Database

You are given the following relational schema:

\[
\begin{align*}
\text{STUDENT(SNO, SNAME, DEPT)} \\
\text{ENROLL(CNO, SNO, GRADE)} \\
\text{COURSE(CNO, DEPT)} \\
\text{PREREQ(CNO, PNO)}
\end{align*}
\]

Keys are underlined. The column ENROLL(SNO) is a foreign key referencing STUDENT(SNO). All the occurrences of the columns CNO and PNO, except for the one in COURSE, are foreign keys referencing COURSE(CNO).

Course numbers are strings, like CSE562. The column COURSE(DEPT) contains just the departmental acronym, e.g., CSE.

You can assume that no columns are null. You may use views.

The example database is available in the following tables: zhouhany.student, zhouhany.enroll, zhouhany.course and zhouhany.prereq.

Problem 1 (12 pts)

Write the following queries in relational algebra and SQL:

- **1.1:** Find the names of all the students enrolled in CSE562:
  \[
  \pi_{\text{SNAME}}(\sigma_{\text{CNO}='CSE562'}(\text{STUDENT} \bowtie \text{ENROLL})).
  \]

SELECT S.SNAME FROM STUDENT S, ENROLL E
WHERE S.SNO=E.SNO AND E.CNO='CSE562'

- **1.2:** Find the names of all the students, whose grade in CSE531 is lower than the grade in CSE562, or who took at most one of CSE531 and CSE562:

  \[
  U1(S1) := \pi_{S1}(\sigma_{C1='CSE531'}(\text{ENROLL(C1,S1,G1)}) \bowtie_{S1=S2,G1<\text{G2}} (\sigma_{C2='CSE562'}(\text{ENROLL(C2,S2,G2))))); \\
  U2(N1) := \pi_{N1}(U1(S1) \bowtie \text{STUDENT(S1,N1,D1))}; \\
  U3(SNAME) := \pi_{\text{SNAME}}((\pi_{\text{SNO}}(\text{STUDENT}) - (\pi_{\text{SNO,CNO}}(\text{ENROLL}) \div \{\text{CSE531}',\text{CSE562}'\} \bowtie \text{STUDENT})) \bowtie \text{STUDENT}); \\
  U(S) := U2(N) \cup U3(N);
  \]

(SELECT S.SNAME FROM ENROLL E1, ENROLL E2, STUDENT S
WHERE E1.CNO='CSE531' AND E2.CNO='CSE562' AND E1.SNO=E2.SNO AND E1.GRADE<E2.GRADE AND S.SNO=E1.SNO)
UNION
(SELECT S.SNAME FROM STUDENT S
WHERE NOT EXISTS
    SELECT * FROM ENROLL E WHERE E.CNO='CSE531' AND E.SNO=S.SNO
    OR NOT EXISTS
    SELECT * FROM ENROLL E WHERE E.CNO='CSE562' AND E.SNO=S.SNO)
• 1.3: Find all the EE students who took all the courses offered by CSE.

\[
U_1(SNO) := \pi_{SNO,CNO}(\sigma_{DEPT='EE'}(STUDENT \bowtie ENROLL)) \div \pi_{CNO}(\sigma_{DEPT='CSE'}(COURSE));
\]

\[
U_2(SNAME) := \pi_{SNAME}(U_1(SNO) \bowtie STUDENT);
\]

\[
\text{SELECT S.SNAME FROM STUDENT S WHERE S.DEPT='EE' AND NOT EXISTS}
\text{ SELECT * FROM COURSE C WHERE C.DEPT='CS' AND NOT EXISTS}
\text{ SELECT * FROM ENROLL E WHERE E.CNO=C.CNO AND E.SNO=S.SNO}
\]

Problem 2 (8 pts)
Write the following queries in SQL:

• 2.1: For every course, list the course together with the average grade in that course;

\[
(\text{SELECT E.CNO, AVG(Grade) FROM ENROLL E GROUP BY E.CNO})
\text{UNION}
(\text{SELECT C.CNO, NULL FROM COURSE C WHERE NOT EXISTS}
\text{ SELECT * FROM ENROLL E WHERE E.CNO=C.CNO})
\]

• 2.2: For every course, list the course together with the department that had the maximum number of students in that course. If multiple departments are in that category, list all of them. If a course has zero enrollment, return null instead of the department name.

\[
\text{CREATE VIEW CDEPT(CNO,DEPT,NUM) AS}
\text{ SELECT E.CNO, S.DEPT, COUNT(*)}
\text{ FROM ENROLL E, STUDENT S}
\text{ WHERE E.SNO=S.SNO}
\text{ GROUP BY E.CNO, S.DEPT;}
\]

\[
(\text{SELECT CD.CNO, CD.DEPT FROM CDEPT CD}
\text{ WHERE NOT EXISTS}
\text{ SELECT * FROM CD CD1}
\text{ WHERE CD1.CNO=CD.CNO AND CD1.NUM>CD.NUM})
\text{UNION}
(\text{SELECT C.CNO, NULL}
\text{ FROM COURSE C WHERE NOT EXISTS}
\text{ SELECT * FROM ENROLL E WHERE E.CNO=C.CNO})
\]

Problem 3 (8 pts)
Write the following queries in SQL, possibly using recursion:
• 3.1: Find all the prerequisites, direct or indirect, of CSE562, assuming that the maximum length of prerequisite chains is 3.

• 3.2: Find all the prerequisites, direct or indirect, of CSE562, without assuming a maximum length of prerequisite chains.

WITH
RECURSIVE FULL(CNO,PNO) AS
(SELECT * FROM PREREQ)
UNION
(SELECT P.CNO, F.PNO
FROM PREREQ P, FULL F
WHERE P.PNO=F.CNO)
SELECT DISTINCT F.PNO FROM FULL F
WHERE F.CNO='CSE562'

• 3.3: Find all the departments that have a course requiring a CSE course as a direct or indirect prerequisite, without assuming a maximum length of prerequisite chains.

WITH
RECURSIVE FULL(CNO,PNO) AS
(SELECT * FROM PREREQ
UNION
SELECT P.CNO, F.PNO
FROM PREREQ P, FULL F
WHERE P.PNO=F.CNO)
SELECT DISTINCT C1.DEPT
FROM COURSE C1, FULL F, COURSE C2
WHERE C1.CNO=F.CNO AND F.PNO=C2.CNO AND C2.DEPT='CSE'

Note: Please bear in mind that Oracle does not support recursion. If you want to test your recursive queries, use PostgreSQL.

Problem 4 (12 pts)
Consider the following SQL queries:

• 4.1:

SELECT *
FROM STUDENT S
WHERE NOT EXISTS
SELECT * FROM ENROLL E
WHERE E.SNO = S.SNO
AND E.GRADE > 3.0

Answer: List all the students who never got a grade above 3.0.

\[(\pi_{SNO}(\text{STUDENT}) - \pi_{SNO}(\sigma_{\text{GRADE} \geq 3.0}(\text{ENROLL}))) \bowtie \text{STUDENT}\]

• 4.2:
SELECT E1.CNO, S.SNAME
FROM ENROLL E1, STUDENT S
WHERE E1.SNO = S.SNO
AND NOT EXISTS
SELECT * FROM ENROLL E2
WHERE E2.CNO = E1.CNO
AND E2.GRADE > E1.GRADE

Answer: For every course, return the names of the highest-scoring students.

\[
U_1(C_1, S_1) := \pi_{C_1, S_1}(ENROLL(C_1, S_1, G_1));
\]

\[
U_2(C_1, S_1) := \pi_{C_1, S_1}(ENROLL(C_1, S_1, G_1) \bowtie_{G_1 < G_2, C_1 = C_2} ENROLL(C_2, S_2, G_2));
\]

\[
U(C, N) := \pi_{C, N}((U_1(C, S) - U_2(C, S)) \bowtie STUDENT(S, N, D));
\]

For both queries:
- Explain what the query is doing.
- Translate the query to relational algebra.

**Problem 5 (10 pts)**

The relax-join operator is defined as follows:

- if the natural join of the relations R and S is nonempty, then return the result of this join;
- otherwise, return the Cartesian product of R and S.

Write queries in relational algebra and SQL that return the relax-join of two relations. Do not use IF-THEN-ELSE.

**Update (02/13/12).** It is enough to consider natural joins defined as:

\[
\pi_{A, B, D}(R(A, B) \bowtie_{B = C} S(C, D)).
\]

**Update (02/16/12).** Replace “natural join” by “equijoin.” Replace the above expression by

\[
R(A, B) \bowtie_{B = C} S(C, D).
\]

**Solution.**

\[
U_1(A, B, C, D) := \sigma_{B = C}(R(A, B) \times S(C, D));
\]

\[
U_2(A, B, C, D) := \pi_{A, B, C, D}(U_1(A_1, B_1, C_1, D_1) \times R(A, B) \times S(C, D));
\]

\[
U(A, B, C, D) := U_1(A, B, C, D) \cup (R(A, B) \times S(C, D) - U_2(A, B, C, D));
\]

The SQL translation is obtained by direct encoding of relational algebra operators.