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))
```

Solution:

We begin with the `IN` case. Due to the set-theoretic properties of `IN`, i.e., $\in$, we can rewrite the queries as follows.

```
SELECT L(r1)
FROM   R1 r1
WHERE  C1(r1) AND [NOT] (r1.A IN (SELECT DISTINCT s1.B1
                                  FROM   S1 s1
                                  WHERE  C2(s1,r1))
                         [OR|AND|AND NOT]
                         r1.A1 IN (SELECT DISTINCT t1.C1
                                  FROM   T1 t1
                                  WHERE  C3(t1,r1)));
```

We proceed with the `IN` cases, i.e., the cases

```
SELECT L(r)
FROM   R1 r1
WHERE  C1(r1) AND (r1.A1 IN (SELECT DISTINCT s1.B1
                             FROM   S1 s1
                             WHERE  C2(s1,r1))
                   [OR|AND|AND NOT]
                   r1.A IN (SELECT DISTINCT t1.C1
                             FROM   T1 t1
                             WHERE  C3(t1,r1)));
```

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

These get translated as follows:

```
SELECT L(r1)
FROM (SELECT r1.*
      FROM   R1 r1 JOIN S1 s1 ON (r1.A1 = s1.B1 AND C1(r1) AND C2(s1,r1))
      [UNION|INTERSECT|EXCEPT]
      SELECT r1.*
      FROM   R1 r1 JOIN T1 t1 ON (r1.A1 = t1.C1 AND C1(r1) AND C3(t1,r1))) q;
```

We can now express this SQL with RA operations in the standard RA notation:

$$\pi_{L(r_1)}(\pi_{r_1.*}(R_1 \bowtie_{A_1=B_1 \wedge C_1(r_1) \wedge C_2(r_1,s_1)} S_1)[\cup|\cap|-]\pi_{r_1.*}(R_1 \bowtie_{A_1=C_1 \wedge C_1(r_1) \wedge C_3(r_1,t_1)} T_1))$$

For the general case, the translations become

```
WITH R1 AS (SELECT r1.* FROM R1 r1)
SELECT L(r1,...,rk)
FROM (SELECT r1.*,...,rk.*
      FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
              JOIN (S1 s1 CROSS JOIN ... Sm sm) ON (r1.A = s1.B1 AND C2)
      [UNION|INTERSECT|EXCEPT]
      SELECT r1.*
      FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
             JOIN (T1 t1 CROSS JOIN ... CROSS JOIN Tn tn) ON (r1.A = t1.C1 AND C1 AND C3)) q;
```

In standard RA this becomes

$$\pi_{L(\mathbf{r})}(\pi_{\mathbf{r}.*}(\mathbf{R} \bowtie_{A_1=B_1 \wedge C_1(\mathbf{r}) \wedge C_2(\mathbf{s},\mathbf{r})} \mathbf{S})[\cup|\cap|-]\pi_{\mathbf{r}.*}(\mathbf{R} \bowtie_{A_1=C_1 \wedge C_1(\mathbf{r}) \wedge C_3(\mathbf{t},\mathbf{r})} \mathbf{T}))$$

where

$$\begin{aligned} \mathbf{r} &= r_1, \ldots r_k \\ \mathbf{s} &= s_1, \ldots r_m \\ \mathbf{t} &= r_1, \ldots r_n \\ \mathbf{r}.* &= r_1.*, \ldots r_k.* \\ \mathbf{R} &= R_1 \times \cdots \times R_k \\ \mathbf{S} &= S_1 \times \cdots \times S_m \\ \mathbf{T} &= T_1 \times \cdots \times T_n. \end{aligned}$$

This expression is also equivalent with

$$\pi_{L(\mathbf{r})}(\pi_{\mathbf{r}.*}(\sigma_{C_1(\mathbf{r})}(\mathbf{R}) \bowtie_{A_1=B_1 \wedge C_2(\mathbf{r},\mathbf{s})} \mathbf{S})[\cup|\cap|-]\pi_{\mathbf{r}.*}(\sigma_{C_1(\mathbf{r})}(\mathbf{R}) \bowtie_{A_1=C_1 \wedge C_3(\mathbf{t},\mathbf{r})} T_1)).$$

We next consider the `NOT IN`, i.e., $\notin$, cases. The translations becomes:

```
WITH R1 AS (SELECT r1.* FROM R1 r1)
SELECT L(r1)
FROM ((SELECT r1.*
       FROM   R1 r1
       WHERE  C1(r1)
```

```
 EXCEPT
 SELECT r1.*
 FROM   R1 r1 JOIN S1 s1 ON (C1(r1) AND r1.A1 = s1.B1 AND C2(s1,r1)))
[UNION|INTERSECT|EXCEPT]
(SELECT r1.*
 FROM   R1 r1
 WHERE  C1(r1)
 EXCEPT
 SELECT r1.*
 FROM   R1 r1 JOIN S1 s1 ON (C1(r1) AND r1.A1 = t1.C1 AND C3(t1,r1)))) q;
```

Translated in standard RA this becomes

$$\pi_{L(r_1)}(\quad \pi_{r_1.*}(\sigma_{C_1(r_1)}(R_1)) - \pi_{r_1.*}(R_1 \bowtie_{A_1=B_1 \wedge C_1(r_1) \wedge C_2(r_1,s_1)} S_1) \\ [\cup \mid \cap \mid -] \\ \pi_{r_1.*}(\sigma_{C_1(r_1)}(R_1)) - \pi_{r_1.*}(R_1 \bowtie_{A_1=C_1 \wedge C_1(r_1) \wedge C_3(r_1,t_1)} T_1) \quad)$$

For the general case, the translations become

```
WITH R1 AS (SELECT r1.* FROM R1 r1)
SELECT L(r1,...,rk)
FROM ((SELECT r1.*,...,rk.
       FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
       WHERE  c1(r1,...,rk)
       EXCEPT
       SELECT r1.*,...,rk.*
       FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
              JOIN (S1 s1 CROSS JOIN ... CROSS JOIN Sm sm) ON (r1.A1 = s1.B1 AND C1 AND C2))
      [UNION|INTERSECT|EXCEPT]
      ((SELECT r1.*,...,rk.
        FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
        WHERE  c1(r1,...,rk)
        EXCEPT
        SELECT r1.*
        FROM   (R1 r1 CROSS JOIN ... CROSS JOIN Rk rk)
               JOIN (T1 t1 CROSS JOIN ... CROSS JOIN Tn tn) ON (r1.A1 = t1.C1 AND C1 AND C3))) q;
```

Translated in RA this becomes

$$\pi_{L(\mathbf{r})}(\quad \pi_{\mathbf{r}.*}(\sigma_{C_1(\mathbf{r})}(\mathbf{R})) - \pi_{\mathbf{r}.*}(\mathbf{R} \bowtie_{A_1=B_1 \wedge C_1(\mathbf{r}) \wedge C_2(\mathbf{r},\mathbf{s})} \mathbf{S}) \\ [\cup \mid \cap \mid -] \\ \pi_{\mathbf{r}.*}(\sigma_{C_1(\mathbf{r})}(\mathbf{R})) - \pi_{\mathbf{r}.*}(\mathbf{R} \bowtie_{A_1=C_1 \wedge C_1(\mathbf{r}) \wedge C_3(\mathbf{r},\mathbf{t})} \mathbf{T}) \quad)$$

where

$$\begin{aligned} \mathbf{r} &= r_1, \ldots r_k \\ \mathbf{s} &= s_1, \ldots r_m \\ \mathbf{t} &= r_1, \ldots r_n \\ \mathbf{r}.* &= r_1.*, \ldots r_k.* \\ \mathbf{R} &= R_1 \times \cdots \times R_k \\ \mathbf{S} &= S_1 \times \cdots \times S_m \\ \mathbf{T} &= T_1 \times \cdots \times T_n. \end{aligned}$$

(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))
```

Solution Intuitively, these are semijoin (anti semijoin) situations.

In this case, we can create temporary views for the two subqueries. The translations for the `IN` cases become

```
WITH
     R1 AS (SELECT L(r1), r1.A1 FROM R1 r1 WHERE C1(r1)),
     S1 AS (SELECT s1.B1 AS A1 FROM S1 s1 WHERE  C2(s1)),
     T1 AS (SELECT t1.C1 AS A1 FROM T1 t1 WHERE  C3(t1))
SELECT L(r1)
FROM   R1 r1 NATURAL JOIN S1 s1
[UNION|INTERSECT|EXCEPT]
SELECT L(r1)
FROM   R1 r1 NATURAL JOIN T1 t1
```

We can now express this SQL with RA operations in the standard RA notation:

$$\pi_{L(r_1)}(\sigma_{C_1(r_1)}(R_1) \ltimes \sigma_{C_2(s_1)}(S_1))[\cup|\cap|-]\pi_{L(r_1)}(\sigma_{C_1(r_1)}(R_1) \ltimes \sigma_{C_3(t_1)}(T_1))$$

The translations for the `NOT IN` cases become

```
WITH
     R1 AS (SELECT L(r1), r1.A1 FROM R1 r1 WHERE C1(r1)),
     S1 AS (SELECT s1.B1 AS A1 FROM S1 s1 WHERE  C2(s1)),
     T1 AS (SELECT t1.C1 AS A1 FROM T1 t1 WHERE  C3(t1))
SELECT L(r1)
FROM
((SELECT r1.*
 FROM    R1 r1
 EXCEPT
```

```
SELECT r1.*
FROM R1 r1 NATURAL JOIN E1 e1)
[UNION|INTERSECT|EXCEPT]
(SELECT r1.*
 FROM   R1 r1
 EXCEPT
 SELECT r1.*
 FROM R1 r1 NATURAL JOIN T1 t1)) q;
```

Translated to RA this becomes

$$\pi_{L(r_1)}(\quad \pi_{r_1.*}(\sigma_{C_1(r_1)}(R_1)) - \pi_{r_1.*}(\sigma_{C_1}(R_1) \ltimes \sigma_{C_2(s_1)}(S_1)) \\ [\cup \mid \cap \mid -] \\ \pi_{r_1.*}(\sigma_{C_1(r_1)}(R_1)) - \pi_{r_1.*}(\sigma_{C_1}(R_1) \ltimes \sigma_{C_3(r_1,t_1)}(T_1)) \quad)$$

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

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

   Solution: Actually, we can give an immediate derivation of the equality:

$$\begin{aligned}
\pi_a(R \bowtie_{R.a=S.b \wedge R.b=S.a} S) &= \{x \mid \exists y \exists u \exists v : R(x, y, u) \wedge S(y, x, v)\} \\
&= \{x \mid \exists y : (\exists u R(x, y, u)) \wedge (\exists v S(y, x, v))\} \\
&= \{x \mid \exists y : (x, y) \in \pi_{a,b}(R) \wedge (x, y) \in \pi_{b,a}(S)\} \\
&= \{x \mid \exists y : (x, y) \in (\pi_{a,b}(R) \cap \pi_{b,a}(S))\} \\
&= \{x \mid x \in \pi_a(\pi_{a,b}(R) \cap \pi_{b,a}(S))\} \\
&= \pi_a(\pi_{a,b}(R) \cap \pi_{b,a}(S)).
\end{aligned}$$

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.
   Solution: The translation gives us

```
with  CSmajor as (select m.sid, m.major from major m where m.major = 'CS')
select distinct sid, sname
from   student
       natural join CSmajor
       natural join buys
       natural join cites c
       natural join book b1
       join book b2 on (c.citedbookno = b2.bookno and b1.price < b2.price)
order by 1,2;
```

   Consider the following expressions:

$$\begin{aligned} CSmajor &= \pi_{sid,major}(\sigma_{major=\text{`CS'}}(M)) \\ E &= S \bowtie CSmajor \bowtie Buys \bowtie Cites \bowtie B_1 \end{aligned}$$

   Then the RA expression becomes

$$\pi_{S.sid,Ssname}(E \bowtie_{citedbookno=B_2.bookno \wedge B_1.price<B_2.price} B_2).$$

(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.
Solution:

```
with  CS as (select sid from major where major = 'CS'),
      B as (select distinct bookno, price from book),
      E as (select distinct sid, sname, bookno
            from   student s
                   natural join CS
                   natural join buys),
      C as (select distinct c.bookno
            from   cites c
                   natural join B b1
                   join B b2 on (c.citedbookno = b2.bookno and b1.price < b2.price))
select distinct sid, sname
from    E natural join C;
```

Consider the following expressions:

$$\begin{aligned} CSmajor &= \pi_{sid}(\sigma_{major=\text{`CS'}}(M)) \\ B &= \pi_{bookno,price}(Book) \\ E &= (S \ltimes CSmajor) \bowtie Buys \\ C &= \pi_{bookno}((Cites \bowtie B_1) \bowtie_{citedbookno=B_2.bookno \wedge B_1.price<B_2.price} B_2) \end{aligned}$$

Then the optimized RA expression is

$$\pi_{sid,sname}(E \ltimes C).$$

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.
Solution:
The translation algorithm produces the SQL query

```
with
   BuysBookLessThan30 as (select t.sid, b.*
                          from   buys t
                                 natural join (select b.*
                                               from   book b
                                               where  price < 30) b),
   BuysBookLessThan60 as (select t.sid, b.*
                          from   buys t
                          natural join (select b.*
                                        from   book b
                                        where  price < 60) b),
   CSMajor as (select m.sid from major m where m.major = 'CS'),
   Major as (select sid, sname, major from student natural join major),
   MajorBuys60 as (select sid, sname, major, bookno
                   from   Major m natural join BuysBookLessThan60 t)
select distinct q.sid, q.sname, q.major
from   ((select m.*
         from   MajorBuys60 m
         except
         select m.*
         from   MajorBuys60 m natural join CSMajor cm)

        intersect

        (select m.*
         from   MajorBuys60 m
         except
         select m.*
         from   MajorBuys60 m join BuysBookLessThan30 t on (m.sid = t.sid))) q;
```

Consider the following expressions:

$$
\begin{aligned}
BuysBookLessThan60 &= Buys \bowtie \sigma_{price<60}(Book) \\
BuysBookLessThan30 &= Buys \bowtie \sigma_{price<30}(Book) \\
CSmajor &= \pi_{sid}(\sigma_{major=\text{`CS'}}(M)) \\
Major &= Student \bowtie M \\
MajorBuys60 &= Major \bowtie BuysBookLessThan60
\end{aligned}
$$

Then the RA expression for the query becomes

$$
\begin{aligned}
&\pi_{sid,sname,major}(\\
&\quad (MajorBuys60 - MajorBuys60 \ltimes CSMajor) \\
&\quad \cap \\
&\quad MajorBuys60 - (MajorBuys60 \bowtie_{MajorBuys60.sid=BuysBookLessThan30.sid} BuysBookLessThan30) \\
&)
\end{aligned}
$$

(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.

Solution:

```
with
   BuysBookLessThan30 as (select distinct sid
```

```
                        from   buys
                               natural join (select distinct bookno
                                             from   book
                                             where  price < 30) b),
   BuysBookLessThan60 as (select distinct sid, bookno
                        from   buys t
                        natural join (select bookno
                                      from   book
                                      where  price < 60) b),
   CSMajor as (select sid from major where major = 'CS'),
   Major as (select sid, sname, major from student natural join major),
   MajorBuys60 as (select distinct sid, sname, major, bookno
                   from   Major natural join BuysBookLessThan60)
select distinct sid, sname, major
from   (select sid, sname, major, bookno
        from   MajorBuys60
        except
        select sid, sname, major, bookno
        from   MajorBuys60
                natural join (select sid from CSMajor
                              union
                              select distinct sid from BuysBookLessThan30) q1) q;
```

Consider the following expressions:

$$\begin{aligned}
BuysBookLessThan60 &= \pi_{sid}(Buys \bowtie \sigma_{price<60}(Book)) \\
BuysBookLessThan30 &= \pi_{sid,bookno}(Buys \bowtie \sigma_{price<30}(Book)) \\
CSmajor &= \pi_{sid}(\sigma_{major=\text{`CS'}}(M)) \\
Major &= Student \bowtie M \\
MajorBuys60 &= Major \bowtie BuysBookLessThan60
\end{aligned}$$

Then the optimized RA expression for the query becomes

$$\pi_{sid,sname,major}(MajorBuys60 - MajorBuys60 \ltimes (CSMajor \cup BuysBookLessThan30))$$

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.
   Solution:

```
with   E as (select sid, sname, bookno, title, price from student
                                                        natural join buys
                                                        natural join book)

select distinct sid, sname, bookno
from   (select e.*
        from   E e
        except
        select e.*
        from   E e
               join (book b1 natural join buys t1) on (e.price < b1.price and e.sid = t1.sid)) q;
```

Consider the expression

$$E = \pi_{sid,sname,bookno,title,price}(Student \bowtie Buys \bowtie B).$$

Then the RA expression for the query becomes

$$\pi_{sid,sname,bookno}(E - \pi_{sid,sname,bookno,title,price}(E \bowtie_{E.price < B_1.price \wedge E.sid = T_1.sid} (B_1 \bowtie T_1)).$$

(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.

Solution:

```
with   E as (select distinct sid, sname, bookno, price from student
                                                          natural join buys
                                                          natural join book),
       T as (select distinct price, sid from (select distinct bookno, price
                                              from   book) q natural join buys)
select sid, sname, bookno
from   E e
except
select e.sid, sname, bookno
from   E e
       join T t on (e.price < t.price and e.sid = t.sid);
```

Consider the expressions

$$E = \pi_{sid,sname,bookno,price}(Student \bowtie Buys \bowtie B)$$

and

$$T = \pi_{price,sid}(\pi_{bookno,price}(Book) \ltimes Buys).$$

Then the RA expression for the query becomes

$$\pi_{sid,sname,bookno}(E) - \pi_{sid,sname,bookno}(E \bowtie_{E.price < T.price \wedge E.sid = T.sid} T).$$

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.

Solution:

```
with  MajorCSorMath as (select m.sid from major m
                        where  m.major = 'CS'
                        UNION
                        select m.sid from major m
                        where  m.major = 'Math')
select distinct q.bookno, q.title
from   (select b.*, s.*
        from   book b cross join student s natural join MajorCSorMath m
        except
        select b.*, s.*
        from   book b natural join buys t natural join student s
       ) q;
```

Consider the expression

$MajorCSorMath = \pi_{sid}(\sigma_{major=\text{`CS'}}(M)) \cup \pi_{sid}(\sigma_{major=\text{`Math'}}(M))$.

Then the RA expression for the query is

$\pi_{bookno,title}(B \times (S \ltimes MajorCSorMath) - \pi_{bookno,title,price,sid,sname}(B \bowtie T \bowtie S))$.

(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.

Solution:

```
with  MajorCSorMath as (select sid from major m
                        where  major = 'CS' or major = 'Math'),
      Bookbno as (select bookno from book),
      T as (select bookno
            from   (select  bookno, sid
```

```
                    from    Bookbno cross join MajorCSorMath
                    except
                    select  bookno, sid
                    from    buys) q)
select bookno, title
from   Book natural join T q order by 1,2;
```

Consider the expressions

$$\begin{aligned} MajorCSorMath &= \pi_{sid}(\sigma_{major=\text{`CS'} \vee major=\text{`Math'}}(M)) \\ Bookbno &= \pi_{bookno}(Book) \end{aligned}$$

Then the optimized RA expression is

$\pi_{bookno,title}(Book \ltimes \pi_{bookno}(\pi_{bookno,sid}(Bookno \times MajorCSorMath) - Buys))$.

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.

   Solution:

```
select distinct a
from   R natural join (select distinct a as b
                       from   R natural join (select distinct a as b
                                              from   R) q) q order by 1;
```

   In RA notation

$$\pi_a(R \bowtie_{R.b=a} \pi_a(R \bowtie_{R.b=a} \pi_a(R))).$$

   (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.

   Solution:

   Here are some experiments:

`R`	Q_3 (in ms)	Q_4 (in ms)
makerandomR(100,100,1000)	30	2
makerandomR(200,200,4000)	487	5
makerandomR(200,200,10000)	14253	22
makerandomR(500,500,50000)	248900	111

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

   Solution:

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

The PostgreSQL engine runs Q_3 using a three-level nested loop over `R` and is thus $O(|R|^3)$.

The PostgreSQL engine runs Q_4 using hash-joins and therefore run in linear time $O(|R|)$.

Clearly, optimization in this case speeds up the query by two orders of magnitude in $|R|$.

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.

   Solution:

```
select distinct a
from   Ra
except
select a
from   (select a, b
        from  R
        except
        select a, b
        from   R natural join S) q;
```

   (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.

   Solution:

`R`	`S`	Q_5 (in ms)	Q_6 (in ms)
makerandomR(1000,10,1000)	makerandomS(10,4)	1	6
makerandomR(10000,20,10000)	makerandomS(20,10)	8	31
makerandomR(10000,20,50000)	makerandomS(20,10)	23	102

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

Solution: For both queries, the performance is quite good. Optimization has not improved the performance. Looking at the PostgreSQL query plan, both queries are implemented with semijoin type operations and run in linear time $O(|R| + |S| + |Ra|)$.

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.

   Solution:

```
select distinct a
from   Ra
except
select a
from   (select a, b
        from   Ra cross join S
        except
        select a, b
        from   R) q;
```

   (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.

   Solution:

R	S	Q_7 (in ms)	Q_8 (in ms)
makerandomR(1000,10,1000)	makerandomS(10,4)	202	10
makerandomR(10000,20,10000)	makerandomS(20,10)	7264	209
makerandomR(10000,20,50000)	makerandomS(20,10)	39925	409

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

Solution: Clearly optimization has helped. Looking at the PostgreSQL query plan for Q_7 we observe a triple nested loop resulting in a time complexity of $O(|Ra| * |S| * |R|)$. On the other hand, the complexity of Q_8 is $O(|Ra| * |S| + |R|)$. This is an order of magnitude better in $|R|$

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

Solution: Given the time complexities, we expect that the ONLY query performs better than the ALL query. Indeed, for the optimized versions, the ONLY query runs in $O(|R| + |S| + |Ra|)$ whereas the ALL query runs in time $O(|S| * |Ra| + |R|)$.

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)
                  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;
```

    Actually, we can make this query 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  r.Bs <@ s.Bs
union
(select ra.a
 from   Ra ra
 except
 select r.a
 from   R r);
```

    Solution:

    We first associate, through nesting, with each $R.a$ value the set of $R.b$ values associated with this a value. We also collect the set of values in S in a set. The ONLY query requires subset containment of each $R.a$'s set of $R.b$ values with the set of S values and that check occurs in the

WHERE clause. We also consider the case when an Ra values does not occur in the a column of R. This is do using the addition of a UNION set operation.

(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.

Solution:

R	S	Q_5 (in ms)	Q_6 (in ms)	Q
makerandomR(1000,10,1000)	makerandomS(10,4)	1	6	
makerandomR(10000,20,10000)	makerandomS(20,10)	8	31	
makerandomR(10000,20,50000)	makerandomS(20,10)	23	102	

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

Solution:

We notice that the performance for Q_9 is in line with that for Q_6 and Q_7. This is understandable since the nesting and subset comparisons can still be done in linear time in R, S, and Ra.

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)
                 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
```

Solution:

We associate, through nesting, with each $R.a$ value the set of $R.b$ values associated with this a value. We also collect the set of values in S in a

set. The ALL query requires superset containment of each $R.a$'s set of $R.b$ values with the set of S values and that check occurs in the WHERE clause.

(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.

Solution:

R	S	Q_7 (in ms)	Q_8 (in ms)	Q_{10} (in ms)
makerandomR(1000,10,1000)	makerandomS(10,4)	202	6	10
makerandomR(10000,20,10000)	makerandomS(20,10)	7264	209	20
makerandomR(10000,20,50000)	makerandomS(20,10)	39925	409	58

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

Solution:

R	S	Q_8 (in ms)	Q_{10} (in ms)
makerandomR(100000,10, 50000)	makerandomS(10,3)	108	109
makerandomR(100000,10, 500000)	makerandomS(10,3)	769	846
makerandomR(100000,10, 1000000)	makerandomS(10,3)	1476	1527

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

Solution:

The ONLY query Q_8 has complexity $O(|Ra| + |S| + |R|$ whereas the ALL Q_{10} queries has complexity $O(|Ra| + |S| + |R|$. Since they both have the same complexity, we would expect similar performance. So in the complex-object relational databases, the ONLY and ALL queries are very similar. That is in sharp contrast with that in the relational setting.

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.
Solution:

```
create or replace function setunion(A anyarray, B anyarray) returns anyarray as
$$
select array( select unnest(A) intersect select unnest(B) order by 1);
$$ language sql;
```

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

Solution:

```
create or replace function setunion(A anyarray, B anyarray) returns anyarray as
$$
select array( select unnest(A) except select unnest(B) order by 1);
$$ language sql;
```

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.
Solution:

```
create or replace view student_books as
   select s.sid, array(select t.bookno
                       from   buys t
                       where  t.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.
Solution:

```
create or replace view book_students as
   select b.bookno, array(select t1.sid
                          from   buys t1
                          where  t1.bookno = b.bookno order by sid) as students
   from   book b order by bookno;
```

(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.
Solution:

```
create or replace view book_citedbooks as
   select b.bookno, array(select c1.citedbookno
                          from   cites c1
                          where  c1.bookno = b.bookno order by citedbookno) as citedbooks
   from   book b order by bookno;
```

(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.
Solution:

```
create or replace view book_citingbooks as
   select b.bookno as bookno, array(select c1.bookno
                                    from   cites c1
                                    where  c1.citedbookno = b.bookno order by bookno) as citingbooks
   from   book b order by bookno;
```

(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.)
Solution:

```
create or replace view major_students as
    select distinct m.major, array(select m1.sid
                                   from major m1
                                   where m1.major = m.major) as students
    from   major m order by major;
```

(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.
Solution:

```
create or replace view student_majors as
    select s.sid, array(select m.major from major m where m.sid = s.sid) as majors
    from   student s order by sid;
```

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.
Solution:

```
select  bookno, b.title
from    book b
where   b.bookno in (select bc.bookno
                     from   book_citedbooks bc
                     where (select count(1)
                            from   book
                            where  price < 50 and isIn(bookno,bc.citedbooks)) >= 3);
```

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

```
select  b.bookno, b.title
from    book b
where   b.bookno in (select bs.bookno
                     from   book_students bs
                            join major_students ms on  not(bs.students && ms.students)
                     where ms.major = 'CS');
```

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

Solution:

```
select  sb.sid
from    student_books sb
where   (select array(select bookno from book where price >= 50)) <@ sb.books;
```

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

Solution:

```
select  bookno
from    book_students bs
where   not(bs.students <@ (select students
                            from   major_students
                            where  major = '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.

Solution:

```
select  bookno
from    book_students bs
where   not(bs.students <@
                       (select array(select  sid
                        from   student_books sb
                        where  (select array(select bookno
                                             from book where price > 45))  <@ sb.books)));
```

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

Solution:

```
select  sb.sid, bc.bookno
from    student_books sb
        join book_citingbooks bc on (not(sb.books <@ bc.citingbooks));
```

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

Solution:

```
select bs1.bookno, bs2.bookno
from   book_students bs1
       join book_students bs2 on (bs1.students <@ bs2.students and
                                  bs2.students <@ bs1.students and
                                  bs1.bookno <> bs2.bookno);
```

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

Solution:

```
select bs1.bookno, bs2.bookno
from   book_students bs1
       join book_students bs2 on (cardinality(bs1.students) = cardinality(bs2.students) ) order by 1,2
```

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

Solution:

```
select sb.sid
from   student_books sb
where  cardinality(sb.books) = (select count(1) from book)-4;
```

(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.

Solution:

```
select sb.sid
from   student_books sb
where  cardinality(sb.books) <= (select count(1)
                                 from   (select UNNEST(sb.books)
                                         from   student_books sb
                                         where isIn(sb.sid, (select students
                                                             from   major_students
                                                            where   major = 'Psychology'))) q);
```