Stanford University Computer Science Department
1999 Comprehensive Exam in Databases
WITH SOLUTIONS

- The exam is *open book and notes.*
- Answer all 5 questions on the exam paper itself, in the space provided for each question.
- The total number of points is 60; questions may have differing point values.
- You have 60 minutes to complete the exam (i.e., one minute per point). It is suggested you review the entire exam first, in order to plan your strategy.
- *Simplicity and clarity of solutions will count.* You may get as few as 0 points for a problem if your solution is far more complicated than necessary, or if we cannot understand your solution.

Provide your magic number here: _____

1	2	3	4
5			

Problem 1: (10 points) Suppose we have a relation $R(a, b, c, d)$ and the functional dependencies $a \to b$ and $b \to c$.

a) What other nontrivial FD's *with singleton right sides, but not necessarily singleton left sides* follow logically from these?

Answer: $ac \to b$, $ad \to b$, $acd \to b$, $a \to c$, $ab \to c$, $ad \to c$, $bd \to c$, $abd \to c$.

b) In the space below, show a possible instance of R that satisfies the given FD's, all FD's that follow logically from them, *but no other nontrivial dependencies.*

Answer:

a	b	c	d
0	1	2	3
4	5	2	3
6	1	2	3
0	1	2	7

c) Give an example of a nontrivial multivalued dependency that follows logically from the given dependencies and involves the attribute d.

Answer: $a \twoheadrightarrow d$.

Problem 2: (10 points) Consider the following ODL description:

```
interface Person (extent People) {
    attribute string name;
    relationship Set<Person> Friends inverse Friends;
}
```

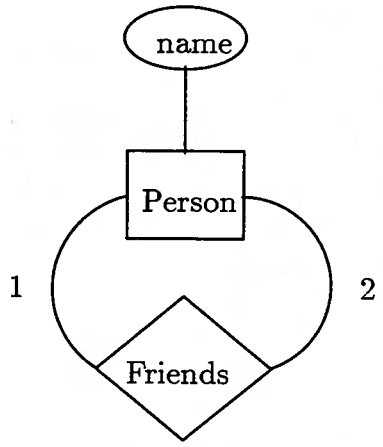
a) Write an OQL query on this database to find the people who are friends with someone who is a friend of the person named "Joe."

Answer:

```
SELECT p.name
FROM People p, p.Friends q, q.Friends r
WHERE r.name = "Joe";
```

b) In the space below, give an entity/relationship diagram for the same database.

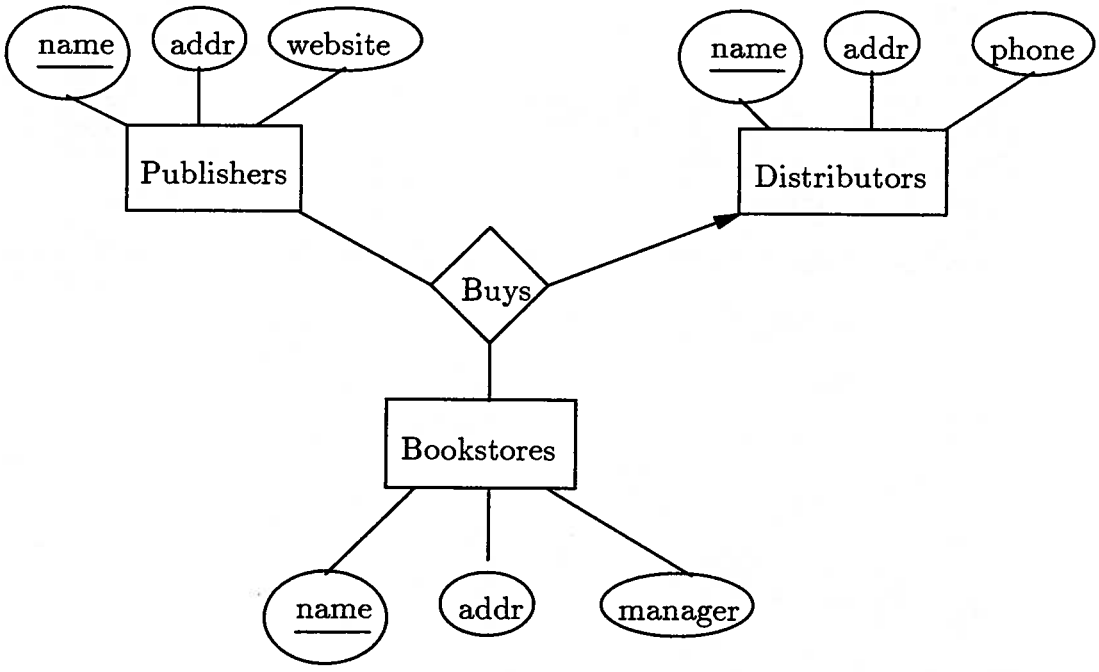
Answer:



Problem 3: (15 points) *Distributors* buy books from *Publishers* and sell them to *Bookstores*. A publisher may sell books to several different distributors, and bookstores may buy from several distributors. However, one bookstore will only buy the books of one publisher from a single distributor. For each publisher, distributor, and bookstore, we wish to record their name and address. We also record the manager of a bookstore, the web-site of a publisher, and the phone of a distributor. Names of publishers may be assumed unique, likewise names of distributors and names of bookstores, although names may be shared by entities of two different types (e.g., a publisher and bookstore could have the same name).

a) In the space below, draw an entity-relationship diagram that represents the information above.

Answer:



b) Give an ODL design for the same information. Indicate keys as appropriate, but you do not have to indicate extents. You may use convenient abbreviations rel = relation; attr = attribute; int = interface; inv = inverse. You may use simple types, e.g., strings, for all your attributes. Please be sparing of space; there is not enough room to use lines for things like single braces.

Answer:

```
interface Publisher (key name) {
    attribute string name;
    attribute string addr;
    attribute string webSite;
    relationship Set<Sale> sells inverse Sale::thePub;
}

interface Bookstore (key name) {
    attribute string name;
    attribute string addr;
    attribute string manager;
    relationship Set<Sale> buys inverse Sale::theStore;
}

interface Distributor (key name) {
    attribute string name;
    attribute string addr;
    attribute string phone;
    relationship Set<Sale> moves inverse Sale::theDist;
}

interface Sale {
    relationship Publisher thePub inverse Publisher::sells;
    relationship Bookstore theStore inverse Bookstore::buys;
    relationship Distributor theDist inverse Distributor::moves;
}
```

Problem 4: (5 points) The following code is in the style of recursive SQL3. If $A(x, y)$ is an arc relation, and we think of $P(x)$ as the set of nodes of a graph that are "on" in a given round, then the recursion defines a sort of "game of life," where a node is on at a given round if and only if it has both on and off successors at the previous round.

```
WITH
    RECURSIVE P(x) AS
        SELECT A1.x
        FROM A A1, A A2, P
        WHERE A1.x = A2.x AND A1.y = P.x AND NOT EXISTS
            (SELECT * FROM P WHERE x = A2.y)
SELECT * FROM P;
```

However, there are several ways in which this code violates the specifications for SQL3 recursion. Give the *two* most important such problems:

Answer: The negation is unstratified, and the recursion is nonlinear.

Problem 5: (20 points) Consider the following relational database table:

`DanceRoster(name, gender, age)`

You may assume that *name* is a key for the table. Some sample data for this table is:

name	*gender*	*age*
Tina	female	21
Brian	male	24
Mary	female	26
Susan	female	28
Edward	male	32
Fred	male	32

a) Write a query that for each dancer finds "good" dance partners. For a dancer D, a partner P is consider "good" if:

   (i) P has a different gender from D, and

   (ii) There is no other dancer P who has a different gender and is closer to D in age. Note that a dancer may have multiple good partners if there are ties in age differences.

   For the sample data above, the query should return:

dancer	*partner*
Tina	Brian
Brian	Mary
Mary	Brian
Susan	Brian
Susan	Edward
Susan	Fred
Edward	Susan
Fred	Susan

You should write this query in standard SQL2, but you may assume the existence of a function ABS that returns the absolute value of its argument. However, for this part, **you may not use any aggregation functions.**

Answer:

```
SELECT R1.name AS dancer, R2.name AS partner
FROM DanceRoster R1, DanceRoster R2
WHERE R2.gender <> R1.gender
    AND ABS(R1.age - R2.age) <= ALL
        (SELECT ABS(R1.age - R3.age)
        FROM DanceRoster R3
        WHERE R3.gender <> R1.gender)
```

b) Repeat part (a), but this time, write the query using the aggregation function MIN in a nontrivial way.

Answer:

```
SELECT R1.name AS dancer, R2.name AS partner
FROM DanceRoster R1, DanceRoster R2
WHERE R2.gender <> R1.gender
    AND ABS(R1.age - R2.age) =
        (SELECT MIN(ABS(R1.age - R3.age))
        FROM DanceRoster R3
        WHERE R3.gender <> R1.gender)
```

c) Write a relational-algebra expression that returns the names of the oldest female and the oldest male dancers. You may assume a relation-renaming operation in the algebra if you need it. Note that more than two dancers' names could be returned if there are ties in ages. For our sample data the query should return:

name
Susan
Edward
Fred

Answer:

DR2(name2,gender2,age2) := DanceRoster;

R1 := DR2 $\bowtie_{gender2=gender\ AND\ age2>age}$ DanceRoster;

R2 := π_{name}(R1);

ANS := π_{name}(DanceRoster) − R2;

d) Using SQL2, SQL3, or something similar, write a constraint asserting that DanceRoster contains the same number of males as females.

Answer:

```
CREATE ASSERTION Balanced CHECK(
    (SELECT COUNT(*) FROM DanceRoster WHERE gender = 'male') =
    (SELECT COUNT(*) FROM DanceRoster WHERE gender = 'female'))
```

e) Suppose that the database supports two modes of constraint-checking: `FOR EACH ROW` (constraint is checked after each tuple-level database modification that could potentially violate the constraint), and `FOR EACH STATEMENT` (constraint is checked at the end of every SQL database modification statement that could potentially violate the constraint). State which option should be selected for your constraint in part (d) and very briefly explain why.

Answer: We must use "for each statement," because otherwise, any insertion of data would cause a violation as soon as the first tuple was inserted, even if the statement maintained the male/female balance after all the insertions were performed. Likewise, a deletion of several tuples would cause a violation upon deletion of the first tuple, even if balance was restored at the end.