Assignment 6: Query Translation and Optimization with solutions Object-Relational Databases

Due Thursday, March 21 2019 by 11:45pm

For this assignment you will need the material covered in the lectures on Translating SQL to RA, RA query optimizations, and object-relational databases (Lectures 14, 15, and 16).

For this assignment, you will need to submit 2 files:

1. One such file is a .sql file that contains the SQL code relating to problems that requires such code.
2. A second file with .pdf extension that contains your solutions for problems where RA expressions are requested. This .pdf file should also contain the answer to problems that require essay answers. No handwritten documents can be submitted.

1 Theoretical Problems

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, nor 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] EXISTS 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] EXISTS set predicate.

```
SELECT L(r1,...,rk)
FROM   R1 r1, ..., Rk rk
WHERE  C1(r1,...,rk) AND
              [NOT] EXISTS (SELECT DISTINCT 1
                            FROM   S1 s1,..., S1 sm
                            WHERE  C2(s1,...,sm,r1,...,rk)
                            [UNION|INTERSECT|EXCEPT]
                            SELECT DISTINCT 1
                            FROM   T1 t1, ..., Tn tn
                            WHERE  C3(t1,...,tn,r1,...,rk))
```

   Notice that there are six cases to consider:

   (a) EXISTS (... UNION ...)

   (b) EXISTS (... INTERSECT ...)

(c) EXISTS (... EXCEPT ...)
(d) NOT EXISTS (... UNION ...)
(e) NOT EXISTS (... INTERSECT ...)
(f) NOT EXISTS (... EXCEPT ...)

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.

To get practice, first consider the following special case where $k = 1$, $m = 1$, and $n = 1$. I.e., the following case: [1]

```
SELECT L(r1)
FROM   R1 r1
WHERE  C1(r1) [NOT] EXISTS (SELECT DISTINCT 1
                            FROM   S1 s1
                            WHERE  C2(s1,r1)
                            [UNION|INTERSECT|EXCEPT]
                            SELECT DISTINCT 1
                            FROM   T1 t1
                            WHERE  C3(t1,r1))
```

Solution (EXISTS case)

```
SELECT DISTINCT L(r1)
FROM   (SELECT r1.* FROM R1 r1 WHERE C1(r1)) r1
             NATURAL JOIN
            (SELECT r2.*, s1.*
             FROM   R2 r2 JOIN S1 s1 ON C2(s1,r2)
             [UNION|INTERSECT|EXCEPT]
             SELECT r3.*, t1.*
             FROM   R3 r3 JOIN T1 t1 ON C3(t1,r3)) q
```

$$\pi_{L(R_1)}(\sigma_{C_1}(R_1) \bowtie (R_2 \bowtie_{C_2(S_1,R_2)} S_1 \, [\cup \mid \cap \mid -] \, R_3 \bowtie_{C_3(T_1,R_3)} T_1))$$

or, equivalently

$$\pi_{L(R_1)}(\sigma_{C_1}(R_1) \ltimes (R_2 \bowtie_{C_2(S_1,R_2)} S_1 \, [\cup \mid \cap \mid -] \, R_3 \bowtie_{C_3(T_1,R_3)} T_1))$$

Remark: In the case of UNION, we can do better

$$\pi_{L(R)}(\sigma_{C_1}(R_1) \bowtie_{C_2(S_1,R_1)} S_1) \cup \pi_{L(R)}(\sigma_{C_1}(R_1) \bowtie_{C_3(T_1,R_1)} T_1)$$

The solution for the NOT EXISTS set predicate is omitted

[1]Once you can handle this case, the general case is a very similar.

Solution (General EXISTS case)

$$\pi_{L(R_1,\ldots,R_k)}(\sigma_{C_1}(\mathbf{R}_1) \ltimes (\mathbf{R}_2 \bowtie_{C_2(\mathbf{S},\mathbf{R}_2)} \mathbf{S}\,[\cup \mid \cap \mid -]\,\mathbf{R}_3 \bowtie_{C_3(\mathbf{T},\mathbf{R}_3)} \mathbf{T}))$$

where

$$\begin{aligned}\mathbf{R} &= R_1 \times \cdots \times R_k\\ \mathbf{S} &= S_1 \times \cdots \times S_m\\ \mathbf{T} &= T_1 \times \cdots \times T_n\end{aligned}$$

Remark: In the case of UNION, we can do better

$$\pi_{L(R_1,\ldots,R_k)}(\sigma_{C_1}(\mathbf{R}) \bowtie_{C_2(\mathbf{S},\mathbf{R})} \mathbf{S}) \cup \pi_{L(R_1,\ldots,R_k)}(\sigma_{C_1}(\mathbf{R}) \bowtie_{C_3(\mathbf{T},\mathbf{R})} \mathbf{T})$$

The solution for the NOT EXISTS case is omitted

2. Let R be a relation with schema (a, b, c) and let S be a relation with schema (d, e).

   Prove the correctness of the following rewrite rule where a project π is pushed over a join $\bowtie$:

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))$$

   Solution: We translate the RA expression in Predicate Logic and we use logical equivalences

$$\begin{aligned}\pi_{a,d}(R \bowtie_{c=d} S) &= \{(a,d) \mid \exists b \exists c \exists e(R(a,b,c) \wedge c = d \wedge S(d,e))\}\\ &= \{(a,d) \mid \exists b \exists c(R(a,b,c) \wedge c = d \wedge \exists e S(d,e))\}\\ &= \{(a,d) \mid \exists c((\exists b R(a,b,c)) \wedge c = d \wedge (\exists e S(d,e)))\}\\ &= \{(a,d) \mid \exists c((a,c) \in \pi_{a,c}(R) \wedge c = d \wedge (d \in \pi_d(S)))\}\\ &= \{(a,d) \mid \exists c((a,c,d) \in \pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))\}\\ &= \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))\end{aligned}$$

3. Consider the same rewrite rule

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))$$

   as in problem 2.

   Furthermore assume that S has a primary key d and that R has foreign key c referencing this primary key in S.

   How can you simplify this rewrite rule? Argue why this rewrite is correct.

   The solution for this problem is omitted.

2 Translating and Optimizing SQL Queries to Equivalent RA Expressions

4. Using the translation algorithm presented in class, translate each of the following SQL queries into equivalent RA expressions. Provide this RA expression in your .pdf file.

   Then using the query optimization rewrite rules, optimize each of these RA expressions into an equivalent but optimized RA expression.

   You are required to specify some, but not necessarily all, of the intermediate steps that you applied during the translations and optimizations. Use your own judgment to specify the most important steps.

   During the optimization, take into account the primary keys and foreign key constraints that are assumed for the Student, Book, Buys, Major, and Cites relations.

   (a) Find the sid and name of each student who majors in 'CS' and who bought a book that is cited by a lower priced book.

   ```
   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   buys t, cites c, book b1, book b2
                  where  s.sid = t.sid and t.bookno = c.citedbookno and
                         c.citedbookno = b1.bookno and c.bookno = b2.bookno and
                         b1.price > b2.price);
   ```

   Let

   $$E \quad = \quad (S \bowtie \sigma_{major='CS'}(M) \bowtie T) \bowtie_{T.bookno=C.citedbookno} C \bowtie_{C.citedbookno=B_1.bookno} B_1$$

   $$\pi_{sid,sname}(E \bowtie_{C.bookno=B_2.bookno \wedge B_1.price>B_2.price} B_2)$$

   We can now start optimizing by pushing selections and projections down:

   $$\begin{aligned} CS &= \pi_{sid}(Major) \\ T &= \pi_{sid,sname,bookno}(S \bowtie CS \bowtie Buys) \\ B &= \pi_{bookno,price}(Book) \\ C &= \pi_{citedbookno}(Cites \bowtie_{citedbookno=B_1.bookno} B_1 \bowtie_{cites.bookno=B_2.bookno \wedge B_1.price>B_2.price} B_2) \end{aligned}$$

   With these expressions we get to the following optimized expression

   $$\pi_{sid,sname}(T \bowtie_{t.bookno=c.citedbookno} C).$$

   (b) Find the sid, name, and major of each student who

   - does not major in 'CS',

- bought a book that cost more that \$30,
- did not buy any books that cost more than \$50.

```
select distinct s.sid, m.major
from   student s, major m
where  s.sid = m.sid and m.major <> 'CS' and
        s.sid = SOME (select t.sid
                      from   buys t, book b
                      where  t.bookno = b.bookno and b.price > 30) and
        s.sid not in (select t.sid
                      from   buys t, book b
                      where  t.bookno = b.bookno and b.price > 50);
```

The solution for this problem is omitted.

(c) Find each (s, b) pair where s is the sid of a student and where b is the bookno of a book whose price is the cheapest among the books bought by that student.

```
select distinct t.sid, b.bookno
from   buys t, book b
where  t.bookno = b.bookno and
       b.price <= ALL (select b1.price
                       from   buys t1, book b1
                       where t1.bookno = b1.bookno and t1.sid = t.sid);
```

Solution: For the translation see the .sql file. After optimization, we get the following expression

$$B \quad = \quad \pi_{bookno,price}(Book)$$

The final expression is

$$\pi_{T.sid,B.bookno}(T \bowtie B - \pi_{T.*,B.*}((T \bowtie B) \bowtie_{T.sid=T_1.sid \wedge B.price>B_1.price} (T_1 \bowtie B_1))$$

(d) Find the bookno of each book that is bought by all students who major in both ‘CS’ and in ‘Math’.

```
select b.bookno
from   book b
where  not exists (select s.sid
                   from   student s
                   where  s.sid in (select m.sid from major m
                                    where m.major = 'CS'
                                    INTERSECT
                                    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));
```

The solution for this problem is omitted.

3 Experiments to Test the Effectiveness of Query Optimization

No solutions provided for this section.

In the following problems, you will conduct experiments to gain insight into whether or not query optimization works. In other words, can it be determined experimentally if optimizing an SQL or an RA expression improves the time (and space) complexity of query evaluation?

You will need to use the PostgreSQL system to do you experiments. Recall that in SQL you can specify RA expression in a way that mimics it faithfully.

As part of the experiment, you might notice that PostgreSQL's query optimizer does not fully exploit all the optimization that is possible as discussed in Lecture 15.

In these following problems you will need to generate artificial data of increasing size and measure the time of evaluating non-optimized and optimized queries. The size of this data can be in the ten or hundreds of thousands of tuples. This is necessary because on very small data is it is not possible to gain sufficient insight into the quality (or lack of quality) of optimization.

Consider a binary relation `R(a int, b int)`. You can think of this relation as a graph, wherein each pair (a, b) represents and edge from a to b. (We work with directed graph. In other words edges (a, b) and (b, a) represent two different edges.) It is possible that `R` contains self-loops, i.e., edges of the form (a, a).

Besides the relation R we will also use a unary relation `S(b int)`

Along with this assignment, I have provided the code of two functions

`makerandomR(m integer, n integer, l integer)`

and

`makerandomS(n int, l int)`

```
create or replace function makerandomR(m integer, n integer, l integer)
returns void as
$$
declare  i integer; j integer;
begin
    delete from Ra; delete from Rb; delete from R;
    for i in 1..m loop insert into Ra values(i); end loop;
    for j in 1..n loop insert into Rb values(j); end loop;
    insert into R select * from Ra a, Rb b order by random() limit(l);
end;
$$ LANGUAGE plpgsql;

create or replace function makerandomS(n integer, l integer)
returns void as
$$
declare i integer;
begin
    delete from Sb; delete from S;
    for i in 1..n loop insert into Sb values(i); end loop;
    insert into S select * from Sb order by random() limit (l);
```

```
end;
$$ LANGUAGE plpgsql;
```

When you run

```
select makerandomR(m,n,l);
```

for some values m, n, and k, this function will generate a random relation with l tuples that is subset of $[1, m] \times [1, n]$. For example,

```
select makerandomR(3,3,4);
```

might generate the relation R

a	b
2	1
3	3
2	3
3	1

But, when you call

```
select makerandomR(3,2,4)
```

again, it may now generate a different random relation such as

a	b
1	2
2	3
3	1
1	1

Notice that when you call

```
select makerandomR(1000,1000,1000000)
```

it will make the entire relation $[1, 1000] \times [1, 1000]$ consisting of one million tuples.

The function `makerandomS`$(\texttt{n}, \texttt{l})$ will generate a random set of size l that is a subset of $[1, n]$.

Now consider the following simple query Q_1:

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

This query can be optimized to the query Q_2:

```
select distinct r1.a
from   R r1 join (select distinct r2.a from R r2) r2 on (r1.b=r2.a)
where  r1.b = r2.a
```

Image that you have created a relation R using the function makerandomR. Then when you execute in PostgreSQL the following

```
explain analyze
select distinct r1.a
from   R r1, R r2
where  r1.b = r2.a;
```

the system will return its execution plan as well as the time to evaluate Q_1 measured in ms.

And, when you execute in PostgeSQL the following

```
select distinct r1.a
from   R r1 join (select distinct r2.a from R r2) r2 on (r1.b=r2.a)
where  r1.b = r2.a
```

the system will return its execution plan as well as the time to evaluate Q_2 measured in ms.

This permits us to compare the performance of the non-optimized query Q_1 with the optimized Q_2 for various-sized relation R.

Here are some of these comparisons for various different random relations R.

makerandomR	Q_1 (in ms)	Q_2 (in ms)
(100,100,1000)	4.9	1.5
(500,500,25000)	320.9	28.2
(1000,1000,100000)	2648.3	76.1
(2000,2000,400000)	23143.4	322.0
(5000,5000,2500000)	−	1985.8

The − symbol indicates that I had to stop the experiment because it was taken too long. (All the experiments where done on a MacBook pro.)

Notice the significant difference between the execution times of the non-optimized query Q_1 and the optimized query Q_2. So clearly, optimization works on query Q_1.

If you look at the execution plan of PostgreSQL for Q_1, you will notice that it does a double nested loop and it therefore is $O(|R|^2)$ whereas for query Q_2 it runs in $O(|R|)$. Clearly, optimization has helped significantly.[2]

5. Now consider query Q_3:

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

[2]It is actually really surprising that the PostgreSQL system did not optimize query Q_1 any better.

Optimize this query and call it Q_4. Then write Q_4 as an SQL query just as was done for query Q_1 and Q_2 above.

Compare queries Q_3 and Q_4 in a similar way as we did for Q_1 and Q_2. What conclusion can you draw from the results of these experiments?

You should experiment with different sizes for `R`. Incidentally, these relation do need not be the same as those shown in the above table for Q_1 and Q_2.

6. Now consider query Q_5 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 relation `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, Q_5 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));
```

   Optimize this query and call it Q_6. Then write Q_6 as an SQL query just as was done for query Q_1 and Q_2 above.

   Compare queries Q_5 and Q_6 in a similar way as we did for Q_1 and Q_2. What conclusion can you draw from the results of these experiments?

   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.

7. Now consider query Q_7 which is an implementation of the ALL set semijoin between `R` and `S`. (See the lecture on set semijoins for more information.)

   In SQL, Q_7 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));
```

   Optimize this query and call it Q_8. Then write Q_8 as an SQL query just as was done for query Q_1 and Q_2 above.

Compare queries Q_7 and Q_8 in a similar way as we did for Q_1 and Q_2. What conclusions can you draw from the results of these experiments?

Furthermore, what conclusion can you draw when you compare you experiment with those for the ONLY set semijoin in problem 6?

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.

8. Reconsider problem 6. We can also solve this problem by using a query wherein set objects are created. Below we show this query. Call this query Q_9.

   Explain briefly how query Q_9 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)
                     union
                     select  q.a, '{}' as Bs
                     from    (select a from ra
                             except
                             select distinct a from  R) q),
         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;
   ```

   Now, using the same experimental data as in question 6 show the evaluation time of running this query and compare those with the ones obtained for queries Q_5 and Q_6. What conclusions can you draw from the results of these experiments?

9. Reconsider problem 7. We can also solve this problem by using a query wherein set objects are created. Below we show this query. Call this query Q_{10}.

   Explain briefly how query Q_{10} 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)
                 union
                 select  q.a, '{}' as Bs
                 from    (select a from ra
                          except
                          select distinct a from  R) q),
      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
```

   Actually, if you know that $S \neq \emptyset$, then this query can be made more efficient as follows

```
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
```

   Now, using the same experimental data as in question 7 show the evaluation time of running this query and compare those with the ones obtained for queries Q_7 and Q_8. You should also run additional experiments that only compare query Q_8 and Q_{10}

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

4 Formulating Queries in the Object-Relational Model

See the .sql file for solutions.

In the following problems, use the data provided for the student, majors, book, cites, buys relations.

The purpose of this assignment is to work with complex-object relations and to use these to solve queries.

10. Consider the function `setunion` which computes the set union of two sets represented as arrays. Notice that this function is defined polymorphically.

```
create or replace function setunion(A anyarray, B anyarray) returns anyarray as
$$
with
```

```
     Aset as (select UNNEST(A)),
     Bset as (select UNNEST(B))
select array( (select * from Aset) union (select * from Bset) order by 1);
$$ language sql;
```

(a) In the style of the `setunion` function, write a function `setintersection` that computes the intersection of two sets.

(b) In the style of the `setunion` function, write a function `setdifference` that computes the set difference of two sets.

You will need to use these functions in the remaining problems.

You can also make use of the function `memberof` which checks if an object x is in a set S. (Again this function is defined polymorphically.)

```
create or replace function memberof(x anyelement, S anyarray)
   returns boolean as
$$
select x = SOME(S)
$$ language sql;
```

The solutions for this problem are omitted.

11. Consider the view `student_books(sid,books)` which associates with each student the set of booknos of books he or she buys.

```
create or replace view student_books as
   select s.sid, array(select t.bookno
                       from   buys t
                       where  t1.sid = s.sid order by bookno) as books
   from   student s order by sid;
```

(a) Define a view `book_students(bookno,students)` which associates with each book the set of sids of students who bought that book. Observe that there may be books that are not bought by any student.

(b) Define a view `book_citedbooks(bookno,citedbooks)` which associates with each bookno of a book the set of booknos of books cited by that book. Observe that there may be books that cite no books.

(c) Define a view `book_citingbooks(bookno,citingbooks)` which associates with each bookno the set of booknos of books that cite that book. Observe that there may be books that are not cited.

(d) Define a view `major_students(major,students)` which associates with each major the set of sids of students who have that major. (You can assume that each major has at least one student.)

(e) Define a view `student_majors(sid,majors)` which associates with each student sid the set of his or her majors. Observe that there can be students who have no major.

Test that each of these views work properly. You will need to use them in the subsequent problems. The solutions for this problem are omitted.

12. Using the above defined functions, views, and the `book` and `student` relations, specify the following queries in SQL.

    So observe that you are not permitted to use the buys, cites, and major relations in your queries. These relations are available but encapsulated inside the views.

    For example, a query such as

    ```
    select t.sid
    from   buys t, book b
    where  t.bookno = b.bookno and b.price < 50
    ```

    is **not** permitted. However, a query such as

    ```
    select s.sid
    from   student_books s, book b
    where  memberof(b.bookno, s.books) and b.price < 50
    ```

    is permitted. By the way, notice that these queries are equivalent.

    Solutions for problems 12.a, 12.c, 12.d, 12.e, 12.l can be found in the `2019-some-solutions.sql` file.

    The solutions for the other problems are omitted.

    (a) Find the bookno of each book that is cited by at least one book that cost less than \$50.

    (b) Find the bookno and title of each book that was bought by a student who majors in CS and in Math.

    (c) Find the bookno of each book that is cited by exactly one book.

    (d) Find the sid of each student who bought all books that cost more than \$50.

    (e) Find the sid of each student who bought no book that cost more than \$50.

    (f) Find the sid of each student who bought only books that cost more than \$50.

    (g) Find the sids of students who major in 'CS' and who did not buy any of the books bought by the students who major in 'Math'."

    (h) Find sid-bookno pairs (s, b) such that not all books bought by student s are books that cite book b.

    (i) Find sid-bookno pairs (s, b) such student s only bought books that cite book b.

    (j) Find the pairs (s_1, s_2) of sid of students that buy the same books.

    (k) Find the pairs (s_1, s_2) of sid of students that buy the same number of books.

    (l) Find the bookno of each book that cites all but 2 books. (In other words, for such a book, there exists exactly 2 books that it does not cite.)