EXAM
Databases (INN120/TDA355/TDA356/TDA357)
Informationssystem (TDA426)

DAY: 13 Mar 2008   TIME: 14:00 – 18:00   PLACE: M

Responsible:	Niklas Broberg, Computing Science mobil 0706 49 35 46
Results:	Will be published on the course web page right after then exam
Extra aid:	A single, hand-written A4 paper. It is legal to write on both sides. This paper should be handed in with the exam.
Grade intervals:	**U**: 0 – 23p, **3**: 24 – 35p, **4**: 36 – 47p, **5**: 48 – 60p, **G**: 24 – 41p, **VG**: 42 – 60p, **Max.** 60p.

IMPORTANT

The final score on this exam is computed in a non-standard way. The exam is divided into 7 blocks, numbered 1 through 7, and each block consists of 2 or 3 levels, named A, B, and optionally C. A level can contain any number of subproblems numbered using i, ii and so on. In the final score you can only count **ONE** level from each block. For example: if you attempt to solve the problems on all three levels in block 4 and manage to obtain 4 points for 4A (block 4, level A), 1 point for 4B and 8 points for 4C, only problem 4C (where you got your highest score) will count towards your final result, so your score for block 4 will be 8 points.

The score for each problem depends on how difficult it is (more points for harder problems) and how important I think it is (more points for more important problems). It does *not* depend on how much work it takes to answer the problem. There could very well be a 12 point problem that takes 15 seconds to answer (given that you know the right answer, of course).

The problems in each block are ordered by increasing difficulty. Hence the A problems are easy, but aim to cover the full basics of its area. The B and C level problems are more difficult, and aim to test your knowledge of the areas beyond the mere basics. If you only solve A problems your maximum score is 36 points, and if you only solve the B problems where there are also C problems it is 40 points.

Please observe the following:

- Answers can be given in Swedish or English
- Use page numbering on your pages
- Start every assignment on a fresh page
- Write clearly; unreadable = wrong!
- Fewer points are given for unnecessarily complicated solutions
- Indicate clearly if you make assumptions that are not given in the assignment

Good advice

- Most problems have been designed to give short answers. Few problem should require more than one page to answer.
- There are more problems than you are likely to solve in 4 hours. This means that you have to think about which problems you attempt to solve. If you try solve the problems in the order they are given, **you are likely to fail the exam**!

Good Luck!

Block 1 - Entity-Relationship Diagrams **max 12p**

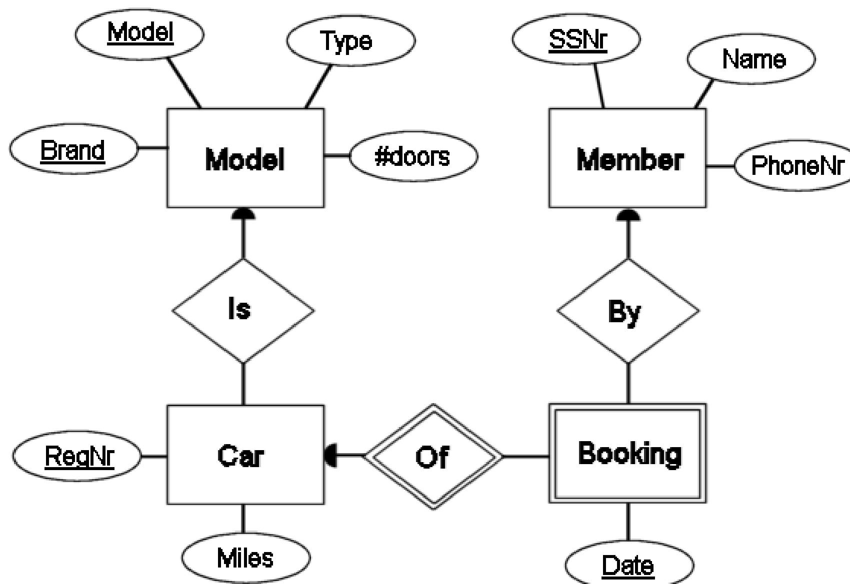
1A (8p)

(i) (4p)

A small carpool organization wants a database to handle booking of cars, and want to store information about vehicles and members. Regarding members, the organization wants to store the social security number, name and phone number. The pool consists of cars of a few different models, and for each model they want to store information regarding the brand and model name, number of doors, and type (combi or sedan). For each individual car the database should store the registration number, what model it is, and how many miles it has run (updated after each use). The cars are bookable on a per-day basis, each member being allowed to book a car a specific number of times per month. For each booking, the organization wants to store which car that was booked, by whom, and for what day.

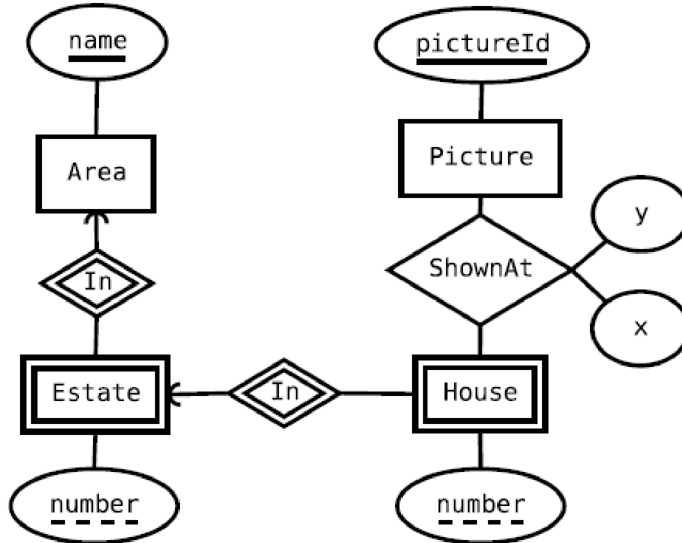
Your task is to draw an ER diagram that correctly models this domain and its constraints.

Solution:



(ii) (4p)

Here is an ER diagram describing the domain of land surveys, consisting of estates as well as pictures that those estates appear on.



Translate this ER diagram into a set of relations, clearly marking keys and references in your answer.

Solution:

```
Areas(_name_)
Estates(_area_, _number_)
  area -> Areas.name
Houses(_area_, _estate_, _number_)
  (area, estate) -> Estates.(area, number)
Pictures(_id_)
ShownAt(_picture_, _area_, _estate_, _house_, x, y)
  picture -> Pictures.id
  (area, estate, house) -> Houses.(area, estate, number)
```

1B (12p)

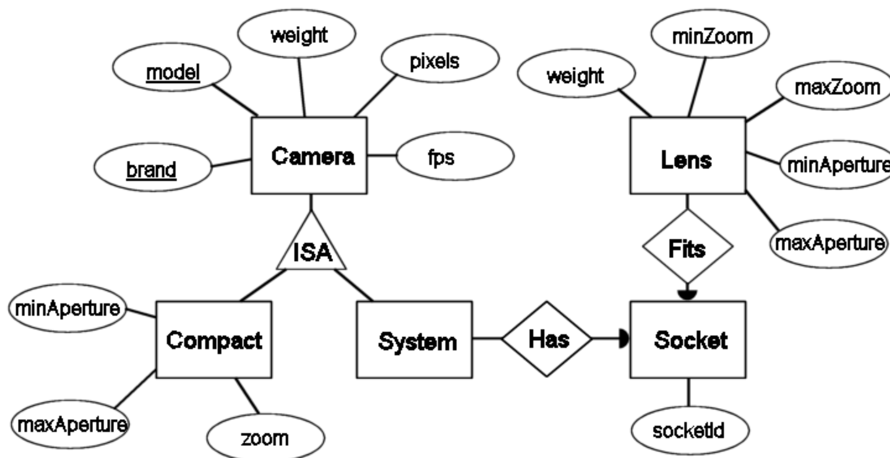
A camera retailer wants a database to hold information about the various digital camera models they sell. The cameras they sell come in two kinds - compact cameras and system cameras. Compact cameras are those with all the components integrated, whereas system cameras have exchangeable lenses. This is a crude simplification of the domain, but for our purposes, and those of the retailer, it is good enough.

All camera models have a brand name and a model name. The retailer also wants to store information about pixels, zoom, frames per second, min and max aperture values and weight. For compact cameras, all those values are dependent on the camera alone. For system cameras, only pixels, weight and frames per second depend on the camera, while aperture values and zoom depend on the lens. The lens also has a weight that should be stored. For compact cameras, only the maximum zoom value should be stored, while for lenses both min and max values are needed.

Furthermore, the database must hold information regarding what lenses fit what cameras. System cameras have a socket, and lenses are built to fit some particular socket. The database should hold a list of the sockets used, as well as what socket each camera has and what socket each lens fits on.

Your task is to draw an ER diagram that correctly models this domain, and then translate it into relations. Don't forget to mark keys and references in your answer.

Solution:



```
Cameras(_brand_,_model_,weight,pixels,fps)
Compacts(_brand_,_model_,zoom,minAperture,maxAperture)
  (brand,model) -> Cameras(brand,model)
Systems(_brand_,_model_,socket)
  (brand,model) -> Cameras(brand,model)
  socket -> Sockets.socketId
Lenses(_model_,weight,minZoom,maxZoom,minAperture,maxAperture,socket)
  socket -> Sockets.socketId
Sockets(_socketId_)
```

Block 2 - Dependencies and Normal Forms **max 12p**

2A (8p)

A mid-sized private clinic uses a database to keep track of their patients, appointments and treatments. They have experienced some problems though, and have called on a database expert (you) to help them.

Regarding patients, the database stores their social security number, name and phone number. For appointments they store the doctor's id number (unique within the clinic) and name, the time of the appointment, and the patient's social security number and phone number to call in case something comes up. The clinic must also keep track of all prescriptions of medicines. For medicines the database stores the medicine's id number, name, the name of the producer company and the phone number of the producer. For prescriptions they store the medicine's id and name and the dose to be taken, the doctor that prescribed it, the patient that got it, and what date the prescription was made. Each prescription is also given a unique identification number.

Their database has the following schema:

Patients(<u>*ssnr*</u>, *name*, *phone_nr*)
Appointments(<u>*doctor_id*</u>, <u>*time*</u>, *doctor_name*, *pat_id*, *pat_phone*)
    pat_id → *Patients.ssnr*
Medicines(<u>*idnr*</u>, *name*, *producer*, *prod_phone*)
Prescriptions(<u>*nr*</u>, *date*, *doctor_id*, *patient*, *medicine*, *med_name*, *dose*)
    patient → *Patients.ssnr*
    medicine → *Medicines.idnr*

You of course immediately see that the schema is not fully normalized, and thus it's no wonder that they suffer from a number of problems. Your task is now to solve these by normalization of the schema.

(i) (3p)
For the given domain, identify all functional dependencies that you expect to hold.
Solution:

```
ssNr -> name, patPhone
doctorId -> doctorName
doctorId, time -> ssNr
medIdNr -> medName, medProducer
medProducer -> prodPhone
prescNr -> date, doctorId, ssNr, medIdNr, dose
```

What names you use for the attributes here don't matter, if there is a reference between them then they are the same attribute.

(ii) (1p)

With the dependencies you have found, identify all BCNF violations in the relations of the database.

Solution:

```
Patients: no violations
Appointments: ssNr -> patPhone, doctorId -> doctorName
Medicines: medProducer -> prodPhone
Prescriptions: medIdNr -> medName
```

(iii) (2p)

Give an example of a deletion anomaly that could occur because of one of the violations.

Solution: If a certain doctor had no appoinments planned, there would be no place in the database to store that doctor's name.

(iv) (2p)

Do a complete normalization of the schema, so that all relations are in BCNF. You do not need to show all steps of the normalization, only the end result schema is fine.

Solution:

```
Patients(_ssNr_,name,phoneNr)
Doctors(_doctorId_, name)
Appointments(_doctor_,_time_,patient)
  doctor -> Doctors.doctorId
  patient -> Patients.ssNr
Medicines(_idNr_,name,producer)
  producer -> Producers.producer
Producers(_producer_,prodPhone)
Prescriptions(_nr_,date,doctor,patient,medicine,dose)
  doctor -> Doctors.doctorId
  patient -> Patients.ssNr
  medicine -> Medicines.idNr
```

2B (12p)

When the clinic (same as above) realizes how good you are, they decide to hire you to extend the database to keep track of their employees as well.

The clinic is organized into a set of wards, each with its own staff, though a doctor could be assigned to several wards. A specific ward is identified by a designated number, and this should be stored together with the phone number to the ward. For doctors, the database should store their id and name, as well as their specialization (e.g. orthopaedian, neurologist etc) and the wards they serve on. For nurses the database should store an id and name, and the ward the nurse belongs to.

Further, the database should keep track of the expensive equipment the clinic uses. Any given piece of equipment belongs to a particular ward. Further, the database should store the name and phone number of the company that maintains that equipment, in case something happens to it. Several pieces of equipment could be maintained by the same company.

They also want the information about patients and medication from section A, but for simplification we leave that out here, it will look the same (and you only need to answer one of the sections).

The following relation sums up all the attributes that should be stored in (this part of) the database:

$$Clinic(ward, ward_phone, doctor_id, doctor_name, doctor_spec, nurse_id, nurse_name, equipment, maint_name, maint_phone)$$

Your task is to use normalization techniques to find a suitable schema for this database.

(i) (8p)

Find all dependencies, both functional and multi-valued, that you expect should hold for this domain given the domain description above.

Solution:

```
ward -> wardPhone
ward ->> doctorId, doctorName, doctorSpec
ward ->> nurseId, nurseName
ward ->> equipment, maintName, maintPhone
doctorId -> doctorName, doctorSpec
nurseId -> nurseName, ward
equipment -> maintName
maintName -> maintPhone
```

(ii) (4p)

Do a complete decomposition of *Clinic* so that the resulting schema fulfills 4NF, without breaking any of the dependencies you identified.
If you find that you must break some dependency then either you are doing something wrong, or you have identified some dependency that we did not think to include. If you think it is the latter case, explain why this is the case, and normalize anyway.
Solution: To not break any dependencies, I will begin to decompose based on the MVDs, with ward as the LHS.

```
Wards(_ward_,wardPhone)
WardDoctors(_ward_,_doctorId_,doctorName,doctorSpec)
WardNurses(ward,_nurseId_,nurseName)
WardEquipment(ward,_equipment_,maintName,maintPhone)
```

From there I can decompose based on the FDs to get the following end result:

```
Wards(_ward_,wardPhone)
Doctors(_doctorId,doctorName,doctorSpec)
WardDoctors(_ward_,_doctor_)
  doctor -> Doctors.doctorId
Nurses(_nurseId_,nurseName,ward)
  ward -> Wards.ward
Equipment(_equipment_,ward,maintName)
  ward -> Wards.ward
  maintName -> Maintainers.name
Maintainers(_name_,phone)
```

Block 3 - SQL Data Definition Language **max 8p**

The domain for this block, and for several following blocks as well, is that of a database used for an online image blog community. Each user has their own image blog, where they can upload images for particular days and write a text entry for each image. Other users in the community can leave comments under each image. Further, a user may create albums to organize their images. An album can be thought of as a way to give quick access to a set of related image entries, but the primary organization of images is the days of their entries.

You are given the following schema of their intended database:

$Users(\underline{username}, name, email)$
$Images(\underline{image_nr}, filename, user, title, year, month, day)$
    $user \rightarrow Users.username$
    $(year, month, day)$ *forms a valid date*
    filename is unique
$Albums(\underline{album_name}, \underline{user}, description)$
    $user \rightarrow Users.username$
$InAlbum(\underline{image_nr}, \underline{album_name}, \underline{user})$
    $image_nr \rightarrow Images.image_nr$
    $(album_name, user) \rightarrow Albums.(album_name, user)$
    Owner of image must be the same user as owner of album
$Comments(\underline{image}, \underline{nr}, user, time, text)$
    $image \rightarrow Images.image_nr$
    $user \rightarrow Users.username$

3A (4p)

Write SQL DDL code that correctly implements these relations as tables in a relational DBMS. Make sure that you implement all given constraints correctly. Do not spend too much time on deciding what types to use for the various columns. We will accept any types that are not obviously wrong. Don't forget to implement all specified constraints, including checks.

Solution:

```
CREATE TABLE Users (
 username VARCHAR(20) PRIMARY KEY,
 name VARCHAR(50),
 email VARCHAR(50)
);
CREATE TABLE Images (
 image_nr INT PRIMARY KEY,
 filename VARCHAR(50) UNIQUE,
 user REFERENCES Users(username),
 title VARCHAR(100),
 year INT, month INT, day INT,
 CHECK (VALID_DATE(year,month,date))
);
CREATE TABLE Albums (
 album_name VARCHAR(50),
 user REFERENCES Users(username),
 description VARCHAR(100),
 PRIMARY KEY (album_name, user)
);
CREATE TABLE InAlbum (
 image_nr REFERENCES Images(image_nr),
 album_name, user,
 FOREIGN KEY (album_name,user) REFERENCES Albums(album_name,user),
 CHECK (user = (SELECT user FROM Images I WHERE I.image_nr = image_nr))
);
CREATE TABLE Comments (
 image REFERENCES Images(image_nr),
 nr INT,
 PRIMARY KEY (image, nr),
 user REFERENCES Users(username),
 time DATE,
 text VARCHAR(1000)
);
```

3B (8p)

Consider the code for creating the *Albums* table, and consider the following four different

ways of implementing the reference:

```
CREATE TABLE Albums (
  ...
  user VARCHAR(20) REFERENCES Users(username),
  ...
);

CREATE TABLE Albums (
  ...
  user VARCHAR(20),
  user REFERENCES Users(username),
  ...
);

CREATE TABLE Albums (
  ...
  user VARCHAR(20) CHECK (user IN (SELECT username FROM Users)),
  ...
);

CREATE TABLE Albums (
  ...
  user VARCHAR(20),
  CHECK (user IN (SELECT username FROM Users)),
  ...
);
```

Explain the differences, if any, between these four different statements, in terms of what modifications that can be performed. (NOTE: The last two checks cannot be specified with Oracle - that's not what we're asking about though, we want to know how they affect modifications, assuming they could be written).

Solution: For all of them, the same modifications can be performed on the Albums table, namely whenever we insert or update a row, the system ensures that the corresponding value of user is in the Users table. The difference is in what operations may be performed on the Users table.

There is no difference at all between the first two. For both of them, we can only delete rows from Users if there are no rows in Albums that depend on that value for user. Same thing for updates.

There is very little difference between the last two, for this particular example. For both of them, there will be no check done when the rows in Users are modified, the checks will only happen when the Albums table itself is modified. The only difference between the last two

is that for the first of them the check will only be done when the user attribute is updated, whereas the second check would be done when *any* attribute is updated.

Block 4 - SQL Queries **max 8p**

Use the relations for the image blog community from the previous block when answering the following problems.

4A (4p)

(i) (1p)

Write an SQL query that lists all images labelled with May 2007, together with the username and name of the owner.

Solution:

```
SELECT *
FROM Images, Users
WHERE user = username
  AND month = 'May'
  AND day = '15';
```

(ii) (3p)

Write an SQL query that lists all albums together with the number of images they contain. Albums with no images should be listed either with 0 or with NULL as their number of images.

Solution:

```
SELECT album_name, user, COUNT(image)
FROM (SELECT album_name, user, image
      FROM Albums LEFT OUTER JOIN Images ON image = image_nr);
```

4B (6p)

Write a query that for each pair of users X and Y lists the number of images belonging to X that Y has left a comment on. The result should contain three columns – X, Y and the number of images. Do not include the case where X = Y. Also do not include those pairs where the number is 0. Order the result by username X and number of comments descending.

Solution

```
SELECT I.user as X, C.user as Y, count(*)
FROM Images I, Comments C
WHERE I.image_nr = C.image
  AND I.user <> C.user
GROUP BY I.user, C.user;
```

4C (8p)

Write a query that lists the user (or users) with the hightest average number of comments on their images. In case of a tie, list all those users that are tied for first place.

Solution:

```
WITH
  Counts AS
   (SELECT image_nr, username, COUNT(nr) as nr
    FROM Images, Comments
    WHERE image = image_nr
    GROUP BY image_nr, username),
  Averages AS
   (SELECT username, AVG(nr) as avg
    FROM Counts
    GROUP BY username)
SELECT username, avg
FROM Averages
WHERE avg = (SELECT MAX(avg) FROM Averages);
```

Block 5 - Relational Algebra **max 6p**

Use the relations for the image blog site from the previous blocks when answering the following problems.

5A (4p)

(i) (1p)
What does the following relational-algebraic expression compute (answer in plain text):

$$\tau_{-nr}(\gamma_{month,COUNT(*)\rightarrow nr}(Images))$$

Solution: The number of images uploaded per each month of the year, ordered so that the month with the highest count is listed first.

(ii) (3p)
The following relational-algebraic expression returns the list of all albums containing 100 or more images. For each such album, the album name, description and number of images are projected. Translate it to a corresponding SQL query:

$$\pi_{album_name,nr,description}(\sigma_{nr>=100}(\gamma_{A.album_name,A.user,description,COUNT(image_nr)\rightarrow nr} \\ (\sigma_{A.user=I.user A.album_name=I.album_name}(\rho_A(Albums) \bowtie \rho_I(InAlbum))))) \quad (1)$$

Solution: First of all, there's an error, it should have been a normal cross-product and not a natural join. We will be nice when we correct it, so no matter how you read it you will get points (if it's otherwise correct).

```
SELECT A.album_name, COUNT(image_nr) as nr, description
FROM Albums A, InAlbum I
WHERE A.user = I.user
  AND A.album_name = I.album_name
GROUP BY A.album_name, A.user, description
HAVING COUNT(image_nr) > 100;
```

5B (6p)

Write a relational algebra expression that lists all images together with the albums they belong to (once for each album, each images could belong to several albums). Images not belonging to any album should also be listed, with NULL as album name.

Solution: Ok, I have a problem here, I don't know how to get this program to give me the outer join symbol. So I'll just write here that the natural join symbol below should really be an outer join.

$$\pi_{image_nr,album_name,user}(Images \bowtie_{image_nr=image} InAlbum)$$

Block 6 - Transactions **max 6p**

Use the relations for the image blog community from the previous blocks when answering the following problems.

6A (4p)

Assume images are uploaded and inserted via a web application that (somewhat stupidly) performs the following operations:

```
... user identified through current_user, file uploaded to new_file...
1 new_image_nr := SELECT MAX(image_nr) FROM Images;
2 INSERT INTO IMAGES VALUES (new_image_nr, new_file, current_user, ...)
```

(i) (1p)
For the program specified above, what atomicity problems could arise if it was not run as a transaction?
Solution: None at all. We only do one write, so there is no way the program can crash in the middle. If it crashes after 1, nothing bad will have happened.

(ii) (1p)
For the program specified above, what isolation problems could arise if it was not run as a transaction at a sufficiently restrictive isolation level?
Solution: First of all, there's an error in the formulation, there's a +1 missing after MAX(image_nr).
We could get the problem that two (or more) parallel transactions both generate the same new"image number, and both try to insert an image with that number (all but one will fail due to the uniqueness constraint).

(iii) (2p)
Which of the four possible isolation levels would solve both of these problems (more than one answer possible)?
Solution: Only SERIALIZABLE would solve the isolation problem.

6B (6p)

Assume that the user may specify an album to add the image to when uploading it, and consider the following sequence of operations:

```
... current_user, new_file and new_image_nr as above...
... user supplies chosen_album...
1 INSERT INTO InAlbum VALUES (new_image_nr, chosen_album, current_user);
2 INSERT INTO Images VALUES (new_image_nr, new_file, current_user, ...);
```

If this program is not run as an assertion, which of the four ACID properties could give

problems in this case (more than one answer possible), and how?

Solution: A — Atomicity: If the program crashes after the first insert, we have an image in an album that doesn't exist (won't happen because of C though). C — Consistency: All constraints must be preserved *at the end points* of the transaction. If this was not a transaction, 1 would fail because of the foreign key constraint between InAlbum and Images. If we did run a transaction, the checking of that constraint would be deferred until after the transaction has completed. I — Isolation: No problems, since we never read anything we're not dependent on what other operations may do. D — Durability: Not applicable.

Block 7 - Semi-structured Data and XML **max 8p**

A school wants to try something new for their course portal, and to allow a bit more flexibility with their information they have decided to try a semi-structured data model, and to interface their data as XML.

You have come up with the following DTD as a schema for their new database:

```
<!DOCTYPE CoursePortal [

<!ELEMENT CoursePortal (Course*, Student*)>
<!ELEMENT Course       (Assistant*)>
<!ELEMENT Assistant    CDATA>
<!ELEMENT Student      (RegCourse*)>
<!ELEMENT RegCourse    EMPTY>

<!ATTLIST Course
    code    ID #REQUIRED
    name    CDATA #REQUIRED
    teacher CDATA #IMPLIED>
<!ATTLIST Student
    ssNr ID #REQUIRED
    name CDATA #IMPLIED>
<!ATTLIST RegCourse
    code IDREF #REQUIRED>

]>
```

7A (4p)

(i) (2p)

Give a minimal XML document that contains information about at least one student who is registered to at least one course, and is valid with respect to the given DTD. Note that minimal does not refer to the length of strings given, but the number of elements and attributes in the document.

Solution:

```
<CoursePortal>
 <Course id="TDA357" name="Databases" />
 <Student ssNr="841224-0123">
  <RegCourse code="TDA357" />
 </Student>
</CoursePortal>
```

(ii) (2p)

For a document conforming to the schema given above, what would the following XQuery expression compute? Answer in plain text:

```
FOR $c IN //Course
LET $x := count($c/*)
WHERE $x > 2
ORDER BY -($x)
RETURN ({$c})
```

Solution: It would return all courses having more than two assistants, ordered so that the course with most assistants comes first.

7B (6p)

For a document conforming to the schema given above, what would the following XQuery expression compute? Answer in plain text, and give an example of a returned XML document:

```
LET $cs := (
  FOR $c IN //Course
  LET $ss := (
    FOR $s IN //Student
    WHEN $s/RegCourse/@code = $c/@code
    RETURN <Student ssNr=({$s/@ssNr}) /> )
  RETURN <Course id=({$c/@id})>({$ss})</Course> )
RETURN <Courses>({$cs})</Courses>
```

Solution: It returns all courses, with the registered students as children. Example:

```
<Courses>
```

```
 <Course id="TDA357">
  <Student ssNr="841224-0123" />
  ... more students ...
 </Course>
 ... more courses ...
</Courses>
```

7C

(8p)

Write an XQuery expression that returns a list of all students together with the number of courses they read. The result should be on the following form:

```
<StudentList>
  <Student ssNr="841224-0123">3</Student>
  ... more students ...
</StudentList>
```

You can use the function COUNT(X) to count the elements in a set X.

Solution: First of all, the number of courses that they read means the number they are registered to.

```
LET $ss := (
  FOR $s IN //Student
  LET $n := COUNT($s/*)
  RETURN <Student ssNr=({$s/@ssNr})>({$n})</Student> )
RETURN <StudentList>({$ss})</StudentList>
```