CHAPTER 27

Formal-Relational Query Languages

Solutions for the Practice Exercises of Chapter 27

Practice Exercises

27.1

Answer:

a. $\{t \mid \exists\, q \in r\,(q[A] = t[A])\}$

b. $\{t \mid t \in r \wedge t[B] = 17\}$

c. $\{t \mid \exists\, p \in r\, \exists\, q \in s\,(t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = p[C] \wedge t[D] = q[D] \wedge t[E] = q[E] \wedge t[F] = q[F])\}$

d. $\{t \mid \exists\, p \in r\, \exists\, q \in s\,(t[A] = p[A] \wedge t[F] = q[F] \wedge p[C] = q[D]\}$

27.2

Answer:

a. $\{ < t > \mid \exists\, p, q\,(< t, p, q > \in r_1)\}$

b. $\{ < a, b, c > \mid < a, b, c > \in r_1 \wedge b = 17\}$

c. $\{ < a, b, c > \mid < a, b, c > \in r_1 \vee < a, b, c > \in r_2\}$

d. $\{ < a, b, c > \mid < a, b, c > \in r_1 \wedge < a, b, c > \in r_2\}$

e. $\{ < a, b, c > \mid < a, b, c > \in r_1 \wedge < a, b, c > \notin r_2\}$

f. $\{ < a, b, c > \mid \exists\, p, q\,(< a, b, p > \in r_1 \wedge < q, b, c > \in r_2)\}$

27.3

Answer:

a. $\Pi_A(\sigma_{B=17}(r))$

b. $r \bowtie s$

c. $\Pi_A(s \bowtie (\Pi_{r.A}(\sigma_{r.b=d.b}(r \times \rho_d(r)))))$

27.4

Answer:

a. Query:

$$\{t \mid \exists m \in manages\,(t[person\_name] = m[person\_name] \wedge m[manager\_name] = \text{'Jones'})\}$$

b. Query:

$$\{t \mid \exists m \in manages\, \exists e \in employee(e[person\_name] = m[person\_name] \wedge m[manager\_name] = \text{'Jones'} \wedge t[city] = e[city])\}$$

c. Query:

$$\{t \mid \exists m1 \in manages\, \exists m2 \in manages(m1[manager\_name] = m2[person\_name] \wedge m1[person\_name] = \text{'Jones'} \wedge t[manager\_name] = m2[manager\_name])\}$$

d. Query:

$$\{t \mid \exists w1 \in works\, \neg\exists w2 \in works(w1[salary] < w2[salary] \exists e2 \in employee\,(w2[person\_name] = e2[person\_name] \wedge e2[city] = \text{'Mumbai'}))\}$$

27.5

Answer:

a. $query(X) :\!\text{-}\ r(X, 17)$

b. $query(X, Y, Z) :\!\text{-}\ r(X, Y), s(X, Z)$

c. $query(X) :\!\text{-}\ s(X, Y), r(X, Z), r(Y, W), Z > W$

27.6

Answer:

a. Query:

query (*X*) :- *p* (*X*)
p (*X*) :- *manages* (*X*, "Jones")
p (*X*) :- *manages* (*X*, *Y*), *p* (*Y*)

b. Query:

query(X, C) :- *p(X), employee(X, S, C)*
p(X) :- *manages(X, "Jones")*
p(X) :- *manages(X, Y), p(Y)*

c. Query:

query(X, Y) :- *p(X, W), p(Y, W)*
p(X, Y) :- *manages(X, Y)*
p(X, Y) :- *manages(X, Z), p(Z, Y)*

d. Query:

query(X, Y) :- *p(X, Y)*
p(X, Y) :- *manages(X, Z), manages(Y, Z)*
p(X, Y) :- *manages(X, V), manages(Y, W), p(V, W)*

27.7

Answer:

A Datalog rule has two parts, the *head* and the *body*. The body is a comma-separated list of *literals*. A *positive literal* has the form $p(t_1, t_2, \ldots, t_n)$ where p is the name of a relation with n attributes, and $t_1, t_2, \ldots, t_n$ are either constants or variables. A *negative literal* has the form $\neg p(t_1, t_2, \ldots, t_n)$ where p has n attributes. In the case of arithmetic literals, p will be an arithmetic operator like $>$, $=$ etc.

We consider only safe rules; see Section 27.4.4 for the definition of safety of Datalog rules. Further, we assume that every variable that occurs in an arithmetic literal also occurs in a positive nonarithmetic literal.

Consider first a rule without any negative literals. To express the rule as an extended relational-algebra view, we write it as a join of all the relations referred to in the (positive) nonarithmetic literals in the body, followed by a selection. The selection condition is a conjunction obtained as follows: If $p_1(X, Y)$, $p_2(Y, Z)$ occur in the body, where p_1 is of the schema (A, B) and p_2 is of the schema (C, D), then $p_1.B = p_2.C$ should belong to the conjunction. The arithmetic literals can then be added to the condition.

As an example, the Datalog query

query(X, Y) :- *works(X, C, S1), works(Y, C, S2), S1 > S2, manages(X, Y)*

becomes the following relational-algebra expression:

$$E_1 = \sigma_{(w1.company\_name = w2.company\_name \wedge w1.salary > w2.salary \wedge manages.person\_name = w1.person\_name \wedge manages.manager\_name = w2.person\_name)} (\rho_{w1}(works) \times \rho_{w2}(works) \times manages)$$

Now suppose the given rule has negative literals. First suppose that there are no constants in the negative literals; recall that all variables in a negative literal must also occur in a positive literal. Let $\neg q(X, Y)$ be the first negative literal, and let it be of the schema (E, F). Let E_i be the relational-algebra expression obtained after all positive and arithmetic literals have been handled. To handle this negative literal, we generate the expression

$$E_j = E_i \bowtie (\Pi_{A_1, A_2}(E_i) - q)$$

where A_1 and A_2 are the attribute names of two columns in E_i which correspond to X and Y respectively.

Now let us consider constants occurring in a negative literal. Consider a negative literal of the form $\neg q(a, b, Y)$ where a and b are constants. Then, in the above expression defining E_j, we replace q with $\sigma_{A_1 = a \wedge A_2 = b}(q)$.

Proceeding in a similar fashion, the remaining negative literals are processed, finally resulting in an expression E_w.

Finally the desired attributes are projected out of the expression. The attributes in E_w corresponding to the variables in the head of the rule become the projection attributes.

Thus our example rule finally becomes the view:

create view *query* **as**
$$\Pi_{w1.person\_name, w2.person\_name}(E_2)$$

If there are multiple rules for the same predicate, the relational-algebra expression defining the view is the union of the expressions corresponding to the individual rules.

The above conversion can be extended to handle rules that satisfy some weaker forms of the safety conditions, and to some restricted cases where the variables in arithmetic predicates do not appear in a positive nonarithmetic literal.