1. (5 pts) **Relational algebra to SQL**

Given relational tables \( R(A, B), S(A, C), T(C, D) \), express the following relational algebra query in SQL.

\[
\Pi_{R.A,R.B,S.C}(\sigma_{R.A<10, T.D=11}(R \bowtie S \bowtie T))
\]

2. (5 pts) **Semijoin**

The *semijoin* of relations \( R \) and \( S \), denoted \( R \bowtie S \), is the set of tuples of \( R \) that agree with at least one tuple of \( S \) on all attributes that are common to the schemas of \( R \) and \( S \). Give an expression of the relational algebra that is equivalent to \( R \bowtie S \).

3. (30 pts) **Practical SQL exercises**

You will use PostgreSQL and a simplified version of a dataset from the UCI KDD archive to write and execute SQL queries. The dataset consists of three tables:

- **FILMS**(filmid, filmname, year, director, category)
- **ACTORS**(stagename, lname, fname, gender, dob, dod, country)
- **CASTS**(film, actor)

**FILMS**.filmid is a key for the **FILMS** table. **ACTORS**.stagename is a key for the **ACTORS** table. **CASTS**.film refers to **FILMS**.filmid and **CASTS**.actor refers to **FILMS**.stagename.

First load the database into PostgreSQL server, which is running in the edlab. Each table is in a separate text file (one tuple per line, with attributes separated by vertical bars). A starter script for loading the data, links to PostgreSQL documentation and other resources, are available on the course web page under “Assignments”.

Write SQL expressions for each of the following queries and execute them. Please turn in only the SQL queries (traces of the output are not necessary).

(a) List all actors in descending order of the number of films they acted in.

(b) Find the film(s) with the largest cast. Find the film(s) with the smallest cast. In both cases, also return the size of the cast.

(c) The Bacon number of an actor is the length of the shortest path between the actor and Kevin Bacon in the ”co-acting” graph. That is, Kevin Bacon has Bacon number 0; all actors who acted in the same film as KB have Bacon number 1; all actors who acted in the same film as some actor with Bacon number 1 have Bacon number 2, etc. Return all actors whose Bacon number is 2.
(d) Find the actors who acted in films of all categories in the database.
(e) Find all actors who acted only in films before 1960.
(f) Find the films with more female actors than male actors.

4. (15 pts) **Monotonicity**

A query or operator on relations is said to be *monotone* if whenever we add a tuple to one of its arguments, the result contains all the tuples that it contained before adding the tuple, plus perhaps more tuples. That is, there is no way to remove tuples from the output by adding tuples to the input.

(a) For each of the operators \{∪, ∩, −, σ, Π, ×\} state whether the operator is monotone.
(b) For each of the queries in the previous exercise, state whether or not they are monotone.
(c) Are all SELECT-FROM-WHERE queries in SQL monotone?

5. (5 pts + ec) **Join ordering**

Recall that joins are commutative and associative. That is, the following relational equivalences hold:

\[ R_1 \bowtie R_2 \equiv R_2 \bowtie R_1 \]
\[ R_1 \bowtie (R_2 \bowtie R_3) \equiv (R_1 \bowtie R_2) \bowtie R_3 \]

This allows us to freely reorder joins in an expression. Consider a query that performs the natural join of \( n \) relations:

\[ Q = R_1 \bowtie (R_2 \bowtie (R_3 \bowtie ... (R_{n-1} \bowtie R_n))) \]

(a) In a left-deep join expression, the right argument of each join operator is a base relation (rather than the result of a join). For example, \(((A \bowtie B) \bowtie C) \bowtie D\) is a left deep expression, but \((A \bowtie B) \bowtie (C \bowtie D)\) is not. (Imagine these expressions as operator trees and the terminology may seem more natural). How many different equivalent left-deep expressions are there for query \( Q \)?

(b) (Extra credit – 5 pts) With no restrictions on the form of expressions, how many different equivalent expressions are there for \( Q \)? Hint: Catalan numbers.

6. (10 pts) Consider the table **Sailors** (\( sid, sname, rating, age \)) and the query: *Find the name of the sailors with a higher rating than all sailors with age < 21.* The following two SQL queries attempt to obtain the answer to this question. Do they both compute the result? If not, explain why.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ( SELECT *
                   FROM Sailors S2
                   WHERE S2.age < 21 AND S.rating <= S2.rating )
```

```
SELECT *
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                        FROM Sailors S2
                        WHERE S2.age < 21 )
```
7. (15 pts) **Normal forms and decomposition** Consider the following collection of relations and dependencies. Assume that each relation is obtained through decomposition from a relation with attributes ABCDEFGHI and that all the known dependencies over relation ABCDEFGHI are listed for each question. (The questions are independent of each other, obviously, since the given dependencies over ABCDEFGHI are different.) For each (sub)relation:

i. State the strongest normal form that the relation is in.

ii. If it is not in BCNF, decompose it into a collection of BCNF relations.

(a) \( R_1(\text{A, C, B, D, E}) \), \( \text{A} \rightarrow \text{B}, \text{C} \rightarrow \text{D} \)

(b) \( R_2(\text{A, B, F}) \), \( \text{AC} \rightarrow \text{E}, \text{B} \rightarrow \text{F} \)

(c) \( R_3(\text{A, D}) \), \( \text{D} \rightarrow \text{G}, \text{G} \rightarrow \text{H} \)

(d) \( R_4(\text{D, C, H, G}) \), \( \text{A} \rightarrow \text{I}, \text{I} \rightarrow \text{A} \)

(e) \( R_5(\text{A, I, C, E}) \)

8. (10 pts) **Dependencies** Suppose that we have the following three tuples in a legal instance of a relation schema \( S \) with three attributes ABC (listed in order): (1, 2, 3), (4, 2, 3), and (5, 3, 3).

(a) Which of the following dependencies can you infer does not hold over schema \( S \)?

i. \( \text{A} \rightarrow \text{B} \),

ii. \( \text{BC} \rightarrow \text{A} \),

iii. \( \text{B} \rightarrow \text{C} \)

(b) Can you identify any dependencies that hold over \( S \)?