Assignment 6

Harsha Raja Shivakumar

March 31, 2020

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 specically, consider the following types of queries using the [NOT] IN set predicate.**

   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

   Given Query

   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))

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

X1 = $\pi_{R1.r}(\sigma_{C1(R1)}(R1)) - (\pi_{S1.B1}(S1 \ltimes_{C2(S1,R2)} R2)[\cup|\cap|-]\pi_{T1.C1}(T1 \ltimes_{C3(T1,R3)} R3))$

$\pi_{R1.*}(R1 \ltimes X1)$

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

X1 = $\pi_{R1.r}(\sigma_{C1(R1)}(R1)) \bowtie (\pi_{S1.B1}(S1 \ltimes_{C2(S1,R2)} R2)[\cup|\cap|-]\pi_{T1.C1}(T1 \ltimes_{C3(T1,R3)} R3))$

$\pi_{R1.*}(R1 \ltimes X1)$

For "NOT IN" General case

X1 = $\pi_{R1.r,....R1.rk}(\sigma_{C1(R1....Rk)}(R1....Rk)) - (\pi_{S1.B1}(S1.....Sm \ltimes_{C2(S1.....Sm,R1.....Rk)} R1.....Rk)[\cup| \cap |-]\pi_{T1.C1}(T1.....Tn \ltimes_{C3(T1.....Tn,R1.....Rk)} R1.....Rk))$

$\pi_{R1.....Rk*}(R1.....Rk \ltimes X1)$

For "IN" General case

X1 = $\pi_{R1.r,....R1.rk}(\sigma_{C1(R1....Rk)}(R1....Rk)) \bowtie (\pi_{S1.B1}(S1.....Sm \ltimes_{C2(S1.....Sm,R1.....Rk)} R1.....Rk)[\cup| \cap |-]\pi_{T1.C1}(T1.....Tn \ltimes_{C3(T1.....Tn,R1.....Rk)} R1.....Rk))$

$\pi_{R1.....Rk*}(R1.....Rk \ltimes X1)$

(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

Given 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))

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

X1 = $\pi_{R1.r}(\sigma_{C1(R1)}(R1)) - (\pi_{S1.B1}(\sigma_{C2(S1)}S1[\cap| \cup |-]\pi_{T1.C1}(\sigma_{C3(T1)}T1)))$

$\pi_{R1.*}(R1 \ltimes X1)$

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

X1 = $\pi_{R1.r}(\sigma_{C1(R1)}(R1)) \bowtie (\pi_{S1.B1}(\sigma_{C2(S1)}S1[\cap|\cup|-]\pi_{T1.C1}(\sigma_{C3(T1)}T1)))$

$\pi_{R1.*}(R1 \ltimes X1)$

2. **Let R be a relation with schema (a; b; c) and let S be a relation with schema (a; b; d). You may assume that the domains for the attributes a, b, and c are the same.**

3. **Find the sid and name of each student who majors in 'CS' and who bought a book that cites a higher priced book. "**

   (a) Using the translation algorithm presented in class, translate this SQL into and equivalent RA expression formulated in the standard RA syntax

   Given Query

   select s.sid, s.sname from student s where s.sid in (select m.sid from major m where m.major = 'CS') and exists (select 1 from cites c, book b1, book b2 where (s.sid,c.bookno) in (select t.sid, t.bookno from buys t) and c.bookno = b1.bookno and c.citedbookno = b2.bookno and b1.price < b2.price);

   Converting the given query to Pure SQL

   Removing In and Exists and converting to natural join and join

   select distinct s.sid,s.sname from student s natural join buys t natural join cites c natural join book b1 join book b2 on c.citedbookno = b2.bookno join major m on m.sid = t.sid where m.major = 'CS' and b1.price < b2.price;

   $\pi_{s.sid,s.sname}((S \ltimes T \bowtie C \bowtie B1) \bowtie_{C.citedbookno=B2.bookno} B2$
   $\bowtie_{M.major='CS' \wedge b1.price<b2.price \wedge M.sid=T.sid} M)$

   (b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression.

   Optimized SQL query

select distinct s.sid,s.sname from student s natural join buys t natural join cites c natural join (select bookno, price from book) b1 join (select bookno, price from book)b2 on (c.citedbookno = b2.bookno and b1.price < b2.price) natural join (select sid from major where major = 'CS')x;

Step 1

we push selections and projections in where possible

$\pi_{s.sid,s.sname}((S \ltimes T \bowtie C \bowtie B1) \bowtie_{C.citedbookno=B2.bookno \wedge b1.price<b2.price} B2 \bowtie (\sigma_{M.major='CS'}(M)))$

Step 2

Keeping only required attributes for joins

$\pi_{s.sid,s.sname}((S \ltimes T \bowtie C \bowtie \pi_{B1.bookno,B1.price}(B1)) \bowtie_{C.citedbookno=B2.bookno \wedge b1.price<b2.price} \pi_{B2.bookno,B2.price}(B2) \bowtie \pi_{M.sid}(\sigma_{M.major='CS'}(M)))$

4. **Find the sid, name, and major of each student who does not major in 'CS',did not buy a book that less than $30, bought some book(s) that cost less than less than $50.**

   (a) Using the translation algorithm presented in class, translate this SQL into and equivalent RA expression formulated in the standard RA syntax

   Given Query

   select distinct s.sid, s.sname, m.major from student s, major m where s.sid = m.sid and s.sid not in (select m.sid from major m where m.major = 'CS') and s.sid <> ALL (select t.sid from buys t, book b where t.bookno = b.bookno and b.price < 30) and s.sid in (select t.sid from buys t, book b where t.bookno = b.bookno and b.price < 60);

   Converting the given query to Pure SQL

   Removing all and in and converting to natural join and join

   select distinct s.sid, s.sname, m.major from student s natural join major m natural join (select t.sid from buys t except select t.sid from buys t join book b on t.bookno = b.bookno and b.price < 30)y natural join (select distinct t.sid from buys t join book b on t.bookno = b.bookno and b.price < 50 except select sid from major where major = 'CS')x;

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

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression.

Optimized SQL query

select distinct s.sid, s.sname, m.major from student s natural join major m natural join (select t.sid from buys t except select t.sid from buys t join (select b.bookno,b.price from book b)b on t.bookno = b.bookno and b.price < 30)y natural join (select distinct t.sid from buys t join (select b.bookno,b.price from book b)b on t.bookno = b.bookno and b.price < 50 except select sid from major where major = 'CS')x;

Since all the attributes are pushed in already and foreign key constraints are satisfied only optimization possible here is keeping only required attributes for joins

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

5. **Find each (s; n; b) triple where s is the sid of a student, n is the name of this student, and where b is the bookno of a book whose price is the most expensive among the books bought by that student.**

   (a) Using the translation algorithm presented in class, translate this SQL into and equivalent RA expression formulated in the standard RA syntax

   Given Query

   select distinct s.sid, s.sname, b.bookno from student s, buys t, book b where s.sid = t.sid and t.bookno = b.bookno and b.price >= ALL (select b.price from book b where (s.sid,b.bookno) in (select t.sid, t.bookno from buys t));

   Converting the given query to Pure SQL

   Removing all and in and converting to natural join and join

   select s.sid, s.sname, x.bookno from student s natural join (select t.sid, t.bookno from buys t except (select t.sid, t.bookno from (buys t natural join

book b) join (buys t1 natural join book b1) on (t.sid = t1.sid and b.price < b1.price)))x;

$\pi_{S.sid,S.sname,B.bookno}(S \bowtie (\pi_{T.sid,T.bookno}(T) - (\pi_{T.sid,T.bookno}((T \bowtie B) \bowtie_{T.sid=T1.sid \wedge B.price<B1.price} (T1 \bowtie B1)))))$

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression.

Optimized SQL query

with books as (select bookno, price from book) select s.sid, s.sname, x.bookno from student s natural join (select t.sid, t.bookno from buys t except (select t.sid, t.bookno from (buys t natural join books b) join (buys t1 natural join books b1) on (t.sid = t1.sid and b.price < b1.price)))x;

Since all the attributes are pushed in already and foreign key constraints are satisfied only optimization possible here is keeping only required attributes for joins

X1 = $\pi_{B.bookno,B.price}(B)$
$X2 = X1$
$\pi_{S.sid,S.sname,B.bookno}(S \bowtie (\pi_{T.sid,T.bookno}(T) - (\pi_{T.sid,T.bookno}((T \bowtie X1) \bowtie_{T.sid=T1.sid \wedge X1.price<X2.price} (T1 \bowtie X2)))))$

6. **Find the bookno and title of each book that is not bought by all students who major in both 'CS' or in 'Math'.**

   (a) Using the translation algorithm presented in class, translate this SQL into and equivalent RA expression formulated in the standard RA syntax

   Given Query

   select b.bookno, b.title from book b where exists (select s.sid from student s where s.sid in (select m.sid from major m where m.major = 'CS' UNION select m.sid from major m where m.major = 'Math') and s.sid not in (select t.sid from buys t where t.bookno = b.bookno));

   Converting the given query to Pure SQL

Removing exists, in and, not in and converting to natural join and join

select distinct b.bookno, b.title from book b natural join (select s.sid,b.bookno from student s cross join book b natural join (select sid from major where major = 'CS' union select sid from major where major = 'Math')x except select * from buys t)y;

$\pi_{B.bookno,B.title}(B \ltimes (\pi_{S.sid,B.bookno}(SXB \bowtie (\pi_{M.sid}(\sigma_{M.major='CS'}(M) \cup \sigma_{M.major='Math'}(M))) - \pi_{T.sid,T.bookno}(T))))$

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression.

Optimized SQL query

select distinct b.bookno, b.title from book b natural join (select s.sid,b.bookno from (select s.sid from student s)s cross join (select b.bookno from book b)b natural join (select sid from major where major = 'CS' union select sid from major where major = 'Math')x except select * from buys t)y;

Since all the attributes are pushed in already and foreign key constraints are satisfied only optimization possible here is keeping only required attributes for joins

$\pi_{B.bookno,B.title}(B \ltimes (\pi_{S.sid,B.bookno}(\pi_{S.sid}(S)X\pi_{B.bookno}(B) \bowtie (\pi_{M.sid}(\sigma_{M.major='CS'}(M) \cup \sigma_{M.major='Math'}(M))) - \pi_{T.sid,T.bookno}(T))))$

7. **Now consider query Q3:**

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

   (a) Translate and optimize this query and call it Q4. Then write Q4 as an SQL query with RA operations query just as was done for query Q2.

   select distinct r1.a from r r1 join r r2 on r1.b = r2.a join (select distinct r3.a from r r3)x on r2.b = x.a;

   (b) Compare queries Q3 and Q4 in a similar way as we did for Q1 and Q2. You should experiment with dierent sizes for R. Incidentally, these

relations do not need to use the same m,n, and l parameters as those shown in the above table for Q1 and Q2.

makerandomR	Q3 (in ms)	Q4 (in ms)
(100,100,1000)	66.5	8
(500,500,25000)	25150.209	670
(1000,1000,100000)	- -	6049
(2000,2000,400000)	- -	41114

(c) What conclusions can you draw from the results of these experiments?

As we can see from the above table Q4 outperforms Q3. Hence optimization of Q3 worked really well in this case. The ratio of Q4:Q3 is around 1:6. Using Q3 only first two tuples were completed in reasonable time. But the last two tuples had to be aborted. But Q4 was successful in running all the tuples.

8. **Now consider query Q5 which is an implementation of the ONLY set semijoin between R and S. (See the lecture on set semijoins for more information.) (Incidentally, if you look at the code for makerandomR you will see a rela- tion Ra that provides the domain of all a values. You will need to use this relation in the queries. Analogously, in the code for makerandomS you will see the relation Sb that contains the domain of all b values.) In SQL, Q5 can be expressed as follows:**

```
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));
```

(a) Translate and optimize this query and call it Q6. Then write Q6 as an SQL query with RA operations just as was done for Q2 above.

– Method 1
select ra.a from Ra ra except select q2.a from Ra ra natural join (select q1.a from (select * from r except select r.a,r.b from r natural join s) q1) q2;

– Method 2
select q1.a from (select ra.a from ra except select q2.a from (select ra.a,r.b from ra natural join r except select ra.a,r.b from r natural join s cross join ra)q2)q1;

– Method 3
with r as (select r.a, r.b from r r except select r.a, r.b from r r natural join s s) select q1.a from (select distinct ra.a from ra ra except select ra.a from ra ra natural join r r)q1;

Method 1 and 3 was giving best results

(b) Compare queries Q5 and Q6 in a similar way as we did for Q1 and Q2. You should experiment with dierent sizes for R and S. (Vary the size of S from smaller to larger.) Also use the same value for the parameter n in makerandomR(m; n; l) and makerandomS(n; l) so that the maximum number of b values in R and S are the same.

makerandomR	Q5 (in ms)	Q6 (in ms)
(100,100,1000)	1.2	4.8
(500,500,25000)	18	55
(1000,1000,100000)	45	194
(2000,2000,400000)	140	810

(c) What conclusions can you draw from the results of these experiments?

As we can see from the above table Q5 outperforms Q6. Hence converting the query to pure SQL and optimizing it did not help in this case. The ratio of Q5:Q6 is around 1:3. Hence we can conclude that using not exists and not in is helpful in this case.

9. **Now consider query Q7 which is an implementation of the ALL set semijoin between R and S. (See the lecture on set semijoins for more information.) In SQL, Q7 can be expressed as follows:**

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));

(a) Translate and optimize this query and call it Q8. Then write Q8 as an SQL query with RA operations just as was done for query Q2 above.

select q1.a from (select ra.a from ra except select q2.a from (select ra.a, s.b from ra cross join s except select ra.a, s.b from ra natural join r natural join s)q2)q1;

(b) Compare queries Q7 and Q8 in a similar way as we did for Q1 and Q2. You should experiment with different sizes for R and S. (Vary the size of S from smaller to larger.) Also use the same value for the parameter n in makerandomR(m; n; l) and makerandomS(n; l) so that the maximum number of b values in R and S are the same.

makerandomR	Q7 (in ms)	Q8 (in ms)
(100,100,1000)	18	20
(500,500,25000)	1500	700
(1000,1000,100000)	12200	3000
(2000,2000,400000)	99200	12050

(c) What conclusions can you draw from the results of these experiments?

As we can see from the above table Q8 outperforms Q7. Hence we can say that optimization worked really well in this case. The time taken by Q7 exponentially increases when compared with Q8. If any new high dataset is used to run this query Q7 might not be able to run this in reasonable time.

(d) Furthermore, what conclusion can you draw when you compare you experiment with those for the ONLY set semijoin in problem 8?

As we can see from Q5 and Q8 which is fastest for Question 8 and Question 9, Q8 takes exponentially more time than Q5. Hence we can say that ONLY set query is faster than ALL set query. This can also be proved in general sense, ONLY set semijoin is faster than ALL set semijoin.

10. **Reconsider problem 8. We can also solve this problem by using a query wherein set objects are created (see Lecture 15). Below we show this**

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

```
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);
```

Above query tries to extract all the elements from 'a' in 'R' for which corresponding 'b' does not exists in 'S'. This is done by creating 2 arrays, one where a with corresponding array of b's are considered and the other with array of b's. We take all 'a' where R(b) array is a subset of S(b) array.

(a) Now, using the same experimental data as in question 8, show the execution time of running this query and compare those with the ones obtained for queries Q5 and Q6.

makerandomR	Q5 (in ms)	Q6 (in ms)	Q9 (in ms)
(100,100,1000)	1.2	4.8	6
(500,500,25000)	18	55	90
(1000,1000,100000)	45	194	500
(2000,2000,400000)	140	810	3100

(b) What conclusions can you draw from the results of these experiments?

As we can see from the above table Q5 still outperforms both Q6 and Q9. Usage of NOT EXISTS and NOT IN in this query is better than pure SQL and set objects method.

11. **Reconsider problem 9. We can also solve this problem by using a query wherein set objects are created. Below we show this query. Call this query Q10. (We assume that S != null ;) Explain briefly how query Q10 works and how it solves the problem.**

```
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;
```

Above query tries to extract all the elements from 'a' in 'R' for which corresponding 'b' has all the elements as 'b' in 'S'. This is done by creating 2 arrays, one where a with corresponding array of b's are considered and the other with array of b's. We take all 'a' where S(b) array is a subset of R(b) array.

(a) Now, using the same experimental data as in question 9, show the execution time of running this query and compare those with the ones obtained for queries Q7 and Q8

makerandomR	Q7 (in ms)	Q8 (in ms)	Q10 (in ms)
(100,100,1000)	18	20	2.2
(500,500,25000)	1500	700	35
(1000,1000,100000)	12200	3000	165
(2000,2000,400000)	99200	12050	540

(b) You should also run additional experiments that only compare query Q8 and Q10.

makerandomR	Q8 (in ms)	Q10 (in ms)
(100,100,1000)	20	2.2
(250,250,5000)	100	8
(500,500,25000)	700	35
(750,750,50000)	1600	60
(1000,1000,100000)	3000	165
(2000,2000,400000)	12050	540

(c) What conclusions can you draw from the results of these experiments?

As we can see from the above table Q10 out performs both Q7 and Q8. Using set objects methods decreases the run time exponentially when compared with Pure SQL and set predicates methods. Since this problem involves subsetting the arrays, set objects works pretty well in this case.