Assignment 6: Query Translation and Optimization Object-Relational Databases

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

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, 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.

```
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))
```

   Notice that there are six cases to consider:

   (a) IN (... UNION ...)
   (b) IN (... INTERSECT ...)
   (c) IN (... EXCEPT ...)

(d) NOT IN (... UNION ...)

(e) NOT IN (... INTERSECT ...)

(f) NOT IN (... EXCEPT ...)

(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.

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) 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))
```

(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))
```

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

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.

Prove the correctness of the following rewrite rule:

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

In other words, you have to prove that

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

and

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

2 Translating and Optimizing SQL Queries to Equivalent RA Expressions

Using the translation algorithm presented in class, translate each of the following SQL queries into equivalent RA expressions. For each query, provide its corresponding RA expression in your .pdf file. This RA expression needs to be formulated in the standard RA syntax.

Then use rewrite rules to 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.

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

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

   (b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression. Specify this optimized RA expression in your .pdf file.

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.

```
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);
```

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

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression. Specify this optimized RA expression in your .pdf file.

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.

```
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));
```

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

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression. Specify this optimized RA expression in your .pdf file.

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

```
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));
```

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

(b) Use the optimization rewrite rules, including those that rely on constraints, to optimize this RA expression. Specify this optimized RA expression in your .pdf file.

3 Experiments to Test the Effectiveness of Query Optimization

In the following problems, you will conduct experiments to gain insight into whether or not query optimization can be effective. 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 14.

In the 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 insights 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
    drop table if exists Ra; drop table if exists Rb;
    drop table if exists R;
    create table Ra(a int); create table Rb(b int);
    create table R(a int, b int);

    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
    drop table if exists Sb;
    drop table if exists S;
    create table Sb(b int);
    create table S(b int);

    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

`makerandomR(m,n,l);`

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

`makerandomR(3,3,4);`

might generate the relation instance

R

a	b
2	1
3	3
2	3
3	1

But, when you call

`makerandomR(3,3,4)`

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

R

a	b
1	2
2	3
3	1
1	1

Notice that when you call

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

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

The function makerandomS(n, 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 translated and optimized to the query Q_2:

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

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 execution time to evaluate Q_1 measured in ms.

And, when you execute in PostgeSQL the following

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

the system will return its execution plan as well as the execution 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 query 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]

7. 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;
```

   (a) Translate and optimize this query and call it Q_4. Then write Q_4 as an SQL query with RA operations query just as was done for query Q_2.

   (b) Compare queries Q_3 and Q_4 in a similar way as we did for Q_1 and Q_2.

   You should experiment with different 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 Q_1 and Q_2.

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

8. 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));
```

   (a) Translate and optimize this query and call it Q_6. Then write Q_6 as an SQL query with RA operations just as was done for Q_2 above.

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

(b) Compare queries Q_5 and Q_6 in a similar way as we did for Q_1 and Q_2.

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.

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

9. 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));
```

(a) Translate and optimize this query and call it Q_8. Then write Q_8 as an SQL query with RA operations just as was done for query Q_2 above.

(b) Compare queries Q_7 and Q_8 in a similar way as we did for Q_1 and Q_2.

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.

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

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

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 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)),
      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 r.a
from   NestedR r
where  r.Bs = '{}';
```

(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 Q_5 and Q_6.

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

Solution:

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 Q_{10}. (We assume that $S \neq \emptyset$.)

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)),
     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
```

(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 Q_7 and Q_8.

(b) You should also run additional experiments that only compare query Q_8 and Q_{10}.

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

4 Formulating Queries in the Object-Relational Model

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 object-relational databases and to use these to solve queries.

12. 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;
```

    Actually, we can also write this functions as follows:

```
create or replace function setunion(A anyarray, B anyarray) returns anyarray as
$$
select array( select unnest(A) union select unnest(B) order by 1);
$$ 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 `isIn` which checks if an object x is in a set S. (Again this function is defined polymorphically.)

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

13. Consider the view `student_books(sid,books)` which associates with each student the set of booknos of books he or she buys. Observe that there may be students who bought no books.

    (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 may be students who have no major.

Test that each of these views work properly. You will need to use them in the subsequent problems.

14. 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. You don't need these relations since that are but encapsulated inside the functions.

For example, a query such as

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

is **not** permitted. Rather, you should use the equivalent object-relational query

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

(a) Find the bookno and title of each book that cites at least three books that cost less than $50.

(b) Find the bookno and title of each book that was not bought by any students who majors in 'CS'.

(c) Find the sid of each student who bought all books that cost at least $50.

(d) Find the bookno of book that was not only bought by students who major in 'CS'.

(e) Find the bookno and title of each book that was not only bought by students who bought all books that cost more than $45.

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

(g) Find the pairs (b_1, b_2) of booknos of books that where bought by the same set of students.

(h) Find the pairs (b_1, b_2) of booknos of books that where bought by the same number of students.

(i) Find the sid of each student who bought all but four books.

(j) Find the sid of each student who bought no more books than the combined number of books bought by the set of students who major in Psychology.