CS589 Principles of Database Systems
Fall 2008 Homework 4 Suggested Answers
100 points possible

1. Let r be a relation over relation schema R with schema(R) = XY and schema (S) = Y, where X and Y are both sets of attributes and all of the attributes of S (Y) are contained in the set of attributes of R.

   Then the relational algebra divide operator can be expressed as follows:

   $R \div S = \pi_X(R) - \pi_X((\pi_X(R) \times S) - R)$

   For the following schema, where s-id is the student id and c-id is the course id. The relation Core-course lists all of the required classes and the relation Completed represents that a particular student has completed a particular class.

   Completed(s-id, c-id)
   Core-course(c-id)

   a. (5 points) Write a relational algebra query WITHOUT using the divide operator that finds all student ids for students that have complete all of the core courses. That is, write a relational algebra query WITHOUT using the divide operator that is equivalent to Completed ÷ Core-course.

      $\pi_{s\text{-}id}(Completed) - \pi_{s\text{-}id}((\pi_{s\text{-}id}(Completed) \times Core\text{-}course) - Completed)$

   b. (5 points) Write a query in Datalog (with negation, without recursion) that is equivalent to Completed ÷ Core-course.

      Product(x,y) :- Completed(x,z),Core-course(y).
      Note: make sure that the variable z doesn't appear elsewhere in the body.
      Difference(x) :- Product(x,y), ¬Completed(x,y).
      Answer(x) :- Completed(x,y), ¬Difference(x).

      This follows the relational algebra. Note: there are other variations that work, as well.

   c. (5 points) Write a query in (tuple or domain) calculus that is equivalent to Completed ÷ Core-course.

      Note: this is quite easy to write because relational calculus languages have a universal quantifier.
      In domain relational calculus:
      $\{ x1{:}s\text{-}id \mid \forall x2{:}c\text{-}id\ (\exists c3{:}c\text{-}id\ (core\text{-}course(x2) \rightarrow Completed(x1,x3) \wedge x2=x3)) \}$
      Here's another query in domain relational calculus:
      $\{ x1{:}s\text{-}id \mid \forall x2{:}c\text{-}id\ (core\text{-}course(x2) \rightarrow Completed(x1,x2)) \}$
      This one is equivalent to:
      $\{ x1{:}s\text{-}id \mid \forall x2{:}c\text{-}id\ (\neg core\text{-}course(x2) \vee Completed(x1,x2)) \}$

      Note: the way that the divide query is written in the book on page 111 is not correct. The query should have used "Topics(x2) →" rather than "Topics(x2) ∧".

2. Using the following schema:

   Person(<u>id</u>, name, age, gender)
   Car(<u>vin</u>, make, model, year, color)
   Owns(<u>id, vin</u>)

   (39 points) Write example, well-formed formulas (that could appear in a domain calculus query) to demonstrate each of the following positive and negative occurrences of variables. Note, you don't need to write 13 different formulas. One of your formulas might allow you to provide an example of more than one of the positive or negative occurrences. Be sure to be clear about which variable, in which formula, you are talking about for each answer.

Positive occurrences of a variable x	**Negative occurrences of a variable x**
$R(y_1, y_2, \dots, x, \dots, y_k)$ Person(x, n, a, g)	
x=v where v is a constant x=6	x=y where y is a variable, y occurs negatively ¬Person(i,y,a,g) ∧ y=x
¬F if x occurs negatively in F ¬(¬Person(i,n,x,g))	¬F if x occurs positively in F ¬(Person(i,n,x,g))
$F_1 \wedge F_2$ if x occurs positively in F_1 or positively in F_2 Person(i, x, a, g) ∧ ¬Person(i2,x,a2,g2)	$F_1 \wedge F_2$ if x occurs negatively in F_1 and negatively in F_2 ¬Person(i, x, a, g) ∧ ¬Person(i2,x,a2,g2)
$F_1 \vee F_2$ if x occurs positively in F_1 and positively in F_2 Person(x, n, a, g) ∨ Owns(x,v)	$F_1 \vee F_2$ if x occurs negatively in F_1 or negatively in F_2 ¬Person(x, n, a, g) ∨ Owns(x,v)
$F_1 \rightarrow F_2$ if x occurs negatively in F_1 and positively in F_2 ¬Person(x,n,a,g) → Owns(x,v)	$F_1 \rightarrow F_2$ if x is positive in F_1 and negative in F_2 Person(x,n,a,g) → ¬Owns(x,v)
∃y:A (F) if x≠y and x occurs pos. in F ∃y:name Person(x,y,a,g)	∃y:A (F) if x≠y and x occurs neg. in F ∃y:name ¬Person(x,y,a,g)

3. Consider the following query language:

   Positive relational algebra consisting of: ∩, ∪, ⋈, σ, Π
   (Note: there is NO set difference in positive relational algebra.)

   Prove that positive relational algebra is contained in allowed (tuple or domain, your choice) relational calculus. In particular:

   a. (5 points) state and prove the base case

      First, observe that ∪, ⋈, σ, Π is a minimal set of operators for this language because

intersection can be expressed using the join operator (with appropriate renaming of attributes.)

Base case: Prove that a positive relational algebra query with zero operators can be expressed as an allowed domain calculus query.

Proof: The form of the relational algebra query with zero operators is $R(x_1, \ldots, x_n)$ for some relation R with n attributes. The equivalent relational calculus query is:
$\{ x_1, \ldots, x_n \mid R(x_1, \ldots, x_n) \}$
This query is allowed because all variables occur positively in the query expression.

b. (5 points) state the inductive hypothesis
Assume that all positive relational algebra queries with q or fewer operators can be expressed in allowed domain relational calculus.

c. (16 points) prove each case of the induction step
Prove that a positive relational algebra query with q+1 operators can be expressed in allowed domain relational calculus.

Case 1: the $q+1^{st}$ operator is $\cup$
The positive relational algebra query has the form $F_1 \cup F_2$ where F_1 and F_2 are each arbitrarily complex positive relational algebra expressions. Since this query has q+1 operators, the expression F_1 has fewer than q+1 operators and thus can be represented in an allowed domain relational calculus expression, based on the inductive hypothesis. Let E_1 be an equivalent allowed, domain relational calculus expression for F_1. Similarly, F_2 has fewer than q+1 operators and thus can be represented in an allowed domain relational calculus expression. Let E_2 be an equivalent allowed, domain relational calculus expression for F_2. Since we know that in the relational algebra expression, F_1 and F_2 must be union-compatible, we know E_1 and E_2 produce intermediate query answers with the same number of attributes, call it k. The equivalent domain relational calculus expression for $F_1 \cup F_2$ is
$\{ x_1, x_2, \ldots, x_k \mid E_1(x_1, x_2, \ldots, x_k) \vee E_2(x_1, x_2, \ldots, x_k) \}$
This query is allowed because E_1 and E_2 are allowed. Thus $x_1, \ldots, x_k$ occur positively in both E_1 and E_2 because they are free variables in E_1 and in E_2. Thus, $x_1, \ldots, x_k$ occur positively in $E_1(x_1, x_2, \ldots, x_k) \vee E_2(x_1, x_2, \ldots, x_k)$. Since all free variables in this query appear positively, the query is allowed. qed

Case 2: the $q+1^{st}$ operator is $\bowtie$
The positive relational algebra query with q+1 operators has the form: $F_1 \bowtie F_2$ where F_1 and F_2 are each arbitrarily complex positive relational algebra expressions. Since this query has q+1 operators, the expression F_1 has fewer than q+1 operators and thus can be represented in an allowed domain relational calculus expression, based on the inductive hypothesis. Let E_1 be an equivalent allowed, domain relational calculus expression for F_1. Similarly, F_2 has fewer than q+1 operators and thus can be represented in an allowed domain relational calculus expression. Let E_2 be an equivalent allowed, domain relational calculus expression for F_2. Assume that E_1 has n free variables, E2 has m free variables, and that E1 and E2 have k free variables in common, where $0 \le k \le$ minimum(m, n). Let x_i, i=1,n-k be the variables that appear in E_1 but not in E_2. Let z_i, i=1,k be the variables in common between E_1 and E_2. Let y_i, i=1,m-k be the variables that appear in E_2 but not in E_1. The equivalent domain relational calculus query is:
$\{x_1, \ldots, x_{n-k}, y_1, \ldots, y_{m-k}, z_1, \ldots, z_k \mid (E_1(x_1, \ldots, x_{n-k}, z_1, \ldots, z_k) \wedge E2(y_1, \ldots, y_{m-k}, z_1, \ldots, z_k) \}$
This query is allowed because E_1 and E_2 are allowed. Thus all free variables in this query occur positively. Thus, this query is allowed. Qed

Case 3: the q+1st operator is σ
The positive relational algebra query with q+1 operators has the form: $\sigma_{x=v}F$ where F is an arbitrarily complex positive relational algebra expression. (We can assume a simple select expression.) Since this query has q+1 operators, the expression F has q operators and thus can be represented in an allowed domain relational calculus expression, based on the inductive hypothesis. Let E be an equivalent allowed, domain relational calculus expression for F. Assume that F has n+1 free variables where the n+1st free variable is named x. The equivalent allowed domain calculus query is:
$\{ x_1, x_2, \ldots , x_k, x \mid E(x_1, x_2, \ldots , x_k, x) \wedge x=v\}$
This query is allowed because E is allowed and thus all free variables in E occur positively in E. And based on the remainder of the query, x occurs positively in the query. qed

Case 4: the q+1st operator is Π
The positive relational algebra query with q+1 operators has the form: $\Pi_{x1, \ldots, xk}F$ where F is an arbitrarily complex positive relational algebra expression. Since this query has q+1 operators, the expression F has fewer than q operators and thus can be represented in an allowed domain relational calculus expression, based on the inductive hypothesis. Let E be an equivalent allowed, domain relational calculus expression for F. Assume that F has n free variables where $n \geq k$. Assume that the first k variables in F are the projected variables. The equivalent allowed domain calculus query is:
$\{ x_1, x_2, \ldots , x_k \mid \exists x_{k+1} (\ldots(\exists x_n(E(x_1, x_2, \ldots , x_n))\ldots) \}$
This query is allowed because E is allowed and thus all free variables in E occur positively in E. Since all free variables occur positively in the query and since $x_{k+1},\ldots , x_n$ occur positively in the subformula E, this query is allowed. qed

4. (20 points) In the Ramakrishnan/Gehrke book, 3rd Edition (the 386/586 book), for problem 24.1, write queries 1, 3, 4, 7 in Datalog (with recursion and negation, as needed) and in SQL:1999 syntax.

1. Result(a) :- Flights(a, "Madison", c, d, e, f).
SELECT F.flno FROM Flights F WHERE F.from = "Madison";

3. Result(a) :- Flights(a, b, c, d, e, f), ¬(b= "Madison").
SELECT F.flno FROM Flights F WHERE F.from ≠ "Madison";

4. Reachable(b, c) :- Flights(a, b, c, d, e, f).
Reachable(x, c) :- Reachable(x, b), Flights(a, b, c, d, e, f).
Result(b) :- Reachable ("Madison", b).

The first two rules compute all pairs of reachable cities.
The third rule chooses from those, those cities reachable from Madison.

With recursive Reachable(from, to) AS
(SELECT from, to FROM Flights)
UNION
(SELECT R.from, F.to FROM Reachable R, Flights F WHERE R.to = F.From)

SELECT R.to FROM Reachable R WHERE R.from "Madison";

7. Reachable(b, c) :- Flights(a, b, c, d, e, f).
Reachable(x, c) :- Reachable(x, b), Flights(a, b, c, d, e, f).
Result (a) :- Flights(a, b, c, d, e, f), ¬Reachable("Madison", b), b ≠ "Madison".

```
with recursive Reachable (from, to) as
(SELECT F.from, F.to FROM Flights)
UNION
(SELECT R.from, F.to FROM Reachable R, Flights F WHERE R.to = F.from)

SELECT F.flno FROM Flights F
WHERE F.from != “Madison” AND NOT EXISTS
                                    (SELECT * FROM Reachable R WHERE
                                      R.to = F.from AND R.from  = “Madison”);
```