Stanford University Computer Science Department
2001 Comprehensive Exam in Databases

- The exam is *open book and notes*.
- There are 7 problems on the exam, with a varying number of points for each problem and subproblem for a total of 60 points (i.e., one point per minute). It is suggested that you look through the entire exam before getting started, in order to plan your strategy.
- Please write your solutions in the spaces provided on the exam. Make sure your solutions are neat and clearly marked.
- *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.

MAGIC NUMBER: _____

Problem	1	2	3	4	5	6	7	TOTAL
Max. points	8	7	7	10	12	4	12	60
Points								

The following is a statement of the Stanford University Honor Code:

A. The Honor Code is an undertaking of the students, individually and collectively:
1. *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
2. *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

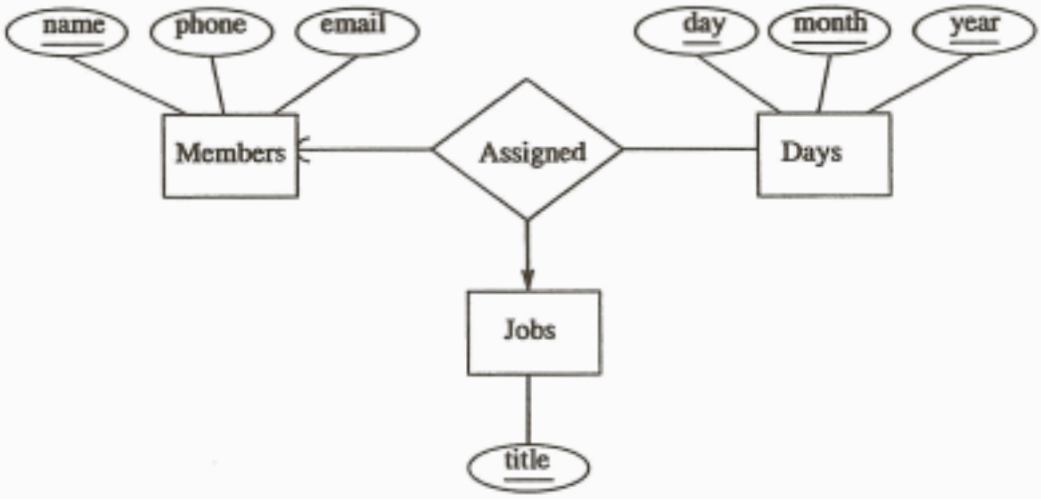
B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

Stanford University Computer Science Department
2001 Comprehensive Exam in Databases
SAMPLE SOLUTIONS

1. **Entity-Relationship Model** (8 points)

An office "coffee club" assigns three tasks each day; the tasks are: (1) Clean the pot. (2) Brew the coffee. (3) Buy the supplies. Each day, each of the jobs is assigned to one of the members, and no member can be assigned to two jobs on the same day. We wish to design a database in which we shall store the assignments of jobs to members for each day, and also store some information about members: their name, phone number and email address. You may assume there are more than three members in the club. The database should enforce the constraints on assignments of jobs as best it can. Give an appropriate E/R diagram for this database, and explain your reasoning.



2. **Functional Dependencies** (7 points)

Suppose we are given a relation $R(A, B, C, D)$ with functional dependencies $ABC \rightarrow D$ and $D \rightarrow AB$.

(a) What are the keys for R?
$\{A, B, C\}$ *and* $\{C, D\}$

(b) Is R in third normal form? Explain your reasoning.
Yes; all attributes are prime, so there cannot be a violation.

(c) Is R in Boyce-Codd normal form? Explain your reasoning.
No; D is not a superkey, so $D \rightarrow AB$ is a BCNF violation.

3. **Multivalued Dependencies** (7 points)

Consider a relation $R(A, B, C)$, and suppose that in each of the columns A, B, and C, no value can appear more than once (although the same value could appear in two or three different columns).

(a) List all of the nontrivial multivalued dependencies satisfied by R.

$A \twoheadrightarrow B, A \twoheadrightarrow C, B \twoheadrightarrow A, B \twoheadrightarrow C, C \twoheadrightarrow A, C \twoheadrightarrow B$

(b) Suppose the constraint on R is relaxed so that column A can have a value appearing more than once, but B and C still do not have duplicate values in their columns. Now, list all of the nontrivial MVD's satisfied by R.

$B \twoheadrightarrow A, B \twoheadrightarrow C, C \twoheadrightarrow A, C \twoheadrightarrow B$

4. **Relational Algebra** (10 points)

Suppose we are given a relation *Sells*(*bar, beer, price*), whose tuples (a, b, c) mean that bar a sells beer b at price c. Write in the "classical" relational algebra (operations union, intersection, difference, selection, projection, product, renaming, and the natural and theta-joins) the following queries. You may use complete expressions, expression trees, or sequences of assignments to local variables as you wish.

(a) Find the bars that sell two different beers at the same price.

```
Sells1(bar,beer1,price) := Sells
Temp1(bar,beer,beer1,price) := Sells NATURAL JOIN Sells1
Temp2(bar,beer,beer1,price) := SIGMA_{beer!=beer1}(Temp1)
Answer(bar) := PI_bar(Temp2)
```

(b) Find the greatest price at which any beer is sold at any bar.

```
Temp1(price) := PI_price(Sells)
Temp2(price1,price2) := Temp1 * Temp1
Temp3(price1,price2) := SIGMA_{price1<price2}(Temp2)
Temp4(price) := PI_price1(Temp3)
Answer(price) := Temp1 - Temp4
```

5. **SQL** (12 points)

A school chess team maintains rankings of its players based on scores, using the following relation:

```
Player(name, score) // name is a key
```

(a) For the first part of the problem assume that scores are unique within the relation. Write a single SQL query over the `Player` relation that takes a player name as an argument (denoted `:n` in the query) and returns the player's ranking by score. For example, if the `<name, score>` tuples in the `Player` relation are:

```
<Tim, 168>
<Kelly, 130>
<Shelby, 129>
<Miles, 110>
<Emma, 92>
<Rachel, 75>
```

then for `:n = Tim` the query result should be 1, for `:n = Shelby` the query result should be 3, for `:n = Rachel` the query result should be 6, etc. Write the query here:

```
SELECT COUNT(*)
FROM   Player
WHERE  score >=
  (SELECT score FROM Player P where P.name = :n)
```

(b) Now suppose that scores are not unique within relation `Player`. In this case, the players are in groups that are ranked according to score. You are to write a single SQL query that takes a player name as an argument (denoted `:n` in the query) and returns the rank of the player's group. For example, if the `<name, score>` tuples in the `Player` relation are:

```
<Tim, 168>
<Kelly, 130>
<Shelby, 130>
<Miles, 130>
<Emma, 92>
<Rachel, 75>
<Richard, 75>
```

then for `:n = Tim` the query result should be 1; for `:n = Kelly`, `:n = Shelby`, or `:n = Miles`, the query result should be 2; for `:n = Emma` the query result should be 3; for `:n = Rachel` or `:n = Richard` the query result should be 4. *Remember that simplicity of solutions does count.* Write the query here:

```
SELECT COUNT(DISTINCT score)
FROM   Player
WHERE  score >=
  (SELECT score FROM Player P where P.name = :n)
```


30

6. **Constraints** (4 points)

What familiar constraint type is encoded by the following SQL general assertion? *Your answer should contain at most two words.*

```
CREATE ASSERTION Mystery AS
   (NOT EXISTS (SELECT * FROM R
                WHERE R.A NOT IN (SELECT B FROM S)))
```

Answer: *referential integrity*

7. **ODL and OQL** (12 points)

Consider the following ODL (Object Definition Language) schema.

```
interface Applicant (extent Apps, key SSN) {
  attribute integer SSN;
  attribute Struct<string first, string last> name;
  relationship Set<Job> applied
     inverse Job::applicants; }

interface Job (extent Jobs, key (company, city) {
  attribute string company;
  attribute string city;
  relationship Set<Applicant> applicants
     inverse Applicant::applied }
```

(a) Is the relationship between applicants and jobs enforced by this schema one-one, one-many, many-one, or many-many?

many-many

(b) What is the simplest modification we can make to the schema to enforce that each applicant can apply for only one job?

Change `relationship Set<Job> applied`
to `relationship Job applied`.

(c) Using OQL (Object Query Language), write a query to find the SSN and last name of all applicants who have applied for a job in Palo Alto. Do not repeat (*SSN,last-name*) pairs in the result, even if the applicant has applied for many jobs in Palo Alto.

```
SELECT DISTINCT Struct(a.SSN, a.name.last)
FROM   Apps a, a.applied j
WHERE  j.city = "Palo Alto"
```