CHAPTER 6

Formal Relational Query Languages

Practice Exercises

6.1 Write the following queries in relational algebra, using the university schema.

a. Find the titles of courses in the Comp. Sci. department that have 3 credits.

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

c. Find the highest salary of any instructor.

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

e. Find the enrollment of each section that was offered in Autumn 2009.

f. Find the maximum enrollment, across all sections, in Autumn 2009.

g. Find the sections that had the maximum enrollment in Autumn 2009

Answer:

a. $\Pi_{title}(\sigma_{dept\_name = \text{'Comp. Sci'} \wedge credits=3}(course))$

b. $\Pi_{ID}(\sigma_{ID = \text{'Einstein'}}(takes \bowtie \rho_{t1(IID,\ course\_id,\ section\_id,\ semester,\ year)}teaches))$
Assuming the set version of the relational algebra is used, there is no need to explicitly remove duplicates. If the multiset version is used, the grouping operator can be used without any agggregation to remove duplicates. For example given relation $r(A, B)$ possibly containing duplicates, ${}_{A,B}\mathcal{G}(r)$ would return a duplicate free version of the relation.

c. $\mathcal{G}_{\mathbf{max}(salary)}(instructor)$

d. $instructor \bowtie (\mathcal{G}_{\textbf{max}(salary)\ \textbf{as}\ salary}(instructor))$
Note that the above query renames the maximum salary as salary, so the subsequent natural join outputs only instructors with that salary.

e. $_{course\_id, section\_id}\mathcal{G}_{\textbf{count}(*)\ \textbf{as}\ enrollment}(\sigma_{year=2009 \wedge semester=\text{Autumn}}(takes))$

f. $t1 \leftarrow {}_{course\_id, section\_id}\mathcal{G}_{\textbf{count}(*)\ \textbf{as}\ enrollment}(\sigma_{year=2009 \wedge semester=\text{Autumn}}(takes))$
result = $\mathcal{G}_{\textbf{max}(enrollment)}(t1)$

g. $t2 \leftarrow \mathcal{G}_{\textbf{max}(enrollment)\ \textbf{as}\ enrollment}(t1)$
where $t1$ is as defined in the previous part of the question.
result = $t1 \bowtie t2$

6.2 Consider the relational database of Figure 6.22, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

a. Find the names of all employees who live in the same city and on the same street as do their managers.

b. Find the names of all employees in this database who do not work for "First Bank Corporation".

c. Find the names of all employees who earn more than every employee of "Small Bank Corporation".

Answer:

a. $\Pi_{person\_name}((employee \bowtie manages) \bowtie_{(manager\_name = employee2.person\_name \wedge employee.street = employee2.street \wedge employee.city = employee2.city)} (\rho_{employee2}(employee)))$

b. The following solutions assume that all people work for exactly one company. If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, the problem is more complicated. We give solutions for this more realistic case later.

$\Pi_{person\_name}(\sigma_{company\_name \neq \text{"First Bank Corporation"}}(works))$

If people may not work for any company:

$\Pi_{person\_name}(employee) - \Pi_{person\_name}(\sigma_{(company\_name = \text{"First Bank Corporation"})}(works))$

c. $\Pi_{person\_name}(works) - (\Pi_{works.person\_name}(works \bowtie_{(works.salary \leq works2.salary \wedge works2.company\_name = \text{"Small Bank Corporation"})} \rho_{works2}(works)))$

6.3 The natural outer-join operations extend the natural-join operation so that tuples from the participating relations are not lost in the result of the join.

Describe how the theta-join operation can be extended so that tuples from the left, right, or both relations are not lost from the result of a theta join.

Answer:

a. The left outer theta join of $r(R)$ and $s(S)$ ($r \mathbin{⟕}_\theta s$) can be defined as
$(r \bowtie_\theta s) \cup ((r - \Pi_R(r \bowtie_\theta s)) \times (null, null, \ldots, null))$
The tuple of nulls is of size equal to the number of attributes in S.

b. The right outer theta join of $r(R)$ and $s(S)$ ($r \mathbin{⟖}_\theta s$) can be defined as
$(r \bowtie_\theta s) \cup ((null, null, \ldots, null) \times (s - \Pi_S(r \bowtie_\theta s)))$
The tuple of nulls is of size equal to the number of attributes in R.

c. The full outer theta join of $r(R)$ and $s(S)$ ($r \mathbin{⟗}_\theta s$) can be defined as
$(r \bowtie_\theta s) \cup ((null, null, \ldots, null) \times (s - \Pi_S(r \bowtie_\theta s))) \cup$
$((r - \Pi_R(r \bowtie_\theta s)) \times (null, null, \ldots, null))$
The first tuple of nulls is of size equal to the number of attributes in R, and the second one is of size equal to the number of attributes in S.

6.4 (**Division operation**): The division operator of relational algebra, "$\div$", is defined as follows. Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R. Then $r \div s$ is a relation on schema $R - S$ (that is, on the schema containing all attributes of schema R that are not in schema S). A tuple t is in $r \div s$ if and only if both of two conditions hold:

- t is in $\Pi_{R-S}(r)$
- For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
  a. $t_r[S] = t_s[S]$
  b. $t_r[R - S] = t$

Given the above definition:

a. Write a relational algebra expression using the division operator to find the IDs of all students who have taken all Comp. Sci. courses. (Hint: project *takes* to just ID and *course_id*, and generate the set of all Comp. Sci. *course_id*s using a select expression, before doing the division.)

b. Show how to write the above query in relational algebra, without using division. (By doing so, you would have shown how to define the division operation using the other relational algebra operations.)

Answer:

a. $\Pi_{ID}(\Pi_{ID, course\_id}(takes) \div \Pi_{course\_id}(\sigma_{dept\_name=\text{'Comp. Sci'}}(course))$

b. The required expression is as follows:
$r \leftarrow \Pi_{ID, course\_id}(takes)$

$$s \leftarrow \Pi_{course\_id}(\sigma_{dept\_name=\text{'Comp. Sci'}}(course))$$
$$\Pi_{ID}(takes) - \Pi_{ID}((\Pi_{ID}(takes) \times s) - r)$$

In general, let $r(R)$ and $s(S)$ be given, with $S \subseteq R$. Then we can express the division operation using basic relational algebra operations as follows:

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see that this expression is true, we observe that $\Pi_{R-S}(r)$ gives us all tuples t that satisfy the first condition of the definition of division. The expression on the right side of the set difference operator

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

serves to eliminate those tuples that fail to satisfy the second condition of the definition of division. Let us see how it does so. Consider $\Pi_{R-S}(r) \times s$. This relation is on schema R, and pairs every tuple in $\Pi_{R-S}(r)$ with every tuple in s. The expression $\Pi_{R-S,S}(r)$ merely reorders the attributes of r.

Thus, $(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ gives us those pairs of tuples from $\Pi_{R-S}(r)$ and s that do not appear in r. If a tuple t_j is in

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

then there is some tuple t_s in s that does not combine with tuple t_j to form a tuple in r. Thus, t_j holds a value for attributes $R - S$ that does not appear in $r \div s$. It is these values that we eliminate from $\Pi_{R-S}(r)$.

6.5 Let the following relation schemas be given:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Let relations $r(R)$ and $s(S)$ be given. Give an expression in the tuple relational calculus that is equivalent to each of the following:

a. $\Pi_A(r)$

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

c. $r \times s$

d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

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

6.6 Let $R = (A, B, C)$, and let r_1 and r_2 both be relations on schema R. Give an expression in the domain relational calculus that is equivalent to each of the following:

a. $\Pi_A(r_1)$

b. $\sigma_{B=17}(r_1)$

c. $r_1 \cup r_2$

d. $r_1 \cap r_2$

e. $r_1 - r_2$

f. $\Pi_{A,B}(r_1) \bowtie \Pi_{B,C}(r_2)$

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

6.7 Let $R = (A, B)$ and $S = (A, C)$, and let $r(R)$ and $s(S)$ be relations. Write expressions in relational algebra for each of the following queries:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 7)\}$

b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$

c. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

Answer:

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

b. $r \bowtie s$

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

6.8 Consider the relational database of Figure 6.22 where the primary keys are underlined. Give an expression in tuple relational calculus for each of the following queries:

a. Find all employees who work directly for "Jones."

b. Find all cities of residence of all employees who work directly for "Jones."

c. Find the name of the manager of the manager of "Jones."

d. Find those employees who earn more than all employees living in the city "Mumbai."

Answer:

a.

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

b.

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

c.

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

d.

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

6.9 Describe how to translate join expressions in SQL to relational algebra.
Answer: A query of the form

select $A1, A2, \ldots, An$
from $R1, R2, \ldots, Rm$
where P

can be translated into relational algebra as follows:
$\Pi_{A1, A2, \ldots, An}(\sigma_P(R1 \times R2 \times \ldots \times Rm))$
An SQL join expression of the form

$R1$ **natural join** $R2$

can be written as $R1 \bowtie R2$.
An SQL join expression of the form

$R1$ **join** $R2$ **on** (P)

can be written as $R1 \bowtie_P R2$.

An SQL join expression of the form

$$R1 \textbf{ join } R2 \textbf{ using } (A1, A2, \ldots, An)$$

can be written as $\Pi_S(R1 \bowtie_{R1.A1=R2.A1 \wedge R1.A2=R2.A2 \wedge \ldots R1.An=R2.An} R2)$
where S is $A1, A2, \ldots, An$ followed by all attributes of $R1$ other than $R1.A1, R1.A2, \ldots, R1.An$, followed by all attributes of $R2$ other than $R2.A1, R2.A2, \ldots, R2.An$,
The outer join versions of the SQL join expressions can be similarly written by using $\supset\!\bowtie$, $\bowtie\!\subset$ and $\supset\!\bowtie\!\subset$ in place of $\bowtie$.[1]
The most direct way to handle subqueries is to extend the relational algebra. To handle where clause subqueries, we need to allow selection predicates to contain nested relational algebra expressions, which can reference correlation attributes from outer level relations. Scalar subqueries can be similarly translated by allowing nested relational algebra expressions to appear in scalar expressions. An alternative approach to handling such subqueries used in some database systems, such as Microsoft SQL Server, introduces a new relational algebra operator called the Apply operator; see Chapter 30, page 1230-1231 for details. Without such extensions, translating subqueries into standard relational algebra can be rather complicated.

[1]The case of outer joins with the **using** clause is a little more complicated; with a right outer join it is possible that $R1.A1$ is null, but $R2.A1$ is not, and the output should contain the non-null value. The SQL **coalesce** function can be used, replacing S by **coalesce**($R1.A1$, $R2.A1$), **coalesce**($R1.A2$, $R2.A2$), ... **coalesce**($R1.An$, $R2.An$), followed by the other attributes of $R1$ and $R2$.