C.bookno=B1.bookno)} (B1)) \bowtie_{(C.citedbookno=B2.bookno \wedge B1.price<B2.price)} (B2))) \bowtie_{(T.sid=M.sid)} (\sigma_{(M.major='CS')}(M)))$$

Step 4

Using semijoins to keep only required attributes

$$\pi_{(S.sid,S.sname)}(\pi_{(S.sid,S.sname)}(S) \ltimes \pi_{(T.sid)}(\pi_{(T.sid)}(T \bowtie_{(T.bookno=C.bookno)} (C \bowtie_{(C.bookno=B1.bookno)} (B1)) \bowtie_{(C.citedbookno=B2.bookno \wedge B1.price<B2.price)} (B2))) \bowtie_{(T.sid=M.sid)} (\sigma_{(M.major='CS')}(M)))$$

Q.4. Part (a)

$$\pi_{(S.sid,S.sname,M.major)}(\sigma_{(M.sid=S.sid)}(M) \times (\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge T.sid=S.sid \wedge T.bookno=B.bookno \wedge B.price<50)}(S \times M \times T \times B))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge M.major='CS')}(S \times M))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge T.sid=S.sid \wedge T.bookno=B.bookno \wedge B.price<30)}(S \times M \times T \times B))))$$

Q.4. Part (b)

Step 1

$$\pi_{(S.sid,S.sname,M.major)}(\sigma_{(M.sid=S.sid)}(M) \times (\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge T.sid=S.sid \wedge T.bookno=B.bookno \wedge B.price<50)}(S \times M \times T \times B))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge M.major='CS')}(S \times M))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid \wedge T.sid=S.sid \wedge T.bookno=B.bookno \wedge B.price<30)}(S \times M \times T \times B))))$$

Step 2

Pushing selections and projections inside before joining

$$\pi_{(S.sid,S.sname,M.major)}(\sigma_{(M.sid=S.sid)}(M) \times (\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=T.sid)}(\sigma_{(T.bookno=B.bookno)}(\sigma_{(B.price<50)}(S \times T \times B))))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=M.sid)}(\sigma_{(M.major='CS')}(S \times M)))$$

$$-$$

$$\pi_{(S.sid,S.sname)}(\sigma_{(S.sid=T.sid)}(\sigma_{(T.bookno=B.bookno)}(\sigma_{(B.price<30)}(S \times T \times B))))))$$

Step 3

Using natural joins instead of cross joins

$$\pi_{(S.sid,S.sname,M.major)}(\pi_{(M.major)}(M) \bowtie_{(M.sid=S.sid)} (\pi_{(S.sid,S.sname)}(S \bowtie_{(S.sid=T.sid)} T \bowtie_{(T.bookno=B.bookno \wedge B.price<50)} B)$$

$$-$$

$$\pi_{(S.sid,S.sname)}(S \bowtie_{(S.sid=M.sid \wedge M.major='CS')} M)$$

$$-$$

$$\pi_{(S.sid,S.sname)}(S \bowtie_{(S.sid=T.sid)} T \bowtie_{(T.bookno=B.bookno \wedge B.price<30)} B)))$$

Step 4

Using semijoins to keep only required attributes

$$\pi_{(S.sid,S.sname,M.major)}(\pi_{(M.major)}(M) \bowtie_{(M.sid=S.sid)} \pi_{(S.sid,S.sname)}(S)) \ltimes$$

$$(\pi_{(T.sid)}(T \bowtie_{(T.bookno=B.bookno \wedge B.price<50)} B) - \pi_{(M.sid)}(\sigma_{(M.major='CS')}M) - \pi_{(T.sid)}(T \bowtie_{(T.bookno=B.bookno \wedge B.price<30)} B))$$

Q.5. Part (a)

$$(\pi_{(S.sid,S.sname,B.bookno)}(\sigma_{(S.sid=T.sid \wedge T.bookno=B.bookno)}(S \times T \times B)))$$

$$-$$

$$(\pi_{(q1.sid,q1.sname,q1.bookno)}(\pi_{(q1.sid,q1.sname,q1.bookno,q1.price)}(\sigma_{(S1.sid=T1.sid \wedge T1.bookno=B1.bookno)}(S1 \times T1 \times B1)q1)$$
$$\bowtie_{(q1.sid=q2.sid \wedge q1.price<q2.price)} \pi_{(q2.sid,q2.sname,q2.bookno,q2.price)}(\sigma_{(S2.sid=T2.sid \wedge T2.bookno=B2.bookno)}(S2 \times T2 \times B2)q2)))$$

Q.5. Part (b)

Step 1

$$(\pi_{(S.sid,S.sname,B.bookno)}(\sigma_{(S.sid=T.sid \wedge T.bookno=B.bookno)}(S \times T \times B)))$$

$$-$$

$$(\pi_{(q1.sid,q1.sname,q1.bookno)}(\pi_{(q1.sid,q1.sname,q1.bookno,q1.price)}(\sigma_{(S1.sid=T1.sid \wedge T1.bookno=B1.bookno)}(S1 \times T1 \times B1)q1))$$
$$\bowtie_{(q1.sid=q2.sid \wedge q1.price<q2.price)} (\pi_{(q2.sid,q2.sname,q2.bookno,q2.price)}(\sigma_{(S2.sid=T2.sid \wedge T2.bookno=B2.bookno)}(S2 \times T2 \times B2)q2)))$$

Step 2

Using natural joins instead of cross joins

$$(\pi_{(S.sid,S.sname,B.bookno)}(S \bowtie_{(S.sid=T.sid)} T \bowtie_{(T.bookno=B.bookno)} B))$$

$$-$$

$$(\pi_{(q1.sid,q1.sname,q1.bookno)}(\pi_{(S1.sid,S1.sname,B1.bookno,B1.price)}(S1 \bowtie_{(S1.sid=T1.sid)} T1 \bowtie_{(T1.bookno=B1.bookno)} B1)q1)$$
$$\bowtie_{(q1.sid=q2.sid \wedge q1.price<q2.price)} (\pi_{(S2.sid,S2.sname,B2.bookno,B2.price)}(S2 \bowtie_{(S2.sid=T2.sid)} T2 \bowtie_{(T2.bookno=B2.bookno)} B2)q2))$$

Step 3

Keep only required attributes and leave out the tables not needed

$$(\pi_{(S.sid,S.sname,T.bookno)}(S \bowtie_{(S.sid=T.sid)} T))$$

$$-$$

$$(\pi_{(q1.sid,q1.sname,q1.bookno)}(\pi_{(S1.sid,S1.sname,B1.bookno,B1.price)}(S1 \bowtie_{(S1.sid=T1.sid)} T1 \bowtie_{(T1.bookno=B1.bookno)} B1)q1)$$
$$\bowtie_{(q1.sid=q2.sid \wedge q1.price<q2.price)} (\pi_{(S2.sid,S2.sname,B2.bookno,B2.price)}(S2 \bowtie_{(S2.sid=T2.sid)} T2 \bowtie_{(T2.bookno=B2.bookno)} B2)q2))$$

Q.6. Part (a)

$$\pi_{(B.bookno,B.title)}(\pi_{(M.sid,B.bookno,B.title)}(\sigma_{(M.major='CS' \vee M.major='Math')}M \times B)$$

$$-$$

$$\pi_{(M1.sid,T1.bookno,B1.title)}(\sigma_{(M1.sid=T1.sid \wedge T1.bookno=B1.bookno) \wedge (M.major='CS' \vee M.major='Math')}T1 \times M1 \times B1))$$

Q.6. Part (b)

Step 1

$$\pi_{(B.bookno,B.title)}(\pi_{(M.sid,B.bookno,B.title)}(\sigma_{(M.major='CS' \vee M.major='Math')}M \times B)$$

$$-$$

$$\pi_{(M1.sid,T1.bookno,B1.title)}(\sigma_{(M1.sid=T1.sid \wedge T1.bookno=B1.bookno) \wedge (M.major='CS' \vee M.major='Math')}M1 \times T1 \times B1))$$

Step 2

Using natural joins instead of cross joins

$$\pi_{(B.bookno,B.title)}(\pi_{(M.sid,B.bookno,B.title)}(\sigma_{(M.major='CS' \vee M.major='Math')}M \times B)$$

$$-$$

$$\pi_{(M1.sid,T1.bookno,B1.title)}(M1 \bowtie_{(M1.sid=T1.sid \wedge (M1.major='CS' \vee M1.major='Math'))} T1 \bowtie_{(T1.bookno=B1.bookno)} B1))$$

Step 3

Pushing selections and projections before joining

$$\pi_{(B.bookno,B.title)}(\pi_{(M.sid,B.bookno,B.title)}(\sigma_{(M.major='CS' \vee M.major='Math')} M \times B)$$

$$-$$

$$\pi_{(M1.sid,T1.bookno,B1.title)}(\sigma_{(M.major='CS' \vee M.major='Math')} M1 \bowtie_{(M1.sid=T1.sid)} T1 \bowtie_{(T1.bookno=B1.bookno)} B1))$$

Part 3

Q.7.

Query 3

```
select distinct r1.a
from R r1, R r2, R r3
where r1.b = r2.a and r2.b = r3.a;
```

Part (a)

Query 4

```
select distinct r1.a
from R r1 natural join
(select distinct r2.a as b from R r2 natural join
(select distinct r3.a as b from R r3) r3) r2;
```

Part (b)

makerandomR	Query 3(in ms)	Query 4(in ms)
(500,500,10)	0.12	0.306
(100,100,1000)	19.194	2.829
(200,200,4000)	318	20.7
(1000,1000,10000)	231.075	42.351
(500,500,25000)	16670	311
(1000,1000,50000)	21383.971	552.831
(1000,1000,100000)	--	3303

Part (c)

1. Query 4 is optimized, hence it takes less time than Query 3.
2. Query Q3 has O(n^3) time complexity because of two cross joins. Whereas, query Q4 has O(n) time complexity because natural join uses hashing. Because of this, the last query in Q3 had to be terminated.
3. As the data increases, the running time increases for both the queries

Q.8.

Query 5

```
select ra.a
from Ra ra
where not exists (select r.b
from R r
where r.a = ra.a and
r.b not in (select s.b from S s));
```

Part (a)

Query 6

```
select * from Ra ra
except
select * from Ra ra natural join
(select q1.a from
(select r1.a,r1.b from R r1
except
select r2.a,s1.b from R r2 natural join S s1) q1) q2;
```

Part (b)

makerandomR	makerandomS	Query 5(in ms)	Query 6(in ms)
(500,500,5000)	(500,5000)	1.561	6.013
(1000,1000,10000)	(1000,10000)	2.6	8.02
(1000,2000,10000)	(2000,10000)	2.95	12.138
(1000,5000,10000)	(5000,10000)	3.488	16.12
(2000,2000,40000)	(2000,40000)	10.919	51.369
(2000,2000,80000)	(2000,80000)	17.162	85.329

Part (c)

1. As we can see from the above table Query 5 takes lesser time than Query 6, hence it outperforms it.
2. Use of natural join and optimization did not help in this case.
3. Although, the ratio of running times remains same in all the cases. Q5:Q6 is around 1:4. Therefore the O() complexity has remained the same.

Q.9.

Query 7

```
select ra.a
from Ra ra
where not exists (select s.b
from S s
where s.b not in (select r.b
from R r
where r.a = ra.a));
```

Part (a)

Query 8

```
select ra.a
from Ra ra
except
select q.a from
(select ra.a,s.b
from S s cross join Ra ra
except
select ra.a,s.b
from S s natural join R r natural join Ra ra)q;
```

Part (b)

makerandomR	makerandomS	Query 7(in ms)	Query 8(in ms)
(500,500,5000)	(500,5000)	198	205.7
(1000,1000,10000)	(1000,10000)	704	677
(1000,2000,10000)	(2000,10000)	749	745.1
(1000,5000,10000)	(5000,10000)	991	774
(2000,2000,40000)	(2000,40000)	5914	3525.903
(2000,2000,80000)	(2000,80000)	12364	4457.523

Part (c)

1. It can be seen that Query 8 performs better than Query 7 as the data is increased.
2. As the data increases, the running time increases for both the queries, but it is quadratic for Query 7 as well as Query 8.
3. Query 7 uses sequential scan but it doesn't use hashing, hence it is slower.
4. Query 8 uses cross join, but with hashing, hence it is faster

Part (d)

1. Between Query 5 and Query 8
2. Query 8 takes exponentially more time than Query 5 in Problem No. 8 and Problem No. 9.
3. Hence, we can satisfactorily conclude that ONLY set query is faster than the ALL set query.
4. ONLY set setjoin is faster than ALL set semijoin.

Q.10.

Part (a)

Query 9

```
with  NestedR as (select  r.a, array_agg(r.b order by 1) as Bs
from    R r
group by (r.a)),
SetS as (select array(select s.b from S s order by 1) as Bs)
select r.a
from   NestedR r, SetS s
where  r.Bs <@ s.Bs
union
(select ra.a
 from Ra ra
 except
 select r.a
 from   R r);
```

Explain briefly how query Q9 works and how it solves the problem.

Query 9's working is explained below:

```
'NestedR' is a view which returns 'a' and its corresponding 'b' as an array from the table 'R'.
'SetS' is a view which returns all the records in the table 'S' as an array.
It takes all the elements from 'a' in 'R' for which corresponding 'b' does not exist in 'S'.
It creates 2 sets where 'a' with corresponding 'b' from one set equals to 'b' of the other set.
It includes all 'a' from 'R' where array('b') from 'R' is a subset of array('b') from 'S'.
```

makerandomR	makerandomS	Query 5(in ms)	Query 6(in ms)	Query 9(in ms)
(500,500,5000)	(500,5000)	1.561	6.013	6.443
(1000,1000,10000)	(1000,10000)	2.6	8.02	19.903
(1000,2000,10000)	(2000,10000)	2.95	12.138	28.544
(1000,5000,10000)	(5000,10000)	3.488	16.12	34.866
(2000,2000,40000)	(2000,40000)	10.919	51.369	106.838
(2000,2000,80000)	(2000,80000)	17.162	85.329	202.644

Part (b)

1. Query 9 and Query 6 have similar running times. They are asymptotically the same.
2. Query 5 runs in O(n) time and is faster than both Query 9 and Query 6.
3. NOT EXISTS and NOT IN performs better than than pure SQL and set objects method.

Q.11.

Part (a)

Query 10

```
with NestedR as (select r.a, array_agg(r.b order by 1) as Bs
from R r
group by (r.a)),
SetS as (select array(select s.b from S s order by 1) as Bs)
select r.a
from NestedR r, SetS s
where s.Bs <@ r.Bs;
```

Explain briefly how query Q10 works and how it solves the problem.

Query 10's working is explained below:

```
Similar to the previous question,
'NestedR' is a view which returns 'a' and its corresponding 'b' as an array from the table 'R'.
'SetS' is a view which returns all the records in the table 'S' as an array.
We take all 'a' from R where S(b) array is a subset of R(b) array.
```

makerandomR	makerandomS	Query 7(in ms)	Query 8(in ms)	Query 10(in ms)
(500,500,5000)	(500,5000)	198	205.7	1.35
(1000,1000,10000)	(1000,10000)	704	677	7.9
(1000,2000,10000)	(2000,10000)	749	745.1	9.67
(1000,5000,10000)	(5000,10000)	991	774	12.073
(2000,2000,40000)	(2000,40000)	5914	3525.903	29.002
(2000,2000,80000)	(2000,80000)	12364	4457.523	61.517

Part (b)

Additional tests for Query 8 and Query 10

makerandomR	makerandomS	Query 8(in ms)	Query 10(in ms)
(100,100,1000)	(100,1000)	8.5	0.56
(500,500,25000)	(500,25000)	208	12.6
(1000,1000,100000)	(1000,100000)	1139	58.7
(2000,2000,400000)	(2000,400000)	3815	213.789
(5000,5000,2500000)	(5000,2500000)	34856	1466

Part (c)

1. We can see from the first table that Query 10 out performs Query 7 as well as Query 8 by a huge margin.
2. Using object relational database methods improves performance as compared to Pure SQL and set predicates methods.
3. This implementation of ALL set semijoin uses arrays and set of objects, which works very well

```
In [ ]:
```