****

****



1. [FDs & Normalization] (25) Consider the following schema for relational data

Employer(Employer\_name, Project\_name, salary, department\_name,department\_manager)

and the following rules:

* + An employer can participate multiple projects;
  + An employer can get paid from each project he/she participates;
  + Each project is managed by only one department;
  + Each department has only one manager.
  1. Give the functional dependencies that express the above data constraints.

{Employer\_name, Project\_name} 🡪 salary

{Project\_name} 🡪 department\_name

{department\_name} 🡪 department\_manager

{Project\_name} 🡪 department\_manager

* 1. Give a primary key of the relation Employer.

The primary key is {Employer\_name, Project\_name}

* 1. Is Employer in the second normal form (2NF)? If not, decompose it into 2NF.

Due to the partial dependency of {Project\_name} 🡪 department\_name, the relation is not in 2NF

Employer\_Project(Employer\_name, Project\_name, salary)

Project(Project\_name, department\_name, department\_manager)

* 1. Is Employer in the third normal form (3NF)? If not, decompose it into 3NF.

There is a transitive dependency where department\_manager depends on department\_name, this means that the relationship is not in 3NF

Employer\_Project(Employer\_name, Project\_name, salary)

Project\_Department(Project\_name, department\_name,

Department(department\_name, department\_manager)

1. [FDs & Normalization] (25) Consider relation R(A,B,C,D,E,F). Given the following set of FDs: C→B, E→D, B→A, CE→B.
   1. There is only one candidate key for R. Determine what it is.

CEF is the only candidate key that is possible.

* 1. Which highest normal form is R in? 1NF, 2NF or 3NF? Explain your choice.

R must be in form 1NF. R cannot be in form 3NF if it is not in form 2NF, and R is not in form 2NF because B depends on C, which is part of the candidate key. The same thing happens to E and D, because D depends on E which is part of the candidate key.

1. **[RA operators]** (25) Given two tables: r with schema R, and s with schema S, where r contains **nr** tuples and s contains **ns** tuples, and **ns>nr>0**.
   1. Give the maximum and minimum possible number of tuples for the result produced by the following relational algebra expressions.

* r ∪ s
  1. Maximum = nr + ns (Contains all of the tuples from both sets because they are all unique)
  2. Minimum = ns (All tuples in r are also in s)
* s – r
  1. Maximum = ns (If no common tuples, all tuples in s are returned)
  2. Minimum = 0 (If all tuples in s are in r, no tuples are returned)
* σA>3 (r × s)
  1. Maximum = nr x ns (All of the tuples satisfy the condition)
  2. Minimum = 0 (None of the tuples satisfy the condition A > 3)
* πB(r)
  1. Maximum = nr (All tuples in r have different values for B)
  2. Minimum = 1 (All tuples in r have the same value for B)
* σA=2(πAB(s))
  1. Maximum = ns (All tuples in s satisfy the condition A=2)
  2. Minimum = 0 (No tuple satisfies the condition A=2)
  3. For each case, state the conditions/assumptions such that at least one tuple will be returned by the RA expression.
* As long as s contains at least one tuple
* One tuple in s that is not in r
* One pair of tuples in r x s that satisfies the condition A>3
* One tuple in r that satisfies the condition B
* One tuple in s with A = 2

1. **[RA expressions] (25)** Consider the following schema:

**Student(**Sid, Sname, Age, Major**) ,**

**Course(**Cno, Cname,Prerequisite, Ccredit**)**

**Score(**Cno,Sno,Grade**)**

1. (15) Express ths following queries in relational algebra (select σ, project ∏, Cartesian product X, join (theta-join), with logic expressions (e.g., ) if needed

-Q1: Find the student id, student name and department of all male students;

- Assuming there is a way to inquiry with gender, otherwise this is an invalid inquiry

- ∏Sid, Sname, Major(σGender = ‘M’(Student))

-Q2: Find the student id, student name, and course number of female students whose major is ‘CS’;

- Same idea as before, there has to be a way to inquiry by gender

- ∏Sid, Sname, Cno(σGender = ‘F’ ^ Major = ‘CS’(Student) join Score)

-Q3: List the names and student ids of those who have selected the course “Introduction to Data Science”

- ∏Sid, Sname(Student join (σCname = ‘Introduction to Data Science’(Course)) join Score)

-Q4: List the student ids and names of the students who didn’t chose the course with id ‘CSDS 101’

- ∏Sid, Sname(Student) - ∏Sid, Sname((σCno = ‘CSDS101’(Score)) theta join Student)

-Q5: List the student id of the students who are enrolled in both ‘CSDS 101’ and ‘CSDS 234’

- ∏Sid((σCno = ‘CSDS101’(Score)) union (σCno = ‘CSDS234’(Score)))

-Q6: List the id of the students, who have chosen the same courses enrolled by the student with id ‘005’

- ∏Sid((σSid = ‘005’(Score) thetajoin Score)

1. (10) Give a natural language description for the following relational algebra.

-Q7: σMajor = ’CS’(Students)

This query selects all students whose major is CS from students

-Q8: πSname (σAge>25(Students)) – πSname(σMajor=’CS’(Students))

This query lists all students who are older than 25 but are not majoring in CS

-Q9: πSname(σ Students.sid=borrows.sid(σmajor=’CS’(Students) × borrows))

This query gets all CS majors who have borrowed books assuming it is a conditional in a database, whether it is for course or for student.

-Q10: πsname(Students) − πS1.sname(σS1.age>S2.age(ρ(S1, Students)×ρ(S2, Students)))

This query gets all the names of students who are CS students and enrolled in courses and joins it with the students majoring in CS by student ID.

1. **Bonus question** [+10].

It is known that the following statements hold for FDs and any relations (called “axioms”):

* Reflexivity: if Y is a subset of X, then X→Y
* Augmentation: if X→Y, then XW→YW.
* Transitivity: if X→Y, Y→Z, then X→Z.
* Union: If X→Y and X→Z, then X→YZ.
* Decomposition: if X→YZ, then X→Y and X→Z.
* Pseudotransitivity: if X→Y, and WY→Z, then XW→Z.
* Set accumulation: if X→YZ, and Z→W, then X→YZW.

From the following set of four FDs, derive 5, 6, and 7 by applying the above axioms. Label each step with the axiom you use when applicable.

1. A -> B
2. C -> B
3. D -> ABC
4. AC -> D.
5. **D - > ABCD**

**We are given that D -> ABC. Also, D -> D (because of reflexivity). Thus, we can union and combine ABC and D to have D -> ABCD**

1. **AC -> BD**

**We are given that AC -> D. We also know that C -> B. Using Augmentation of A and axion 2, AC -> AB. Now, we can union AC -> D with AC -> AB to get AC -> ABD. Since A -> A is implied by reflexivity, the final can be AC -> BD by decomposition.**

1. **AC -> ABCD**

**We know that A -> B. We also know that AC -> ABD because of the previous question. We can apply augmentation AC -> BC. We can use union for AC -> ABD and AC -> BC to get AC -> ABCD.**