Final exam practice questions

CSCC43 – Introduction to Databases

These are actual questions from last year's midterm and final exam. They should give a good indication of the sorts of questions I will ask this year.

Solutions are inside boxes to distinguish them from the questions

NOTE: There are not very many questions about functional dependencies or normal forms because this year I completely changed how I teach those, and the old questions would be useless to you (they asked busywork questions about situations that only arise if you fail to start from a proper minimal basis). The practice FD problems posted online this semester are both numerous and very close to what you can expect to see on the final.

1. Intersection vs. join vs. Cartesian product

Consider relations R and S shown to the right. Compute the cardinality and arity of each of the following:

a) R x S
b) R ∩ S
c) R ⋈ $\rho_{C=X}$(S)
d) R ⋈ $\rho_{B=X,C=Y}$(S)

R

A	B	C
1	0	1
1	0	0
1	1	1
1	2	0
0	2	2
0	0	1
2	0	0
0	2	0

S

A	B	C
2	0	0
2	0	1
0	2	1
0	2	0
2	2	0
0	2	0
2	2	0
2	1	2

R x S has six columns and |R|*|S| = 8*8 = 64 rows.

R ∩ S has 3 columns and only the two rows that are present in both R and S (200 and 020).

R ⋈ $\rho_{C=X}$(S) has four columns and only combines groups of rows if they have the same A and B: {022, 020} x {021, 020, 020} + {200} x {200, 201} = 8 rows

R ⋈ $\rho_{B=X,C=Y}$(S) has five columns and combines groups of rows if they have the same A: {022,001,020} x {021,020,020} + {200}x{200,201,220,220,212} = 14 rows

2. Joins and Aggregation

Consider the following relations:

F·amousPerson(personID, name, …)
R·eporter(reporterID, name, …)
C·aughtOnCamera(personId, reporterId, timestamp, …)

Give SQL queries that give the following information. Timestamps can be converted to Julian days with the function '`julianday()`' and the function '`now()`' returns the current timestamp. For brevity you may use F/R/C as table names and pid/rid for person and reporter ids.

a) Name each famous person who was photographed at least five times in the last 90 days. Ignore people photographed by only one reporter. Your query should also compute the number of qualifying photos taken and how many reporters were involved.

```
select fp.name, count(distinct rid) nreps, count(*) nphoto
from fp join c using (pid)
where julianday(c.day) + 90 >= julianday(now() )
group by fp.pid, fp.name
having nreps > 2 and nphoto > 5;
```

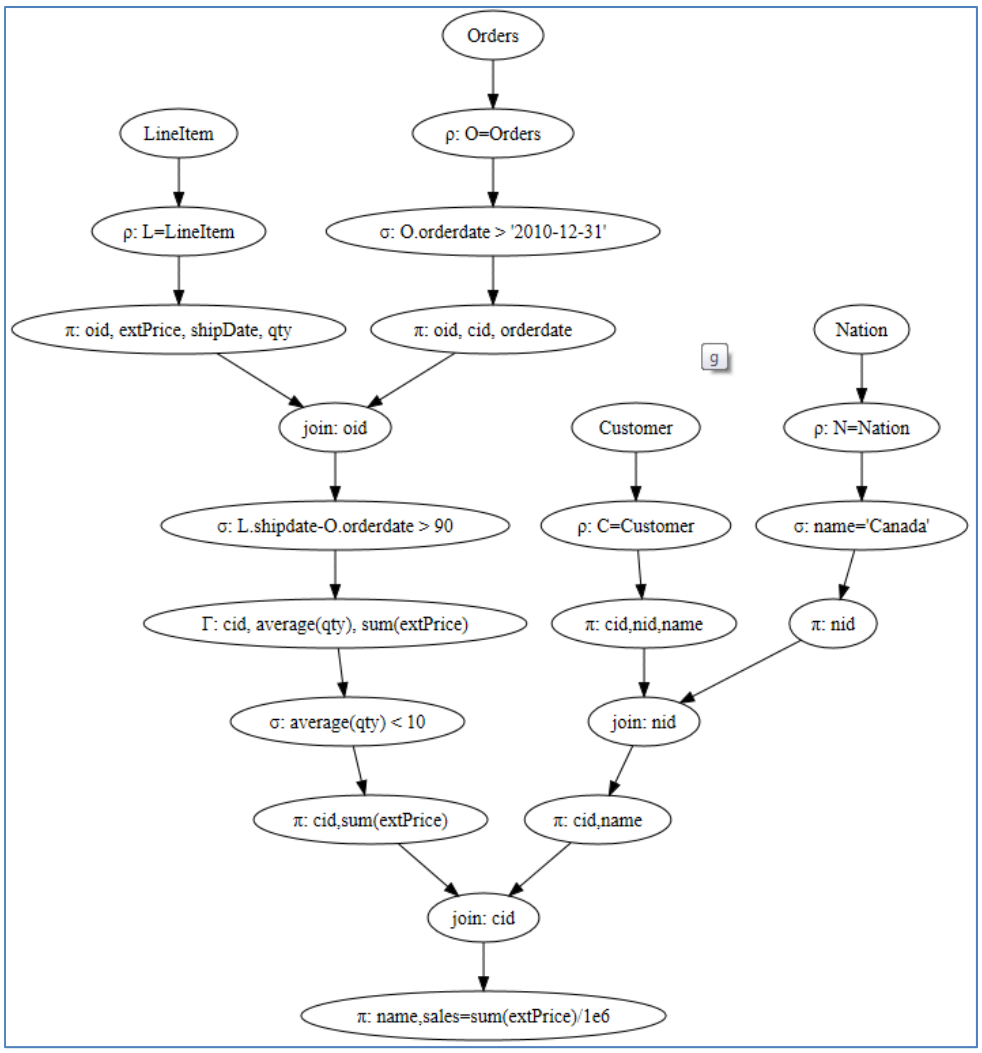
b) Name each reporter and the last person (if any) they photographed on 1 Aug 2011.

```
select r.name,fp.name
from r left join (        -- get the most recent photo time for each reporter
                select rid,max(timestamp) timestamp from c
                where julianday(timestamp)=julianday('2011-08-01')
                group by rid
        ) using (rid)                   -- allow reporters having no photos
        join c using (rid,timestamp)    -- pull in pid, if present
        left join fp using (pid);       -- pull in person's name, if present
```

3. Relational algebra expression trees

a) Convert the following SQL query into a relational algebra expression tree that places σ and π operators so as to minimize the amount of data the system must process.

```
select C.name, sum(L.extPrice)/1e6 sales
from LineItem L, Orders O, Customer C, Nation N
where L.oid=O.oid and O.cid=C.cid and C.nid=N.nid
   and N.name = 'Canada' and O.orderdate > '2010-12-31'
   and julianday(L.shipdate) - julianday(O.orderdate) > 90
group by C.name having average(L.qty) < 10
order by sales desc
```



4. Entity-relationship data model

Create an E/R diagram which captures the following information:

- One Superhero protects each city
- Every supervillain operates out of a secret lair at some location (not always a city)
- Some cities host multiple supervillains, some host none.
- Every superhero has a supervillain nemesis.
- A superhero and supervillain always fight in some city.
- Each superhero can team up with other superheroes; supervillains form conspiracies.
- Teams of superheroes can battle conspiracies at any location.
- Battles in cities, and nemeses that live in the same city, are especially interesting.

Hint: this is pretty straightforward; just take it one step at a time.

<protects>: [superhero] (0:n); [city] (1:1)

<lair-of>: [location] (0:n); [supervillain] (1:1)

[city] \is-a/ [location]

<nemesis>: [superhero] (1:1); [supervillain] (0:n)

<fight>: [superhero] (0:n); [supervillain] (0:n); [city] (0:n)

<cooperates>: [superhero] (0:n); [team] (2:n)

<conspires>: [supervillain] (0:n); [conspiracy] (2:n)

<battle>: [team] (0:n); [conspiracy] (0:n); [location] (0:n)

5. Functional dependencies and normal forms

Consider the relation and functional dependencies (FDs) given below:

```
R(address, author, genre, publisher, title)

author title -> publisher
genre title -> author
publisher -> address genre
```

a) List all candidate keys for R.

author title -> author title publisher address genre title genre -> title genre author publisher address publisher title -> publisher title address genre author

b) Identify any FD(s) that prevent R from satisfying 3NF.

publisher -> address (address is not prime and does not depend on a whole key) NOTE: publisher -> genre is ok because genre is prime!

6. Outer joins

Consider the following three relations:

R	
a	**b**
0	5
1	2
3	4
0	2
1	3

S		
b	**c**	**d**
3	5	4
1	2	3
0	5	0
2	1	1

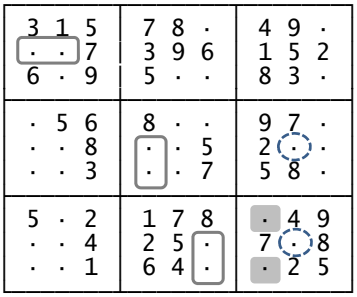
T	
d	**e**
3	2
4	3
5	0
2	1

Populate the leftmost column of the table below by computing (R ⟗ S) ⟗ T – the full outer join of all three relations – using a "-" to represent NULL values where appropriate. Then, for each remaining column, put a check mark by any rows that also appear in the result of the specified operation:

(R ⟗ S) ⟗ T	R ⟗ (S ⟗ T)	(T ⟗$_L$ S) ⟗ R	(T ⟗$_L$ S) ⟗$_L$ R	(R ⟗$_L$ S) ⟗$_L$ T	(R ⟗ S) ⟗$_L$ T
R . S . T					
a b c d e					
0 5 . . .				*	
1 2 1 1 .				*	*
3 4 . . .				*	
0 2 1 1 .				*	*
1 3 5 4 3	*	*	*	*	*
. 1 2 3 2			*		
. 0 5 0 .					
. . . 5 0			*		
. . . 2 1			*		

7. Sudoku revisited

In the Sudoku puzzle to the right, the gray-filled cells form a "locked" 3-6 pair. The circled cells also form a locked pair (1-6), and empty boxes mark three more. The locked numbers cannot appear in other cells of a locked pair's group (row/col/block). Example: the gray 3-6 pair "shadows" the nearby circled cell (id:70), so cell 70 cannot legally take the value 6. This is useful because deleting candidate (70,6) solves the cell: only candidate (70,1) remains.

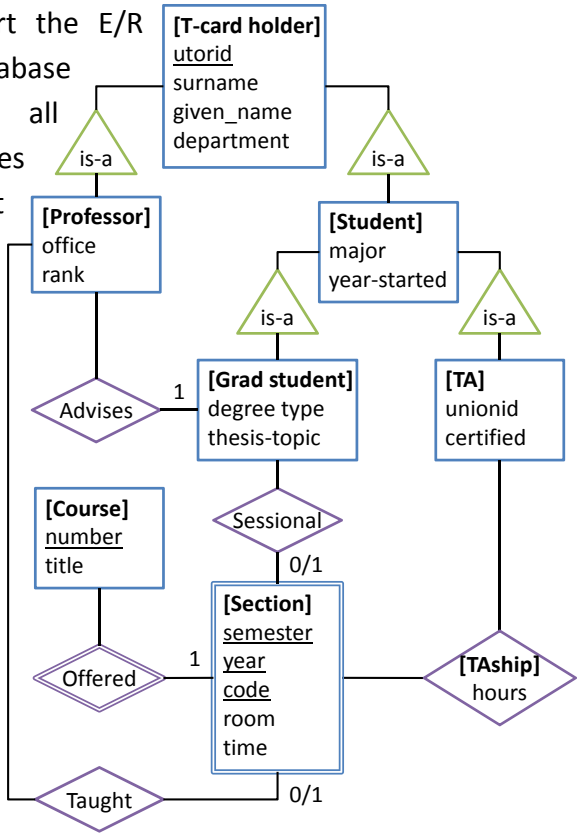


We developed a query in class to find shadowed candidates like (70,6) that should be deleted. Now do the reverse: write a query that returns the ids of any locked pair(s) – ***and only those pairs*** – which require us to delete candidate (70,6). In the example above, only the gray-filled pair should be returned, even though other locked pairs shadow other candidates as well. Assume that CN(id,n) contains all remaining candidates and C(id, x, y, b) maps cell ids to locations on the board. Your solution may create views if that helps.

```
create view Pairs as
    select id, x, y, b, n1, n2 from C join (
       select id, min(n) n1, max(n) n2 from CN
       group by id having count(n)=2
    ) using(id) where n1=6 or n2=6 and id != 70;

select p1.id, p2.id from Pairs p1, Pairs p2, C
where p1.id < p2.id and C.id=70 and (
    (p1.x=p2.x and p1.x=c.x) or (p1.y=p2.y and p1.y=c.y) or (p1.b=p2.b and
    p1.b=c.b)
);
```

8. E/R diagrams, relational schemata, and constraints

a) Using the space on the next pages, convert the E/R diagram on the right into an efficient SQL database schema. Your schema should incorporate all constraints specified in the diagram, and does not need to incorporate any others. Correct schemata defining fewer than ten tables earn bonus marks (eight tables is possible).



Answers should be formatted nicely, wrapping and indenting if they're longer than one line. To save space, you may shorten long table and attribute names and abbreviate SQL syntax. The intent must be clear, however. As an example, the tables below would be considered equivalent for marking purposes:

```
CREATE TABLE movie_star(
   star_name PRIMARY KEY,
   studio_id REFERENCES studio(id)
); -- too long for one line
```

```
TABLE star(name PK, sid REF studio(id)); -- short enough for one line
```

b) Assume that CHECK constraints do *not* allow nested queries. Extend your schema from part (a) to enforce the following additional constraints:

- Course titles are unique, as are TA unionids.
- Semesters are "F" or "W" and TAship hours are between 10 and 70
- Only graduate TAs can hold more than three TAships (lifetime total).
- A professor and a sessional cannot both teach the same section.

c) Extend your schema from part (a) with views that make the following information easily accessible to a web application that uses the database:

- A listing of graduate TAships (TA name, course code, semester, year, etc.).
- A full description (location, instructor, course title) of all sections offered.
- Instructors (whether professor and sessional) and semesters when they have taught.

d) You may assume that all primary and foreign key fields already have indexes. Create additional indexes as needed to accelerate searches for the following:

- people by surname
- sections by year and semester

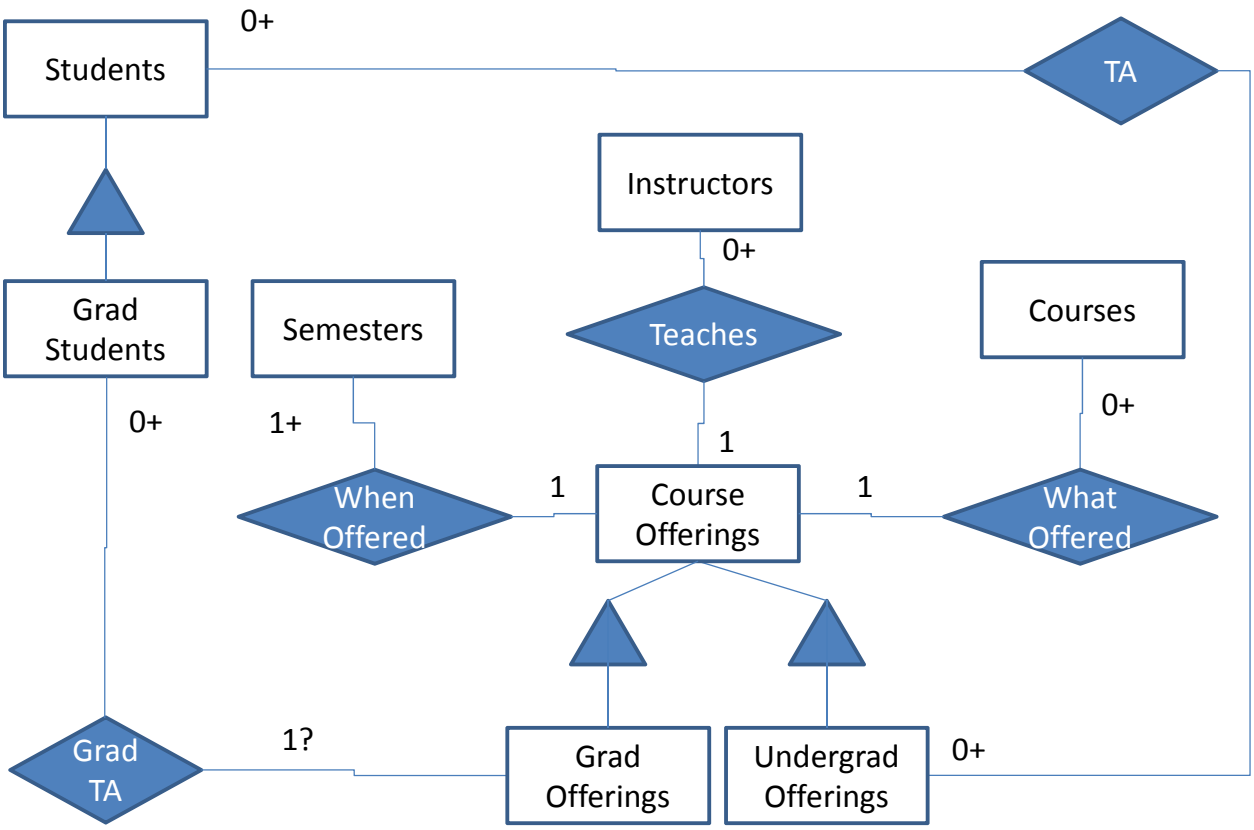
```
table tcard_holder (
        utorid PK,
        name,
        dept)

table professor(
        utorid PK ref tcard_holder(utorid),
        office,
        rank)

table student(utorid PK ref tcard_holder(utorid),
        major,
        year_started)

table ta(utorid PK ref student(utorid),
        unionid unique,
        certified check(certified="yes" or certified="no")

table grad(
        utorid PK ref student(utorid),
        advisor ref professor(utorid) not null,
        degtype,
        topic)

table course(
        number PK,
        title unique)

create index person_surname on tcard_holder(surname)
create index section_semester on section(year, semester);

table section(
        course ref courses(num),
        semester,
        year,
        name,
        prof ref professor(utorid),
        sessional ref grad(utorid),
        room,
        time,
        PK (num,name,semester,year),
        check ((prof is NULL) != (sessional is NULL))
)
```

```
table TAship(
        utorid ref TA(utorid),
        course,
        semester,
        year,
        code,
        hours check(hours between 10 and 70),
        FK (course, semester, year, code)

        PK(utorid,course,semester,year,code))

create trigger ugrad_ta_max before insert on taship
where not exists(select new.utorid from grad)
        and (select count(*) from taship

begin
        select raise(ABORT, "Too many taships for an undergrad");
end;
```

9. More E/R diagrams

Consider the following (perhaps unrealistic) aspects of a university. Create an E/R diagram which captures the constraints below. When necessary, be sure to restrict cardinality (you may use "1" (exactly 1), "1+" (at least one), "0+" (zero or more) and "1?" (zero or one) if you wish. Include with each entity set one or more attributes of the real-life object which make a reasonable candidate key (don't just create an arbitrary "ID" attribute out of thin air).

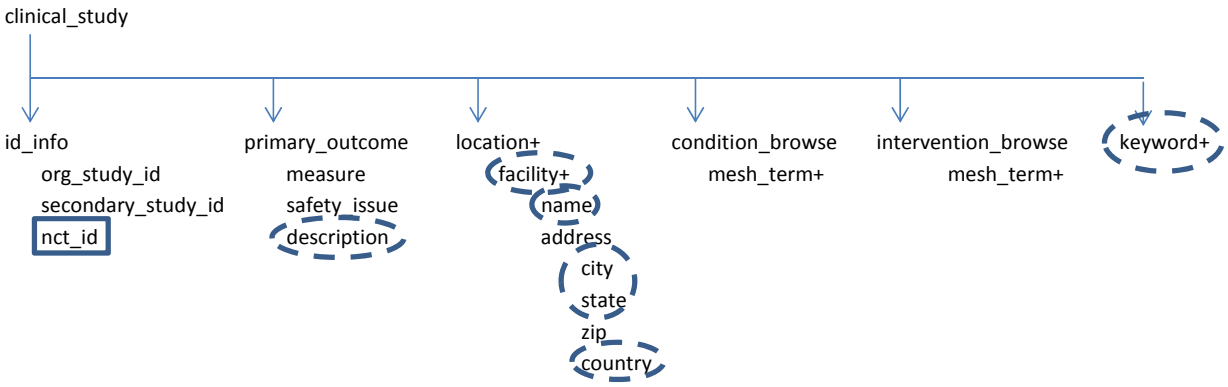
- A course has exactly one instructor every semester it is offered
- An instructor may teach several courses in a semester
- Each undergraduate course offering must have at least one TA
- Graduate courses can have at most one TA
- Only graduate students may TA graduate courses



It doesn't matter what key you choose for each entity set, as long as it's reasonable; UTorID for Students, S12 F13, etc. for Semesters; course codes (CSCC43, etc.) for Courses, and so on. Subclasses (GradStudents, UndergradOfferings, etc.) don't need their own key because they use the key of their parent class.

10. Database operations in XQuery

Suppose you are given a variable, **`$input`**, which contains a sequence of **`clinical_study`** elements structured according to the diagram below:



a) Write an XQuery expression that returns any study that mentions "cancer" as part of a keyword, condition, or outcome (assume string tests are case insensitive).

```
//clinical_study[.//(keyword|condition|outcome)[matches(.,'cancer')]]
```

b) Write an XQuery expression that converts **`$input`** into a sequence of **`study`** elements containing only the elements circled on the diagram. The **`nct_id`** should become an attribute of **`study`**. If the parent of an element is not selected, make that element a child of the lowest ancestor that is selected.

```
for $s in $input
let $f :=          (for $x in $s//facility
                    return <facility>{
                          $x/name,
                          $x/address/(city|state|country)
                    } </facility>)
return <study nct_id="{$s//nct_id}"> {
                    $s/primary_outcome/description,
                    $f,
                    $s/keyword
}</study>
```

c) Give an Xquery expression that identifies the `city` involved in the most clinical studies? Give the city's name and study count, breaking ties by returning all such cities. NOTE: count studies, not locations.

```
let $counts := (
        let $cities := local:distinct ($input//location//city)
        for $c in $cities
                let $n := count($input/clinical_study[location//city =
                $c])
                order by $n descending,$c
                return <city count="{$n}">{$c}</city>)
return $counts[@count = $counts[1]/@count]
```

d) List pairs of cities which are joint locations of any study. Keep in mind that many studies have 3+ locations. No pair of cities should appear more than once in the output.

```
for $s in $input
        let $c := $s//location//city
        for $c1 in $c, $c2 in $c
                where $c1 < $c2
                return
                <pair>{$c1,$c2}</pair>
```