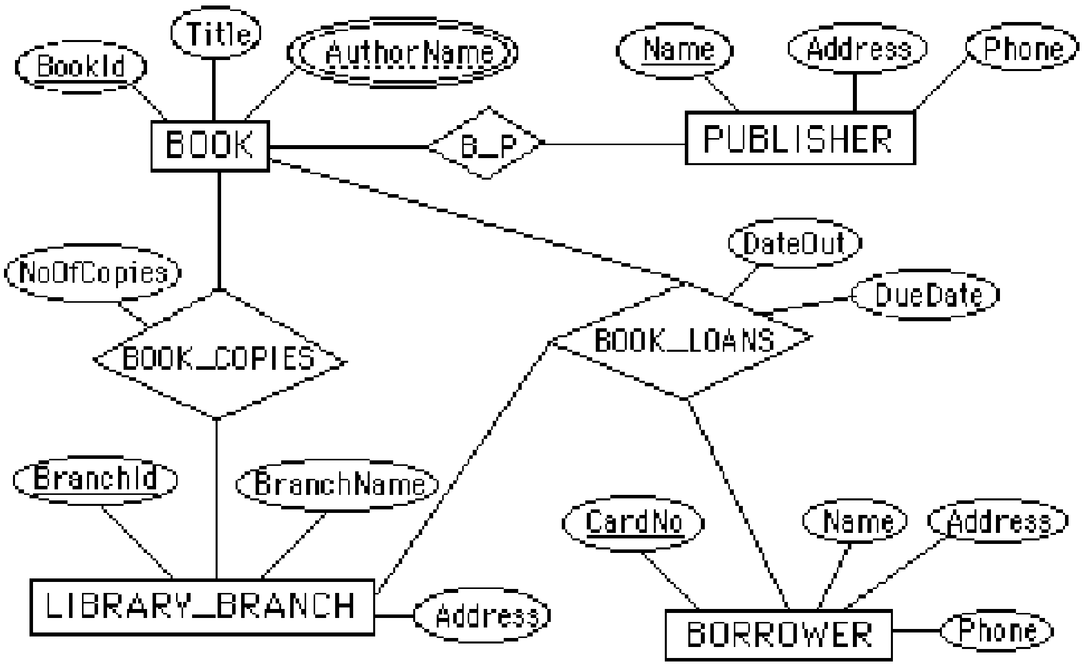
New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Homework #4 – Solutions

Course Title: Database Systems **Course Number:** CSCI-GA.2433-001
Instructor: Jean-Claude Franchitti

9.3 - Try to map the relational schema of Figure 6.14 into an ER schema. This is part of a process known as reverse engineering, where a conceptual schema is created for an existing implemented database. State any assumptions you make.

Answer:



Note: We represented BOOK_AUTHORS as a multi-valued attribute of BOOK in the above

ER diagram. Alternatively, it can be represented as a weak entity type.

9.5 Map the BANK ER schema of Exercise 7.23 (shown in Figure 7.21) into a relational schema. Specify all primary keys and foreign keys. Repeat for the AIRLINE schema (Figure 7.20) of Exercise 7.19 and for the other schemas for Exercises 7.16 through 7.24.

Partial Answer:

BANK
CODE NAME ADDR
ACCOUNT
ACCTNO BALANCE TYPE BCODE BNO
CUSTOMER
SSN NAME PHONE ADDR
LOAN
LOANNO AMOUNT TYPE BCODE BNO
BANK_BRANCH
BCODE BRANCHNO ADDR
A_C
SSN ACCTNO
L_C
SSN LOANNO
f.k.
f.k. f.k.
f.k.
f.k. f.k.
f.k.
f.k. f.k.

10.23 - A possible DDL corresponding to Figure 3.1 is shown below:

```
CREATE TABLE STUDENT (
    NAME            VARCHAR(30) NOT NULL,
    SSN                     CHAR(9)             PRIMARY KEY,
    HOMEPHONE       VARCHAR(14),
    ADDRESS         VARCHAR(40),
    OFFICEPHONE     VARCHAR(14),
    AGE                     INT,
    GPA                     DECIMAL(4,3)
);
```

Discuss the following detailed design decisions:

a. The choice of requiring NAME to be NON NULL.
b. Selection of SSN as the PRIMARY KEY.
c. Choice of field sizes and precision.
d. Any modification of the fields defined in this database.
e. Any constraints on individual fields.

Answer:
a) This means that no Students can exist without a Name.

b) This has a possible problem because there are some restrictions on using SSN as an identifier for people, hence why most schools assign a student ID. It also does not allow (easily) for a change to a student's SSN if this key is used elsewhere in the DB.

c) Giving Name a size of 30 does not allow for long names (think foreign students), especially since it contains a student's entire name (first, middle, last). Homephone and Officephone, with size 14, might not allow for overseas or foreign formatted phone numbers. Age is expected to be (roughly) between 1 and 100, so using an INT is much more space than needed. A TinyInt would be better, since it allows 0 to 255. The GPA field does not allow for GPA systems that go above 9 (such as the 10 point scale and the 12 point scale), though these are rare.

d) I would suggest breaking the Name field into three fields: FirstName, MiddleName, and LastName. It is more complicated and less specific to search for a student by name if the name data is in one field. See question 3.19.

The same reasoning can be applied to the Homephone and Officephone fields, breaking them into multiple fields such as AreaCode, PhoneNumber, Extension, etc.

Another possible modification is to increase the size of the Homephone, Officephone, and SSN fields so that formatting characters can be entered, such as SSN becoming a VARCHAR(11) to allow for the two dashes in XXX-YY-ZZZZ.

The table design allows for two phone numbers, Homephone and Officephone, either of which may be NULL. This means that some rows will have empty fields for those attributes. A solution is then to add a Phone# (SSN, Type, Number) table and remove the two phone number fields from the Students table. This allows for any student to have zero or more phone numbers of various types, which we can specify (Home, Office, Cell, etc.).

Consider not storing the AGE data and instead storing DateOfBirth. Typically only the year component of Age is stored which is less accurate than if DateOfBirth were stored and the age computed from it. That is, we can generate Age from DateOfBirth, but we cannot generate DateOfBirth from Age.

e) A suggested constraint is on the GPA field, to restrict its values from 0.000 to 4.000. This would, however, prevent the (rarely seen) 10 or 12 point grade scale. Another possible constraint is to limit Age to be from 0 to 100 (or even a lower maximum value).

6.16 - Specify the following queries on the COMPANY relational database schema shown in Figure 3.5, using the relational operators discussed in this chapter. Also show the result of each query as it would apply to the database state of Figure 3.6.

(a) Retrieve the names of employees in department 5 who work more than 10 hours per week on the 'ProductX' project.

(b) List the names of employees who have a dependent with the same first name as themselves.

(c) Find the names of employees that are directly supervised by 'Franklin Wong'.

(d) For each project, list the project name and the total hours per week (by all employees) spent on that project.

(e) Retrieve the names of employees who work on every project.

(f) Retrieve the names of employees who do not work on any project.

(g) For each department, retrieve the department name, and the average salary of employees working in that department.

(h) Retrieve the average salary of all female employees.
(i) Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.
(j) List the last names of department managers who have no dependents.

Answers:
In the relational algebra, as in other languages, it is possible to specify the same query in multiple ways. We give one possible solution for each query. We use the symbol s for SELECT, P for PROJECT, J for EQUIJOIN, * for NATURAL JOIN, and f for FUNCTION.

(a) EMP_W_X <-- (s PNAME='ProductX' (PROJECT)) J (PNUMBER),(PNO) (WORKS_ON) EMP_WORK_10 <-- (EMPLOYEE) J (SSN),(ESSN) (s HOURS>10 (EMP_W_X)) RESULT <-- P LNAME,FNAME (s DNO=5 (EMP_WORK_10))

Result:
LNAME FNAME
Smith John
English Joyce

(b) E <-- (EMPLOYEE) J (SSN,FNAME),(ESSN,DEPENDENT_NAME) (DEPENDENT) R <-- P LNAME,FNAME (E)

Result (empty):
LNAME FNAME

(c) WONG_SSN <-- P SSN (s FNAME='Franklin' AND LNAME='Wong' (EMPLOYEE)) WONG_EMPS <-- (EMPLOYEE) J (SUPERSSN),(SSN) (WONG_SSN) RESULT <-- P LNAME,FNAME (WONG_EMPS)

Result:
LNAME FNAME
Smith John
Narayan Ramesh
English Joyce

(d) PROJ_HOURS(PNO,TOT_HRS) <-- PNO f SUM HOURS (WORKS_ON) RESULT <-- P PNAME,TOT_HRS ((PROJ_HOURS) J (PNO),(PNUMBER) (PROJECT))

Result:
PNAME TOT_HRS
ProductX 52.5
ProductY 37.5
ProductZ 50.0
Computerization 55.0

Reorganization 25.0
Newbenefits 55.0

(e) PROJ_EMPS(PNO,SSN) <-- P PNO,ESSN (WORKS_ON) ALL_PROJS(PNO) <-- P PNUMBER (PROJECT) EMPS_ALL_PROJS <-- PROJ_EMPS -:- ALLPROJS (* DIVISION operation *) RESULT <-- P LNAME,FNAME (EMPLOYEE * EMP_ALL_PROJS)

Result (empty):
LNAME FNAME

(f) ALL_EMPS <-- P SSN (EMPLOYEE) WORKING_EMPS(SSN) <-- P ESSN (WORKS_ON) NON_WORKING_EMPS <-- ALL_EMPS - WORKING_EMPS (* DIFFERENCE*) RESULT <-- P LNAME,FNAME (EMPLOYEE * NON_WORKING_EMPS)

Result (empty):
LNAME FNAME

(g) DEPT_AVG_SALS(DNUMBER,AVG_SAL) <-- DNO f AVG SALARY (EMPLOYEE) RESULT <-- P DNUMBER,AVG_SAL (DEPT_AVG_SALS * DEPARTMENT)

Result:
DNUMBER AVG_SAL
Research 33250
Administration 31000
Headquarters 55000

(h) RESULT(AVG_F_SAL) <-- f AVG SALARY (s SEX='F' (EMPLOYEE))

Result:
AVG_F_SAL
31000

(i) E_P_HOU(SSN) <-- P ESSN (WORKS_ON J(PNO),(PNUMBER) (s PLOCATION='Houston' (PROJECT))) D_NO_HOU <-- P DNUMBER (DEPARTMENT) - P DNUMBER (s DLOCATION='Houston' (DEPARTMENT)) E_D_NO_HOU <-- P SSN (EMPLOYEE J(PNO),(DNUMBER) (D_NO_HOU)) RESULT_EMPS <-- E_P_HOU - E_D_NO_HOU (* this is set DIFFERENCE *) RESULT <-- P LNAME,FNAME,ADDRESS (EMPLOYEE * RESULT_EMPS)

Result:
LNAME FNAME ADDRESS
Wallace Jennifer 291 Berry, Bellaire, TX

(j) DEPT_MANAGERS(SSN)<-- P MGRSSN (DEPARTMENT) EMPS_WITH_DEPENDENTS(SSN) <-- P ESSN (DEPENDENT) RESULT_EMPS <-- DEPT_MANAGERS - EMPS_WITH_DEPENDENTS RESULT <-- P LNAME,FNAME (EMPLOYEE * RESULT_EMPS)

Result:
LNAME FNAME
Borg James

6.24 - Specify queries (a), (b), (c), (e), (f), (i), and (j) of Exercise 6.16 in both tuple and domain relational calculus.

Answer:
(a) Retrieve the names of employees in department 5 who work more than 10 hours per week on the 'ProductX' project.

Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND e.DNO=5 AND (EXISTS p) (EXISTS w) (WORKS_ON(w) AND PROJECT(p) AND e.SSN=w.ESSN AND w.PNO=p.PNUMBER AND p.PNAME='ProductX' AND w.HOURS>10) }

Domain relational Calculus:
{ qs | EMPLOYEE(qrstuvwxyz) AND z=5 AND (EXISTS a) (EXISTS b) (EXISTS e) (EXISTS f) (EXISTS g) (WORKS_ON(efg) AND PROJECT(abcd) AND t=e AND f=b AND a='ProductX' AND g>10) }

(b) List the names of employees who have a dependent with the same first name as themselves.

Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (EXISTS d) (DEPENDENT(d) AND e.SSN=d.ESSN AND e.FNAME=d.DEPENDENT_NAME) }

Domain relational Calculus:
{ qs | (EXISTS t) (EXISTS a) (EXISTS b) (EMPLOYEE(qrstuvwxyz) AND DEPENDENT(abcde) AND a=t AND b=q) }

(c) Find the names of employees that are directly supervised by 'Franklin Wong'.

Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (EXISTS s) (EMPLOYEE(s) AND s.FNAME='Franklin' AND s.LNAME='Wong' AND e.SUPERSSN=s.SSN) }

Domain relational Calculus:
{ qs | (EXISTS y) (EXISTS a) (EXISTS c) (EXISTS d) (EMPLOYEE(qrstuvwxyz) AND EMPLOYEE(abcdefghij) AND a='Franklin' AND c='Wong' AND y=d) }

(e) Retrieve the names of employees who work on every project.

Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (FORALL p) (NOT(PROJECT(p)) OR (EXISTS w) (WORKS_ON(w) AND p.PNUMBER=w.PNO AND w.ESSN=e.SSN)) }

Domain relational Calculus:
{ qs | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND (FORALL b) (NOT(PROJECT(abcd)) OR (EXISTS e) (EXISTS f) (WORKS_ON(efg) AND e=t AND f=b)) }

(f) Retrieve the names of employees who do not work on any project.

Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND NOT(EXISTS w) (WORKS_ON(w) AND w.ESSN=e.SSN) }

Domain relational Calculus:
{ qs | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND NOT(EXISTS a) (WORKS_ON(abc) AND a=t)) }

(i) Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.

Tuple relational Calculus:
{ e.LNAME, e.FNAME, e.ADDRESS | EMPLOYEE(e) AND (EXISTS p) (EXISTS w) (WORKS_ON(w) AND PROJECT(p) AND e.SSN=w.ESSN AND w.PNO=p.PNUMBER AND p.PLOCATION='Houston' AND NOT(EXISTS l) (DEPT_LOCATIONS(l) AND e.DNO=l.DNUMBER AND l.DLOCATION='Houston')) }

Domain relational Calculus:
{ qsv | (EXISTS t) (EXISTS z) (EMPLOYEE(qrstuvwxyz) AND (EXISTS b) (EXISTS c) (EXISTS e) (EXISTS f) (WORKS_ON(efg) AND PROJECT(abcd) AND t=e AND f=b AND c='Houston' AND NOT(EXISTS h) NOT(EXISTS i) (DEPT_LOCATIONS(hi) AND z=h AND i='Houston')) }

(j) List the last names of department managers who have no dependents.

Tuple relational Calculus:
{ e.LNAME | EMPLOYEE(e) AND (EXISTS d) (DEPARTMENT(d) AND e.SSN=d.MGRSSN AND NOT(EXISTS x) (DEPENDENT(x) AND e.SSN=x.ESSN)) }

Domain relational Calculus:
{ s | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND (EXISTS c) (DEPARTMENT(abcd) AND t=c AND NOT(EXISTS e) (DEPENDENT(efghi) AND e=t)) }

6.32 - A nested query is query within a query. More specifically, a nested query is a parenthesized query that can be used as a value in a number of places, such as instead of a relation or a selection condition. Specify the following queries on the database specified in Figure 3.5 using the concept of nested queries and the relational operators discussed in this chapter. Also show the result of each query as it would apply to the database state of Figure 3.6.

a. List the names of all employees who work in the department that has the employee with the highest salary among all employees.

b. List the names of all employees whose supervisor's supervisor has '888665555' for SSN.

c. List the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

Answers:

a) RESULT <-- P LNAME (S DNO = (P DNO (S SALARY = MAX(SALARY) EMPLOYEE)) EMPLOYEE)

b) RESULT <-- P LNAME (S SUPERSSN = (P SSN (S SUPERSSN = ‘88866555’ EMPLOYEE)) EMPLOYEE)

c) RESULT <-- P LNAME (S SALARY >= 10000 + P SALARY (S SALARY = MIN(SALARY) EMPLOYEE) EMPLOYEE)