Collection of database exam solutions

Rasmus Pagh

August 27, 2012

This is a supplement to the collection of database exams used in the course *Introduction to Database Design*, which includes answers. The idea is that it can be used to:

- Check your own solutions against.
- Get an impression of what is required for a written solution to be considered complete.

In the solution for problem 1 of January 2006, the rounded arrow indicates a participation constraint (corresponding to a bold arrow in RG).

Introduction to Database Design

IT University of Copenhagen

January 3, 2012

The exam consists of 5 problems with 13 questions in total. It has 13 pages (including this page). **It is recommended to read the problems in order**, but it is not important that you solve them in any specific order. Your answer is supposed to fit on the problem sheet itself. If you need more space, or need to replace your answer, use ordinary paper (one question per sheet, clearly referenced on the problem sheet).

The weight of each problem is stated. You have 4 hours to answer all questions. If you cannot give a complete answer to a question, try to give a partial answer.

"KBL" refers to the course book "Database Systems - an application approach, 2nd edition (Introductory Version)", by Michael Kifer, Arthur Bernstein and Philip M. Lewis.

All written aids are allowed.

1 Data modeling (30%)

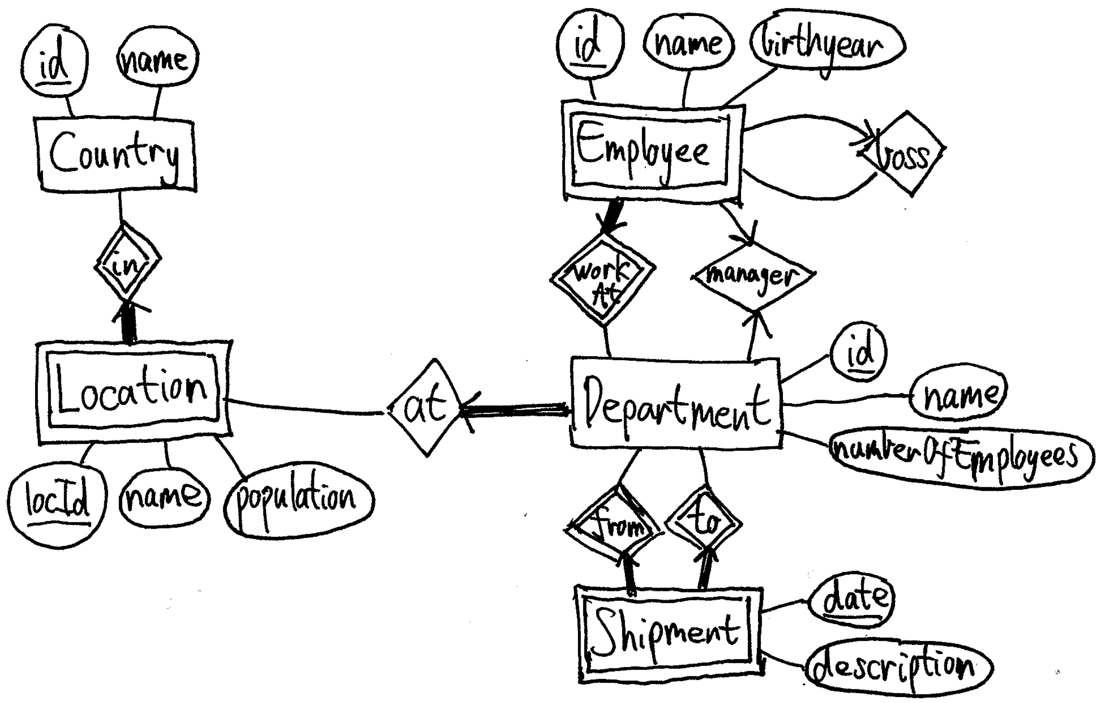
Consider the following SQL DDL schema:

```
CREATE TABLE Country (
   id INT PRIMARY KEY,
   name VARCHAR(50)
);
CREATE TABLE Location (
   locId INT,
   countryId INT NOT NULL REFERENCES Country(id),
   name VARCHAR(50),
   population INT,
   PRIMARY KEY (locId,countryId)
);
CREATE TABLE Department (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   numberOfEmployees INT,
   location INT NOT NULL,
   country INT NOT NULL,
   manager INT,
   FOREIGN KEY (location,country) REFERENCES Location(locId,countryId),
   UNIQUE KEY (manager)
);
CREATE TABLE Employee (
   id INT,
   name VARCHAR(50),
   birthYear INT,
   boss INT REFERENCES Employees(id),
   worksAt INT NOT NULL REFERENCES Department(id) ON DELETE CASCADE,
   PRIMARY KEY (id,worksAt)
);
CREATE TABLE Shipment (
   fromDepartment INT REFERENCES Department(id),
   toDepartment INT REFERENCES Department(id),
   date DATE,
   description VARCHAR(500),
   PRIMARY KEY (fromDepartment,toDepartment,date)
);

ALTER TABLE Department ADD FOREIGN KEY managerIsInDepartment
(manager,id) REFERENCES Employee(id,worksAt);
```

a) Draw an ER diagram, using the notation from KBL, corresponding to the schema above. You should put emphasis on expressing as many constraints of the schema as possible in your ER diagram (e.g., primary, foreign key, and participation constraints), while not further restricting the data that is allowed.

Draw your answer here:



b) Suppose that we want the database to store not just the current set of employees, the boss(es) of each employee, and where each employee works, but the *history* of employments. That is, it should be possible to ask a query that returns the contents of `Employees` and `Department` for a given time in the past. Describe a revised data model, using SQL DDL, that makes this possible, and briefly explain how it will be used. Primary and foreign key constraints should be specified.

WRITE YOUR ANSWER HERE:

Several solutions are possible. One is to keep the current schema to store the current data, and add relations to store historical data. We assume that the only changes to `Department` that we wish to record is change of manager, and that there is at most one such change per day. Similarly, we assume that for `Employee` we only wish to track changes to where they work, and who is their boss. This is achieved by the following relations:

```
CREATE TABLE DepartmentManagers (
departmentId INT REFERENCES Department(id),
from DATE,
to DATE,
managerId INT REFERENCES Employee(id),
PRIMARY KEY (departmentId,from)
);

CREATE TABLE EmployeeHistory (
employeeId INT REFERENCES Employee(id),
from DATE,
to DATE,
bossId INT REFERENCES Employee(id),
worksAt INT REFERENCES Department(id),
PRIMARY KEY (employeeId,from)
);
```

2 XML (25%)

The below XML document contains part of a database using the schema of problem 1. The employees of each department are represented in a tree structure where the parent node of each employee node is the boss of that employee.

```
<d:database xmlns:d="http://database.org">
   <d:country name="Denmark">
      <d:location id="23" population="100000">
         <d:department id="2301">
            <d:name>SuperOne Ørestad</d:name>
            <d:employee id="42" name="Aaron Aardwark">
               <d:employee id="43" name="Chris Cat"/>
               <d:employee id="44" name="Dolly Dove"/>
            </d:employee>
         </d:department>
         <d:department id="2302">
            <d:name>SuperOne Dragør</d:name>
            <d:employee id="52" name="Edwin Eagle">
               <d:employee id="53" name="Frank Fish">
                  <d:employee id="54" name="Garth Gorilla"/>
               </d:employee>
            </d:employee>
         </d:department>
      </d:location>
      <d:location id="27">
         <d:department id="2701">
            <d:name>SuperOne Grindsted</d:name>
            <d:employee id="62" name="Harry Horse"/>
         </d:department>
  </d:location>
   </d:country>
   <d:country name="Iceland">
      <d:location id="31" population="120000">
         <d:department id="3101">
            <d:name>SuperOne Kringlan</d:name>
            <d:employee id="44" name="Dolly Dove"/>
         </d:department>
      </d:location>
   </d:country>
</d:database>
```

a) Write the results of running the following XPath queries on the document above.

1. `//country/@name/string()`
2. `//country//name/string()`
3. `//employee[@name="Aaron Aardwark"]/descendant::employee`
4. `//employee[employee]/@id/string()`
5. `//employee[@id="52"]/employee`
6. `//employee[@name="Dolly Dove"]/../employee/@id/string()`

Write your answer here:

1.
```
Denmark
Iceland
```
2.
```
SuperOne Ørestad
SuperOne Dragør
SuperOne Grindsted
SuperOne Kringlan
```
3.
```
<employee id=''43'' name=''Chris Cat'' />
<employee id=''44'' name=''Dolly Dove'' />
```
4.
```
42
52
53
```
5.
```
<employee id=''53'' name=''Frank Fish''>
<employee id=''54'' name=''Garth Gorilla'' />
</employee>
```
6.
```
43
44
44
```

The following is an unfinished XML Schema for an XML language that is supposed to include the XML document above:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:d="http://database.org"
   targetNamespace="http://database.org"
   elementFormDefault="qualified">

<element name="database"> <complexType>
   <sequence>
      <element ref="*** 1 ***" maxOccurs="200"/>
   </sequence>
</complexType> </element>

<element name="country"> <complexType>
   <sequence>
      <element ref="d:location" *** 2 ***/>
   </sequence>
   <attribute name="name" type="string" use="required"/>
</complexType> </element>

<element name="location"> <complexType>
   <sequence>
      <element ref="d:department" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="id" use="required"/>
   *** 3 ***
</complexType> </element>

<element name="department"> <complexType>
   <sequence>
      <element name="name" type="string"/>
      <element name="employee" type="d:employee_with_subordinates"/>
   </sequence>
   <attribute name="id" type="integer"/>
</complexType> </element>

<complexType name="employee_with_subordinates">
   <sequence>
      *** 4 ***
   </sequence>
   <attribute name="id" type="integer"/>
   <attribute name="name" type="string"/>
</complexType>

</schema>
```

b) Suggest ways to fill in the parts marked by `*** 1 ***`, `*** 2 ***`, `*** 3 ***`, and `*** 4 ***`. Your schema should accept the XML document above, and similar documents extracted from the database of Problem 1. You do not need to introduce attributes that are mentioned in Problem 1 but not used in the XML document.

Write your answer here:

1. `d:country`
2. `maxOccurs=''unbounded''`
3. `<attribute name=''population'' type=''integer'' use=''optional''/>` (the default value of `use` is `''optional''`, so this could be omitted).
4. `<element name=''employee'' type=''d:employee_with_subordinates'' minOccurs=''0'' maxOccurs=''unbounded''/>`

c) Write an XQuery expression that finds all employees that work in more than one department. For each such employee, the query should return his/her `id`, and the `ids` of the locations in which he/she works. For example, when run on the XML document above it should return `<employee id=''44''>23 31</employee>`. You may want to use the functions `fn:distinct-values` and `fn:count`. Refer to the document as `doc(``hierarchy.xml'')`.

Write your answer here:

```
let $ids := doc(''hierarchy.xml'')//employee/@id/string()
for $id in fn:distinct-values($ids)
where fn:count($ids[. eq $id])>1
return <employee id=''{$id}''>
{doc(''hierarchy.xml'')//location[.//employee[@id eq $id]]/@id/string()}
</employee>
```

3 SQL (25 %)

We now consider queries on a database using the schema given in problem 1 (without your modifications).

a) Write a query that returns the name of each `Location`, and the name of the country where this location lies (referenced using `countryId`).

WRITE YOUR ANSWER HERE:

```
SELECT C.name, L.name FROM Country C, Location L
WHERE C.id=L.countryId;
```

b) Write a query that returns, for each country, the number of departments in that country with more than 10 employees (`numberOfEmployees)`.

WRITE YOUR ANSWER HERE:

```
SELECT C.name, COUNT(*) FROM Country C, Department D
WHERE C.id=D.country AND numberOfEmployees>10
GROUP BY C.name;
```

c) Write one or more SQL statements that remove from `Department` all departments that did not receive a shipment, as recorded in `Shipment(toDepartment)`. Make sure to remove further information as needed to preserve referential integrity.

WRITE YOUR ANSWER HERE:

It is assumed that references in Shipments to departments that are deleted should be set to NULL.

```
ALTER TABLE ADD CONSTRAINT FOREIGN KEY (fromDepartment) REFERENCES
Department(id) ON DELETE SET NULL;
DELETE FROM Department
WHERE id NOT IN (SELECT toDepartment FROM Shipment)
```

Employees working in a deleted department will automatically be deleted due to `ON DELETE CASCADE` of the foreign key.

We next consider the following queries:

A.
```
SELECT sum(population) FROM Department D, Location L
  WHERE D.location=L.locId AND D.country=L.countryId;
```

B.
```
SELECT sum(DISTINCT population) FROM Department D, Location L
  WHERE D.location=L.locId AND D.country=L.countryId;
```

C.
```
SELECT sum(population) FROM Location WHERE (locId,countryId)
  IN (SELECT location, country FROM Department);
```

d) Indicate for each of the following statements if it is *true* or *false*:

1. A and B always return the same result.
2. B and C always return the same result.
3. One of of A, B, and C has a correlated subquery.
4. One of the queries computes the total population in locations where there is at least one department.
5. One of the queries computes the total population in all countries.

A correct answer suffices to get full points, but you are welcome to add brief explanations of your answers.

Write your answer here:

1. False. A counts the population multiplied by the number of departments.
2. False. They may differ if two locations have the same population.
3. False. The only subquery is not correlated.
4. True. This is query C.
5. False. None of them counts the population in locations with no departments.

e) Write an SQL query that lists all departments where no employee has a birth year before the birth year of the manager.

Write your answer here:

```
SELECT D.id, D.name FROM Department D, Employee E
WHERE D.manager = E.id and D.id = E.worksAt and E.birthyear <=
(SELECT min(birthyear) FROM Employee E2 WHERE E2.worksAt=D.id)
```

or without a correlated subquery:

```
SELECT D.id, D.name
FROM Department D,
Employee E,
(SELECT worksAt,MIN(birthyear) AS minb FROM Employee GROUP BY worksAt) M
WHERE D.manager = E.id and D.id = M.worksAt and E.birthyear <= minb;
```

4 Normalization (10%)

Consider the relation `Employee` from problem 1. The attribute `id` uniquely identifies a person, but is not a key for the relation.

a) Give a brief and convincing argument that `Employee` is not in 3rd normal form (3NF).

WRITE YOUR ANSWER HERE:
Since `id` determines a person, we have the functional dependency `id` $\rightarrow$ `name birthyear`. Since `id` is not a key, this violates 3NF (and even BCNF).

b) Perform a decomposition of `Employee` into 3NF. It suffices to write the attributes of the final relations, with key attributes underlined.

WRITE YOUR ANSWER HERE:
```
Person(id, name, birthYear)
Employment(personId,departmentId,boss)
```

5 Indexing (10 %)

Consider the following queries on the schema given in problem 1:

1. `SELECT id FROM Department WHERE numberOfEmployees > 10;`
2. `SELECT name FROM Department WHERE numberOfEmployees > 10;`
3. `SELECT id FROM Department WHERE country = 45 AND numberOfEmployees > 10;`
4. `SELECT boss FROM Employee WHERE name LIKE 'Ra%';`

5.
   ```
   SELECT * FROM Department D, Employee E
   WHERE D.manager = E.id AND D.id = E.worksAt;
   ```

We assume that the DBMS has *not* automatically created indexes on the primary keys of the relations `Department` and `Employee`. The following are potential secondary indexes:

A. `CREATE INDEX A ON Department(id,name,numberOfEmployees)`

B. `CREATE INDEX B ON Department(numberOfEmployees,country,id)`

C. `CREATE INDEX C ON Employee(id,name)`

D. `CREATE INDEX D ON Employee(birthyear,id,worksAt)`

a) Among all combinations of an index and a query, indicate with a + those that match, i.e., where a DBMS might access the index to answer the given query, and with a - those that do not match. While not needed to get full points, you are welcome to explain your choices.

WRITE YOUR ANSWER HERE:

Query	1.	2.	3.	4.	5.
Index A	+	+	-	-	+
Index B	+	+	+	-	-
Index C	-	-	-	-	+
Index D	-	-	-	-	-

Data Storage and Formats

IT Universitety of Copenhagen

January 5, 2009

This exam is a translation, by Michael Magling, of an original Danish language exam. It consists of 6 problems with a total of 15 questions. The weight of each problem is stated. You have 4 hours to answer all questions. The complete assignment consists of 11 pages (including this page). **It is recommended to read the problems in order**, but it is not important to solve them in order.

If you cannot give a complete answer to a question, try to give a partial answer.

The pages in the answer must be ordered and numbered, and be supplied with name, CPR-number and course code (BDLF). Write only on the front of sheets, and order them so that the problems appear in the correct order.

"KBL" refers to the set in the course book "Database Systems - an application approach, 2nd edition", by Michael Kifer, Arthur Bernstein and Philip M. Lewis.

All written aids are allowed.

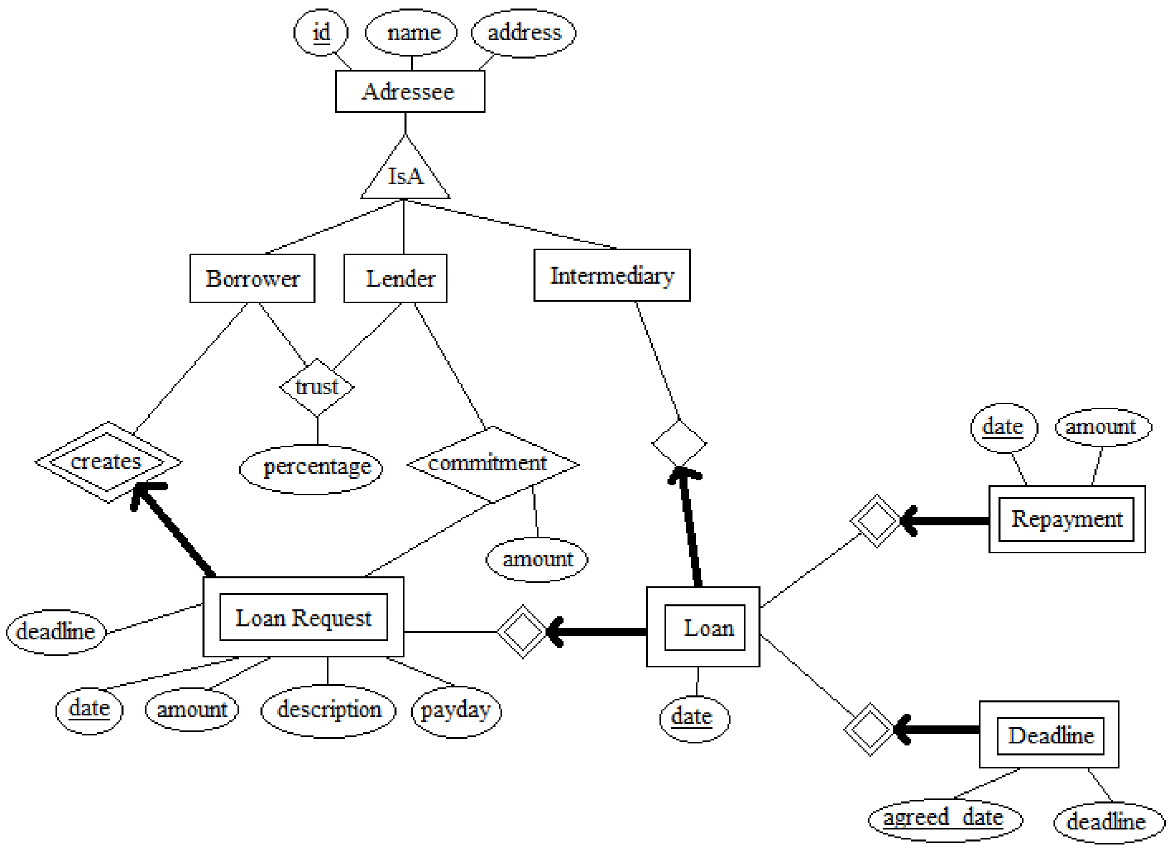
1 Data modeling (25%)

Micro loans are small loans, which is beginning to gain popularity especially among borrowers in developing countries. The idea is to bring venture lenders together using information technology. Typically, the loans will be used to finance startup or development of the borrower's company, so that there is a realistic chance for repayment. The money in a loan can, unlike traditional loans, come from many lenders. In this problem, you must create an E-R model that describes the information necessary to manage micro loans. The following information form the basis for creating the model:

- Each borrower and lender must be registered with information about name and address.
- A loan starts with a loan request, which contains information about when the loan should at latest be granted, The total amount being discussed (US-dollars), and how long the payback period is. Also, a description is included of how the money will be used. The rent on the payment is calculated in the loan amount, which is to say, the full amount is not paid .
- Lenders can commit to an optional portion of the total amount of a loan request.
- When the commitments for the loan request covers the requested amount, the request is converted to a loan. If not enough commitments can be reached, the loan request is cancelled. A borrower can have more than one request, and more than one loan at a time, but can at most make one request per day.
- The loan is paid through an "intermediary", typically a local department of a charity, who has a name and an address.
- The borrower chooses when he or she will make a payment. Every payment must be registered in the database with an amount and a date (at most one payment per loan per day). The lenders share the repayment based on how large a part of the loan they are responsible for.
- If the loan is not repaid before the agreed upon deadline, a new date is agreed. The database must not delete the old deadline, but save the history (the deadline can be overridden multiple times).
- Each lender can for each burrower save a "trust", which is a number between 0 and 100 that determines the lender's evaluation of the risk of lending money to that person. The number must only be saved for the borrowers, for whom there has been made such an evaluation.

a) Make an E-R model for the data described above. If you make any assumptions about data that doesn't show from the problem, they must be described. Use the E-R notation from KBL. Put an emphasis on having the model express as many properties about the data as possible, for instance participation constraints.

Example answer:



b) Make a relational data model for micro loans:

- Describe at least **two** of the relations using SQL DDL (make reasonable assumptions about data types), and
- state the relation schemas for the other relations.

The emphasis is if there is a correlation between the relational model and the E-R diagram from a), along with primary key and foreign key constrations being stated for all relation. It is *not* necessary to state CHECK constraints and the like.

Example answer: It is assumed, that borrowers, lenders, and intermediaries are disjoint entities. Below MySQLs ENUM type is used, which is not part of the syllabus.

```
CREATE TABLE Adressee (
  id INT PRIMARY KEY,
  type ENUM('borrower', 'lender', 'intermediary'),
  name VARCHAR(50),
  address VARCHAR(50)
);

CREATE TABLE Trust (
  borrower INT REFERENCES Adressee(id),
  lender INT REFERENCES Adressee(id),
  percentage INT,
  PRIMARY KEY (borrower,lender)
);

CREATE TABLE LoanRequest (
  id INT REFERENCES Adressee(id),
  date DATE,
  amount INT,
  description VARCHAR(1000),
  payday DATE,
  deadline DATE,
  PRIMARY KEY (id,date)
);

CREATE TABLE Commitment (
  lender INT REFERENCES Adressee(id),
  borrower INT,
  loanrequestDate DATE,
  FOREIGN KEY (borrower,loanrequestDate) REFERENCES LoanRequest(id,dato),
  amount INT,
  PRIMARY KEY (lender, borrower, loanrequestDate,amount)
);

CREATE TABLE Loan (
  id INT,
  RequestDate DATE,
  date DATE,
  intermediary REFERENCES Adressee(id),
  FOREIGN KEY (id,RequestDate) REFERENCES LoanRequest(id,date),
  PRIMARY KEY(date,id,RequestDate)
);

CREATE TABLE Repayment (
  id INT,
  date DATE,
  RequestDate DATE,
  amount INT,
  FOREIGN KEY (id,RequestDate) REFERENCES LoanRequest(id,date),
  PRIMARY KEY (date,id,RequestDate)
);

CREATE TABLE Deadline (
  id INT,
  agreedDate DATE,
  RequestDate DATE,
  deadline DATE,
  FOREIGN KEY (id,RequestDate) REFERENCES LoanRequest(id,date),
  PRIMARY KEY (agreedDate,id,RequestDate)
);
```

2 XML (20%)

Consider the following XML document, `loaners.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="mystylesheet.xsl" type="text/xsl"?>
<microloans>
<loaner>
  <name>
    <first>Nandela</first>
    <last>Melson</last>
  </name>
  <address>Freedom Way 1, 23456 Johannesburg, South Africa</address>
  <loan>
    <amount>1000</amount>
    <payout-date>1990-01-01</payout-date>
    <repayment amount="100" date="1991-01-01"/>
    <repayment amount="100" date="1992-01-01"/>
  </loan>
  <loan>
    <amount>500</amount>
    <payout-date>1993-01-01</payout-date>
    <repayment amount="100" date="1991-01-01"/>
  </loan>
</loaner>
<loaner>
  <name>
    <first>Majeev</first>
    <last>Rotwani</last>
  </name>
  <address>Circle Strait 8, 98764 Bumbai, India</address>
</loaner>
</microloans>
```

a) Write an XPath expression that returns all of the name (`name` elements) in `loaners.xml`. Emphasis is on if the expression also works on other, similar, XML documents.

Example answer:
//name

b) Write an XPath expression that returns all the names of borrowers, who have (had) at least one loan, which is to say, where there is a `loan` element. Emphasis is on if the expression also works on other, similar, XML documents.

Example answer:
//loaner[loan]/name

Consider the following XSL stylesheet, `mystylesheet.xsl`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="microloans">
   <html>
   <body>
     <xsl:apply-templates select="//loan"/>
   </body>
   </html>
</xsl:template>

<xsl:template match="loan">
   <xsl:apply-templates select="../name/last"/>,
   <xsl:apply-templates select="../name/first"/>:
   <xsl:apply-templates select="amount"/><br/>
</xsl:template>
</xsl:stylesheet>
```

c) State the result of running `mystylesheet.xsl` on `loaners.xml`.

Example answer:

```
<html>
<body>
   Melson, Nandela: 1000<br/>
   Melson, Nandela: 500<br/>
</body>
</html>
```

d) Write an XQuery expression that for each borrower in `loaners.xml` computes the total amount, which is to say the sum of the numbers in the `amount` elements, minus the sum of the numbers in the repayment attribute of the `repayment` elements. The output must be valid XML that for each borrower states name and outstanding amount (in a `debt` element).

Example answer:

```
<loaners>
{
for $x in doc("loaners.xml")//loaner
return <loaner>
        {$x/name}
<debt> {fn:sum($x/loan/amount) - fn:sum($x/loan/repayment/@amount)} </debt>
</loaner>
}
</loaners>
```

3 Normalization (10%)

The following relation schema can be used to register information on the repayments on micro loans (see the text in the problem 1 for the explanation on micro loans, and the example on data about micro loans in problem 2).

`Repayment(borrower_id,name,address,loanamount,requestdate,repayment_date,request_amount)`

A borrower is identified with an unique `borrower_id`, and has only one address. Borrowers can have multiple simultaneous loans, but they always have different request dates. The borrower can make multiple repayments on the same day, but not more than one repayment per loan per day.

a) State a key (candidate key) for `Repayment`.

Example answer: `{borrower_id,requestdate,repayment_date}`

b) Make the normalization to BCNF. State for every step in the normalization, which functional dependency that causes it.

Example answer: Functional dependencies:

```
borrower_id → name address
borrower_id requestdato → loanamount
```

BCNF:

```
Repayment1(borrower_id,name,address)
Repayment2(borrower_id,requestdate,loanamount)
Repayment3(borrower_id,requestdate,repayment_date,repayment_amount)
```

4 SQL (25 %)

This problem is about writing SQL for the relation `Repayment` from problem 3:

`Repayment(borrower_id,name,address,loanamount,requestdate,repayment_date,repayment_amount)`

To solve the problem, the information from the description in problem 3 must be used.

a) Write an SQL request that returns all the tuples with information on repayments from the borrower with `id` equal to 42, and where the lent amount exceeds 1000 USD.

Example answer:

```
SELECT *
FROM Repayment
WHERE borrower_id=42 AND loanamount>1000;
```

b) Write an SQL request that for each address finds the total repaid amount for the address.

Example answer:

```
SELECT address, SUM(repayment_amount)
FROM Repayment
GROUP BY address;
```

c) Write an SQL request that finds all names which has a unique address, which to say is where there does not exist a tuple with a different name and same address.

Example answer:

```
SELECT name
FROM Repayment A
WHERE 1=
   (SELECT COUNT(DISTINCT name)
    FROM Repayment B
    WHERE A.address=B.address);
```

d) Write an SQL command, which deletes all information on ended loans, which is to say loans where the total repaid amount equals the lend amount.

Example answer:

```
DELETE FROM Repayment A
WHERE loanamount=
   (SELECT SUM(repayment_amount)
    FROM Repayment B
    WHERE B.borrower_id=A.borrower_id AND B.requestdate=A.requestdate);
```

5 Transactions (10 %)

Consider the following transactions, which uses explicit locking of tuples. Here `?1,?2,?3,?4` is used to reference parameters that are substituted in.

```
1. SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
   SELECT * FROM R WHERE id=?1 FOR UPDATE;
   SELECT * FROM S WHERE pk=?2 FOR UPDATE;
   UPDATE R SET a=?3 WHERE id=?1;
   UPDATE S SET b=?4 WHERE pk=?2;
   COMMIT;

2. SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
   SELECT * FROM S WHERE pk=?1 FOR UPDATE;
   SELECT * FROM R WHERE id=?2 FOR UPDATE;
   UPDATE S SET d=?3 WHERE pk=?1;
   UPDATE R SET c=?4 WHERE id=?2;
   COMMIT;
```

a) Argue that there is a possibility for deadlocks, if the two transactions are run at the same time. State a specific sequence of locks, that leads to a deadlock.

Example answer: If the two transactions both locks the same tuples in `R` of `S` deadlocks are created for instance in the situation where transaction 1 locks `R`, and transaction 2 after that locks `S`.

b) Suggest a change of the transactions, so deadlocks can no longer be created, and give a short argument that this is in fact the case. Emphasis is on that the transactions keep their original effect.

Example answer: By changing the order of the rowlocks in one of the transaction, the problem is avoided (the two transactions locks in the same order, and as asuch, deadlocks cannot be created).

6 Indexing (10%)

We again look at the relation `Repayment` from problem 3 (un-normalized). Assume that the following four SQL commands are known to be frequent (with actual parameters substituted in for `?`):

```
1. SELECT DISTINCT name, address
   FROM Repayment
   WHERE borrower_id = ?;

2. SELECT *
   FROM Repayment
   WHERE borrower_id = ? AND repayment_date > ?;

3. SELECT borrower_id, loanamount
   FROM Repayment
   WHERE loanamount BETWEEN ? AND ?;

4. INSERT INTO Request VALUES (?,?,?,?,?,?,?);
```

a) Suggest one or more indexes, taking into account of the above. State the indexed attributes for each index, along with the index type (primary or secondary). Argue shortly for your choices. Emphasis is on the suggested indexes supports the SQL commands as effectively as possible.

Example answer:

Primary index (B-tree) on `borrower_id, repayment_date` (used with 1 and 2).
Secondary index (B-tree) on `loanamount, borrower_id` (covering index from request 3).

Databasesystemer

IT Universitetet i København

8. juni 2006

Eksamenssættet består af 5 opgaver med 16 spørgsmål, fordelt på 10 sider (inklusiv denne side), *samt et svarark, hvorpå visse spørgsmål skal besvares.*

Vægten af hver opgave er angivet. Du har 4 timer til at besvare alle spørgsmål. Hvis du ikke er i stand til at give et fuldt svar på et spørgsmål, så prøv at give et delvist svar. Du kan vælge at skrive på engelsk eller dansk (evt. med engelske termer).

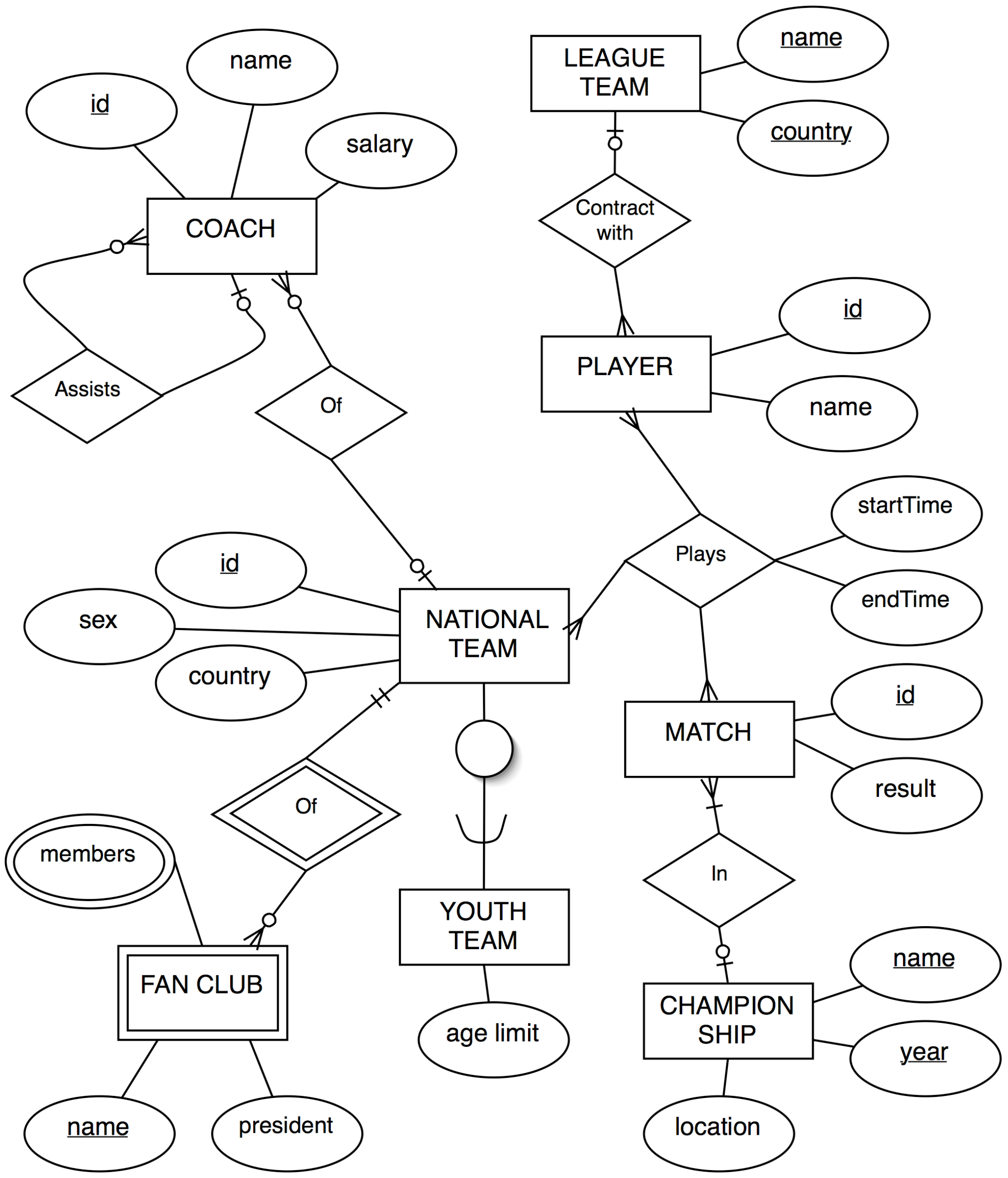
Siderne i besvarelsen skal være nummererede, og forsynet med navn, CPR nummer og kursuskode (DBS). Skriv kun på forsiden af arkene, og sortér dem inden nummereringen, så opgaverne forekommer i nummerrækkefølge.

"MDM" refererer i sættet til kursusbogen "Modern Database Management 7th edition" af Jeffery A. Hoffer, Mary B. Prescott and Fred R. McFadden.

Alle skriftlige hjælpemidler er tilladt.

1 Datamodellering (35%)

Betragt nedenstående EER diagram, der modellerer data om landsholdsfodbold: Trænere (COACH), fanklubber (FAN CLUB), kampe (MATCH), mesterskaber (CHAMPIONSHIP) og spillere (PLAYER). For trænere modelleres data om, hvem der assisterer hvem (Assists). For spillere modelleres data om, hvilket ligahold (LEAGUE TEAM) de har kontrakt med (Contract with). Visse landshold er ungdomslandshold (YOUTH TEAM). For fanklubber modelleres data om medlemmer, og hvem der er formand (president). For hver kamp modelleres data om hvilke spillere, som var med (Plays), og hvornår de blev skiftet ind og ud (startTime og endTime). Hvis hele kampen spilles, er værdierne af disse attributter henholdsvis 0 og 90.



a) Indikér for hvert af følgende udsagn hvorvidt det stemmer overens med EER diagrammet. (Bemærk at diagrammet ikke nødvendigvis er en eksakt modellering af virkeligheden.) Brug svararket til dine svar — sæt præcis ét X i hver søjle.

1. Et landshold har altid mindst 1 træner.
2. En trænerassistent kan selv have en assistent.
3. En spiller har højst kontrakt med 1 ligahold.
4. En spiller kan deltage i kampe for mere end 1 land.
5. En spiller kan blive skiftet ind flere gange i en kamp, og således have flere starttider.
6. Et ungdomshold kan spille med i et mesterskap.
7. Der kan være 20 spillere på banen for hvert hold i en kamp.
8. Der kan findes to fanklubber med samme navn.

Svar:

Spørgsmål 1.a	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Stemmer med EER		**X**	**X**	**X**		**X**	**X**	**X**
Stemmer ikke med EER	**X**				**X**			

b) Konvertér EER diagrammet til relationer. I forbindelse med konvertering af 1-til-mange relationships skal du bruge den metode, der giver det mindste antal relationer. Din besvarelse skal angive skemaerne for de resulterende relationer, med attributterne i primærnøglen understreget.

Svar:

```
COACH(id, name, salary, assists, nationalteamId)
LEAGUETEAM(name,country)
PLAYER(id,name,contractwithName,contractwithCountry)
NATIONALTEAM(id,sex,country)
MATCH(id,result,championshipName,championshipYear)
FANCLUB(name,nationalteamId,president)
YOUTHTEAM(nationalteamId,agelimit)
CHAMPIONSHIP(name,year,location)
Plays(playerId,nationalteamId,matchId,startTime,endTime)
FanClubMembers(name,nationalteamId,member)
```

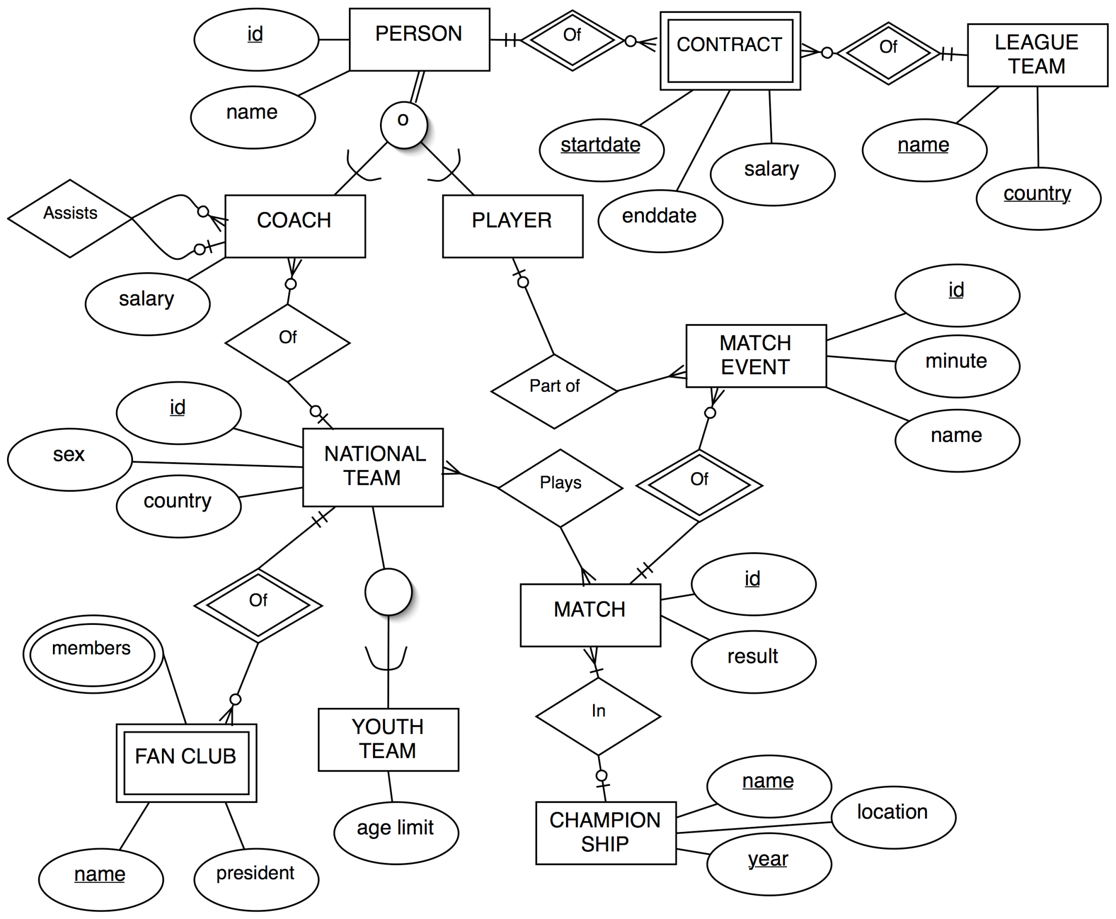
EER diagrammet modellerer ikke historiske data om spilleres karriere (hvilke hold, de har spillet for, i hvilke perioder, og til hvilken løn). Desuden vil en spillende træner svare til en instans af COACH entiteten såvel som PLAYER entiteten, uden nogen information om, at der er tale om samme person. Der ønskes en ny datamodel, hvor disse restriktioner ikke gælder.

Desuden ønskes det, at datamodellen skal gøre det muligt ikke blot at registrere kampens resultat, men også de vigtigste hændelser i løbet af en kamp:

- Scoringer (hvem målscoreren er, og i hvilket minut scoringen er lavet).
- Straffespark (hvilket minut, hvem der begik straffesparket, og mod hvem).
- Advarsler og udvisninger (hvem og hvornår).
- Udskiftninger og indskiftninger – som i det nuværende EER diagram.

c) Lav en revideret EER model, der tager højde for de beskrevne ønsker. De dele af EER diagrammet, der ikke ændres, kan evt. udelades. Der lægges vægt på, at datamodellen let skal kunne tilpasses ønsker til mere detaljeret information. Supplér om nødvendigt dit diagram med forklarende tekst.

Svar:



Forklaring: Alle hændelser er abstraheret til ”match events”, inklusive ind- og udskiftninger af spillere. Et straffespark registreres som to hændelser (at det bliver begået, og at det bliver dømt). Det er valgt at lade en ”contract” være mellem en person og et ligahold, da dette muliggør at gemme information om træneres kontrakter med ligahold (og det ville ikke være en simplere løsning ikke at tillade dette).

2 Normalisering (15%)

Betragt en relation med skemaet: `Varesalg(forhandler,producent,produkt,omsætning)`. Følgende er en gyldig instans af `Varesalg`:

forhandler	**producent**	**produkt**	**omsætning**
Silman	SoftFloor AG	Velour	101000
Bjarnes Tæpper	Bøgetæpper	Berber	207000
Top Tæpper	Bøgetæpper	Kashmir	77000
Silman	SoftFloor AG	Berber	72000
Bjarnes Tæpper	Bøgetæpper	Valnød	17000

a) Hvilke af følgende potentielle FDer gælder *ikke*, baseret på ovenstående instans?

1. `omsætning` → `produkt`
2. `omsætning` → `produkt forhandler`
3. `produkt` → `producent`
4. `producent` → `produkt`
5. `forhandler produkt` → `omsætning`

Hvis instansen ikke siger noget om en FD, markeres den som “måske FD”. Brug svararket til dit svar — sæt præcis ét X i hver søjle..

Svar:

Spørgsmål 2.a	**1**	**2**	**3**	**4**	**5**
Ikke FD			**X**	**X**	
Måske FD	**X**	**X**			**X**

Bemærk, at instansen ovenfor kan fås ved “natural join” af følgende relationer:

forhandler	**producent**
Silman	SoftFloor AB
Bjarnes Tæpper	Bøgetæpper
Top Tæpper	Bøgetæpper

forhandler	**produkt**	**omsætning**
Silman	Velour	101000
Bjarnes Tæpper	Berber	207000
Top Tæpper	Kashmir	77000
Silman	Berber	72000
Bjarnes Tæpper	Valnød	17000

b) Angiv en funktionel afhængighed (FD), der sikrer at `Varesalg` kan opsplittes som i ovenstående eksempel uden tab af information. FDen skal med andre ord sikre, at SQL sætningen

```
(SELECT forhandler, producent FROM Varesalg) NATURAL JOIN
(SELECT forhandler, produkt, omsætning FROM Varesalg)
```

altid returnerer en relation, der er identisk med `Varesalg`. Forklar endvidere i ord, hvad FDen udtrykker.

Svar:

FDen `forhandler` $\rightarrow$ `producent` skal gælde. (FDen `forhandler` $\rightarrow$ `produkt omsætning` rækker også, men gælder ikke ifølge den viste instans.) FDen udtrykker, at enhver forhandler kun har produkter fra een producent.

c) Angiv en instans af `Varesalg`, hvor den viste opsplitning går galt, dvs. hvor SQL sætningen i spørgsmål b) *ikke* returnerer den samme instans.

Svar:

forhandler	**producent**	**produkt**	**omsætning**
Bjarnes Tæpper	Bøgetæpper	Berber	207000
Bjarnes Tæpper	SoftFloor AG	Velour	17000

3 SQL (30 %)

Betragt relationerne `fan(id,navn,cprnr,indmeldelse,favorit)` og `spiller(id,navn,land)`, og instanser med følgende data:

id	**navn**	**cprnr**	**indmeldelse**	**favorit**
1	Birger Hansen	1412861261	2000	5
2	Mads Mikkelsen	2605807413	1995	5
3	Jens Green	0909928475	2005	2
4	Hans Westergaard	1006701245	1980	1
5	Christian Lund	1102524895	1975	2
6	Jesper Andersen	1501661569	2000	3
7	Betina Jørgensen	1506751486	2005	5

id	**navn**	**land**
1	Peter Ijeh	Nigeria
2	Marcus Allbäck	Sverige
3	Martin Bernburg	Danmark
4	Jesper Christiansen	Danmark
5	Michael Gravgaard	Danmark

Relationerne indeholder data om medlemmerne i en fanklub, og deres favoritspillere.

a) Hvor mange tupler returneres der fra hver af følgende forespørgsler, hvis de køres på ovenstående instanser?

1. `SELECT * FROM fan WHERE indmeldelse = 2003;`
2. `SELECT * FROM fan WHERE indmeldelse >= 2000 AND favorit <> 5;`
3. `SELECT COUNT(*), indmeldelse FROM fan GROUP BY indmeldelse;`
4. `SELECT * FROM fan WHERE navn LIKE 'Hans%';`

5.
   ```
   SELECT R1.navn, R2.navn FROM fan R1, fan R2
   WHERE R1.favorit = R2.favorit and R1.id < R2.id;
   ```

6.
   ```
   SELECT navn FROM fan R1
   WHERE (select count(*) FROM fan R2 WHERE R2.favorit=R1.favorit) > 1;
   ```

7.
   ```
   SELECT navn FROM fan WHERE favorit NOT IN
   (SELECT id FROM spiller WHERE land='Danmark');
   ```

Brug svararket til dine svar.

Svar:

Spørgsmål 3.a	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Antal tupler	0	2	5	1	4	5	3

b) Skriv en SQL-kommando der, for alle tupler i relationen `fan` hvor `cprnr` er større end `3112999999` eller mindre end `0101000000`, sætter `cprnr` til værdien `NULL`.

Svar:

UPDATE fan SET cprnr = NULL WHERE cprnr > 3112999999 OR cprnr < 0101000000;

c) Skriv en SQL-kommando der sletter alle tupler i `fan` hvor `cprnr` har værdien `NULL`.

Svar:

DELETE fan WHERE cprnr = NULL;

d) Skriv en SQL forespørgsel der for hver medlem i fanklubben viser medlemmets navn `(fan)` og navn på medlemmets favorit `(spiller)`.

Svar:

SELECT fan.navn,spiller.navn FROM fan,spiller WHERE fan.favorit=spiller.id;

e) Skriv en SQL forespørgel der beregner gennemsnittet af kolonnen `indmeldelse` i relationen `fan`.

Svar:

```
SELECT AVG(indmeldelse) FROM fan;
```

f) Definér i SQL et view der for hver medlem i fanklubben viser medlemmets navn `(fan)` og navn på medlemmets favorit `(spiller)`. Brug dit view til at udregne hvor mange fans de forskellige spillere har (spillerens navn skal fremgå).

Svar:

```
CREATE VIEW MyView AS
SELECT fan.navn AS fannavn, spiller.navn AS spillernavn
FROM fan,spiller
WHERE fan.favorit=spiller.id;
```

(Omdøbningen er nødvendigt i Oracle for at undgå at flere attributer har samme navn. En besvarelse uden omdøbning er også korrekt.) Brug af viewet:

```
SELECT spillernavn,COUNT(*) FROM MyView GROUP BY spillernavn;
```

g) Skriv en SQL forespørgsel der returnerer en relation med *én* attribut indeholdende alle navne i `fan` og `spiller`. Du kan antage, at datatyperne for `navn` attributterne er identiske.

Svar:

```
(SELECT navn FROM spiller) UNION ALL (SELECT navn FROM fan);
```

h) Skriv en SQL forespørgsel der returnerer navnene på alle spillere, der har flere fans blandt kvindelige end blandt mandlige fans. En person i `fan` er mand hvis udtrykkket `cprnr % 2 = 1` er sandt, og kvinde hvis `cprnr % 2 = 0`.

Svar:

```
SELECT navn FROM spiller s WHERE
(SELECT COUNT(*) FROM fan f WHERE f.favorit = s.id AND cprnr % 2 = 1) <
(SELECT COUNT(*) FROM fan f WHERE f.favorit = s.id AND cprnr % 2 = 0);
```

4 Transaktioner (10 %)

Betragt to databaseforbindelser, der laver opdateringer og forespørgsler på relationen `MyFan(id, navn)`:

Forbindelse 1	**Forbindelse 2**
`INSERT INTO MyFan VALUES (3,'Bent Ølgård');`	
	`INSERT INTO MyFan VALUES (7,'Birger Hansen');`
`INSERT INTO MyFan VALUES (5,'Birger Hansen');`	
	`SELECT * FROM MyFan;` (1)
`COMMIT;`	
	`DELETE FROM MyFan;`
`SELECT * FROM MyFan;` (2)	
	`ROLLBACK;`
	`SELECT * FROM MyFan;` (3)

a) Antag at `MyFan` ikke indeholder nogen tupler, at transaktionerne kører på isoleringsniveau `READ COMMITED`, og at de enkelte SQL kommandoer sendes til DBMSen i den rækkefølge, der er vist ovenfor. Hvilke tupler returneres af hver af de 3 `SELECT` statements?

Svar:

(1) (7,'Birger Hansen')
(2) (3,'Bent Ølgård') og (5,'Birger Hansen')
(3) (3,'Bent Ølgård') og (5,'Birger Hansen').

5 Constraints (10%)

Antag at relationerne `fan` og `spiller` er oprettet uden nogen form for constraint og at tabllerne indeholder de data der er vist i opgave 3. Vi tilføjer nu constraints til tabellerne med følgende kommandoer:

- `ALTER TABLE spiller ADD CONSTRAINT MyFirstConstraint PRIMARY KEY (id);`
- `ALTER TABLE fan ADD CONSTRAINT MySecondConstraint`
  `FOREIGN KEY (favorit) REFERENCES spiller(id);`
- `ALTER TABLE fan ADD CONSTRAINT MyThirdConstraint UNIQUE (cprnr);`

a) Angiv for hver af følgende kommandoer hvilke af de tre ovenstående constraints (om nogen)der brydes, dvs. giver anledning til en fejlmeddelelse.

1. `DELETE FROM spiller WHERE land='Sverige';`
2. `INSERT INTO spiller VALUES (6,'Michael Gravgaard','Danmark');`
3. `UPDATE fan SET cprnr=1214650124 where navn LIKE '%Hans%';`
4. `INSERT INTO fan VALUES (7,'Hans Metz',NULL,2001,7);`
5. `UPDATE fan set favorit=NULL where navn LIKE '%e%';`

Brug svararket til dit svar. Skriv plus (+) for at markere en brudt constraint, og minus (-) for at markere en constraint, der ikke brydes. Hvis du er i tvivl om et svar, kan du skrive et spørgsmålstegn.

Svar:

Spørgsmål 5.a	**1**	**2**	**3**	**4**	**5**
MyFirstConstraint	-	-	-	-	-
MySecondConstraint	+	-	-	+	-
MyThirdConstraint	-	-	+	-	-

Svarark (afleveres)

Navn		**Sidenummer**	
CPR		**Totalt sidetal**	

Instruktioner. I spørgsmål 1.a og 2.a skal du sætte præcis ét X i hver søjle. I spørgsmål 3.a skal du skrive 7 heltal. I spørgsmål 5.a skal du markere svarene med + og -. Hvis du er i tvivl kan du angive et spørgsmålstegn.

Bemærk at retningen vil blive gjort på en måde, så tilfældige svar ikke betaler sig. For eksempel vil to korrekte svar og et forkert svar give det samme antal point som ét korrekt svar og to spørgsmålstegn. Det er tilstrækkeligt at give korrekte svar for at få maksimumpoint, men hvis du vælger at forklare dine svar vil dette blive taget i betragtning ved retningen.

Spørgsmål 1.a	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Stemmer med EER								
Stemmer ikke med EER								
?								

Spørgsmål 2.a	**1**	**2**	**3**	**4**	**5**
Ikke FD					
Måske FD					
?					

Spørgsmål 3.a	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Antal tupler							

Spørgsmål 5.a	**1**	**2**	**3**	**4**	**5**
MyFirstConstraint					
MySecondConstraint					
MyThirdConstraint					

Introduction to Databases

IT University of Copenhagen

January 16, 2006

This exam consists of 5 problems with a total of 16 questions. The weight of each problem is stated. You have 4 hours to answer all 16 questions. The complete assignment consists of 12 numbered pages (including this page), *plus an answer sheet to be used for several of the questions.*

If you cannot give a complete answer to a question, try to give a partial answer. You may choose to write your answer in Danish or English. Write only on the front of sheets, and remember to write your CPR-number on each page. Please start your answer to each question at the top of a *new* page. Please order and number the pages before handing in.

GUW refers to *Database Systems – The Complete Book* by Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom, 2002.

All written aids are allowed / Alle skriftlige hjælpemidler er tilladt.

1 Database design (25%)

The academic world is an interesting example of international cooperation and exchange. This problem is concerned with modeling of a database that contains information on researchers, academic institutions, and collaborations among researchers. A researcher can either be employed as a professor or a lab assistant. There are three kinds of professors: Assistant, associate, and full professors. The following should be stored:

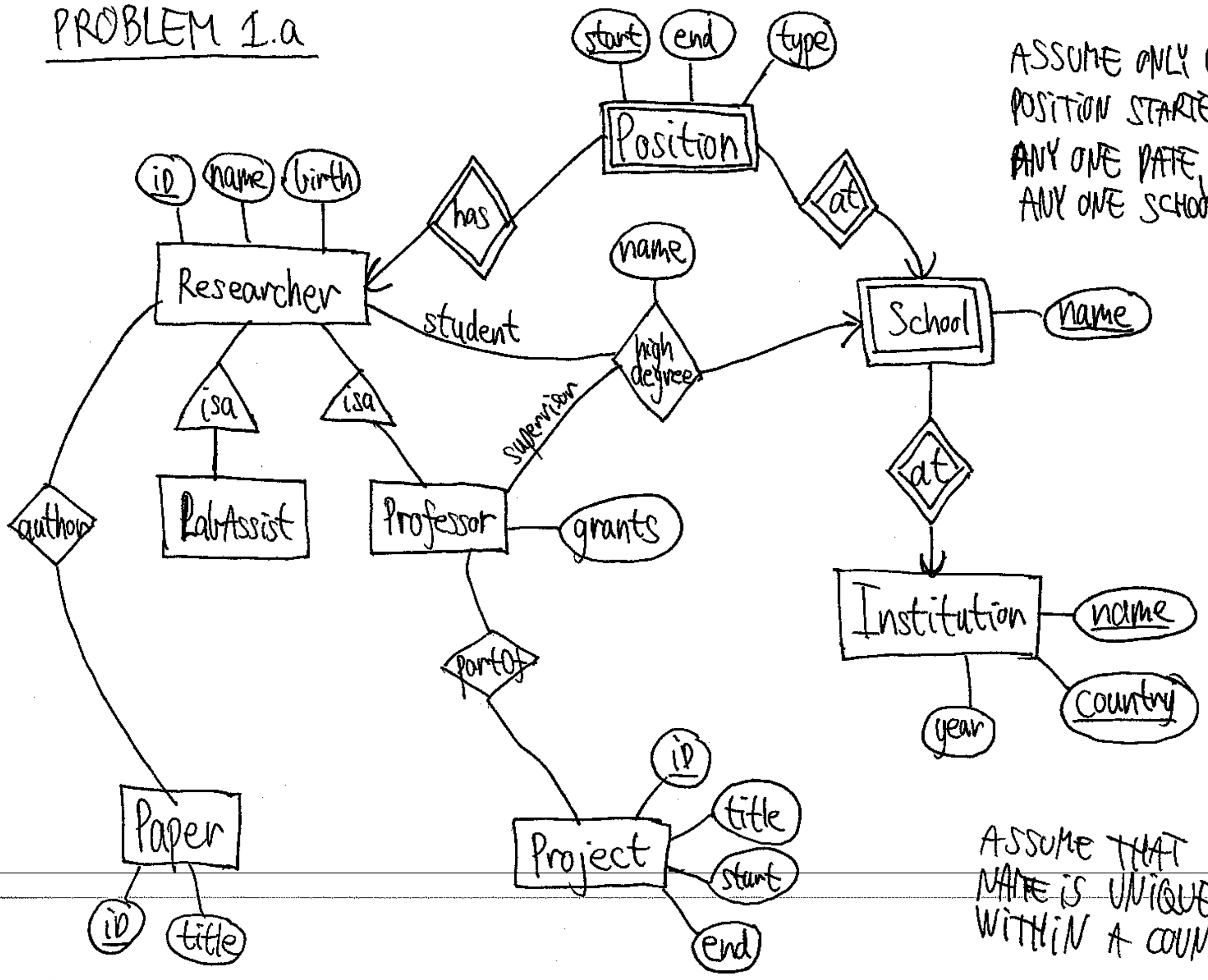
- For each researcher, his/her name, year of birth, and current position (if any).
- For each institution, its name, country, and inauguration year.
- For each institution, the names of its schools (*e.g. School of Law, School of Business, School of Computer Science,. . .*). A school belongs to exactly one institution.
- An employment history, including information on all employments (start and end date, position, and what school).
- Information about co-authorships, i.e., which researchers have co-authered a research paper. The titles of common research papers should also be stored.
- For each researcher, information on his/her highest degree (BSc, MSc or PhD), including who was the main supervisor, and at what school.
- For each professor, information on what research projects (title, start date, and end date) he/she is involved in, and the total amount of grant money for which he/she was the main applicant.

a) Draw an E/R diagram for the data set described above. Make sure to indicate all cardinality constraints specified above. The E/R diagram should not contain redundant entity sets, relationships, or attributes. Also, use relationships whenever appropriate. If you need to make any assumptions, include them in your answer.

Answer:

See next page.

PROBLEM 1.a



ASSUME ONLY ONE POSITION STARTED AT ANY ONE DATE AT ANY ONE SCHOOL

ASSUME THAT NAME IS UNIQUE WITHIN A COUNTRY

b) Convert your E/R diagram from question a) into relations, and write SQL statements to create the relations. You may make any reasonable choice of data types. Remember to include any constraints that follow from the description of the data set or your E/R diagram, including primary key and foreign key constraints.

Answer:

```
CREATE TABLE Researcher (
ID int PRIMARY KEY,
name VARCHAR(50),
birth INT
);

CREATE TABLE Professor (
ID int REFERENCES Researcher(ID),
grants INT
);

CREATE TABLE Paper (
ID int PRIMARY KEY,
title VARCHAR(50)
);

CREATE TABLE Position (
researcherID int REFERENCES Researcher(ID),
start DATE,
end DATE,
schoolName VARCHAR(50),
institutionName VARCHAR(50),
FOREIGN KEY (schoolName,institutionName) REFERENCES School(name,insitutionName),
PRIMARY KEY (researcherID,start,schoolName,institutionName)
);

CREATE TABLE Project (
ID int PRIMARY KEY,
title VARCHAR(50),
start DATE,
end DATE
);

CREATE TABLE School (
name VARCHAR(50),
instititionName VARCHAR(50) REFERENCES Institution(name),
PRIMARY KEY (name, instititionName)
);
```

```
CREATE TABLE Institution (
name VARCHAR(50),
country VARCHAR(50),
year INT,
PRIMARY KEY (name, country)
);

CREATE TABLE Author (
researcherID INT REFERENCES Researcher(ID),
paperID INT REFERENCES Paper(ID),
PRIMARY KEY (researcherID, paperID)
);

CREATE TABLE PartOf (
professorID INT REFERENCES Professor(ID),
studentID INT REFERENCES Researcher(ID),
PRIMARY KEY (professorID, projectID)
);

CREATE TABLE HighDegree (
professorID INT REFERENCES Professor(ID),
projectID INT REFERENCES Project(ID),
schoolName VARCHAR(50),
institutionName VARCHAR(50),
FOREIGN KEY (schoolName,institutionName) REFERENCES School(name,insitutionName),
PRIMARY KEY (professorID, studentID, schoolName, institutionName)
);
```

2 Normalization (15%)

We consider the following relation:

`Articles(ID,title,journal,issue,year,startpage,endpage,TR-ID)`

It contains information on articles published in scientific journals. Each article has a unique ID, a title, and information on where to find it (name of journal, what issue, and on which pages). Also, if results of an article previously appeared in a "technical report" (TR), the ID of this technical report can be specified. We have the following information on the attributes:

- For each journal, an issue with a given number is published in a single year.
- The `endpage` of an article is never smaller than the `startpage`.
- There is never (part of) more than one article on a single page.

The following is an instance of the relation:

ID	title	journal	issue	year	startpage	endpage	TR-ID
42	Cuckoo Hashing	JAlg	51	2004	121	133	87
33	Deterministic Dictionaries	JAlg	41	2001	69	85	62
33	Deterministic Dictionaries	JAlg	41	2001	69	85	56
39	Dictionaries in less space	SICOMP	31	2001	111	133	47
57	P vs NP resolved	JACM	51	2008	1	3	99
77	What Gödel missed	SICOMP	51	2008	1	5	98
78	What Gödel missed	Nature	2222	2008	22	22	98

a) Based on the above, indicate for each of the following sets of attributes whether it is a key for `Articles` or not. Use the answer sheet of the exam for your answer.
1. {`ID`}; 2. {`ID,TR-ID`}; 3. {`ID,title,TR-ID`}
4. {`title`}; 5. {`title,year`}; 6. {`startpage,journal,issue`}
If you wish, you may additionally write a brief explanation for each answer, which will be taken into account, but is not necessary to get full points.

Answer:

#	1	2	3	4	5	6
Key		X				
Not a key	X		X	X	X	X

b) Based on the above, indicate for each of the following potential functional dependencies, whether it is indeed an FD or not. Use the answer sheet of the exam for your answer.
1. `ID` → `title`; 2. `startpage` → `endpage`; 3. `journal issue` → `year`
4. `title` → `ID`; 5. `ID` → `startpage endpage journal issue`; 6. `TR-ID` → `ID`
If you wish, you may additionally write a brief explanation for each answer, which will be taken into account, but is not necessary to get full points.

Answer:

#	**1**	**2**	**3**	**4**	**5**	**6**
FD	**X**		**X**		**X**	
Not an FD		**X**		**X**		**X**

c) Based on a) and b), perform normalization into BCNF, and state the resulting relations.

Answer:

All FDs above are avoidable (by question a), so we decompose according to:

```
journal issue → year
ID → title journal issue startpage endpage
```

This gives the following relations:

```
Articles1(journal,issue,year)
Articles2(ID,title,journal,issue,startpage,endpage)
Articles3(ID,TR-ID)
```

3 SQL and relational algebra (35%)

We consider again the relation `Articles` from problem 2.

a) Indicate for each of the following expressions whether it is a valid SQL statement or not. A valid statement, as described in GUW, should be accepted by a standard SQL interpreter, whereas an invalid statement should result in an error message. Use the answer sheet of the exam for your answer.

1. `SELECT * FROM Articles WHERE endpage-startpage>10;`
2. `SELECT * FROM Articles WHERE endpage-startpage<0;`
3. `SELECT SUM(title) FROM Articles;`
4. `SELECT AVG(year) FROM Articles WHERE title LIKE 'C%';`
5. `SELECT COUNT(*) FROM Articles GROUP BY year;`
6. `SELECT year,COUNT(*) FROM Articles WHERE COUNT(*)>10 GROUP BY year;`

Answer:

#	**1**	**2**	**3**	**4**	**5**	**6**
Valid	**X**	**X**		**X**	**X**	
Invalid			**X**			**X**

b) Indicate for each of the following queries, how many tuples would be returned if it was run on the instance of `Articles` from problem 2. Use the answer sheet of the exam for your answer.

1. `SELECT ID FROM Articles WHERE year<2006;`
2. `SELECT DISTINCT ID FROM Articles WHERE year<2006;`
3. `SELECT AVG(year) FROM Articles GROUP BY journal;`
4. `SELECT ID FROM Articles WHERE title LIKE '%d';`

Answer:

#	**1**	**2**	**3**	**4**
Number of tuples	4	3	5	3

Consider the relations `Authors(auID,name)` and `Authoring(articleID,authorID)`, containing information on names of authors, and who is authoring which papers, respectively.

c) Write an SQL query that returns for each article, its ID, title and the number of authors.

Answer:

```
SELECT ID, title, COUNT(*)
FROM Articles, Authoring
WHERE ID=articleID
GROUP BY ID,title;
```

d) Write an SQL query that returns the titles of articles authored by '`Robert Tarjan`'.

Answer:

```
SELECT title
FROM Articles, Authors, Authoring
WHERE ID=articleID AND auID=authorID AND name='Robert Tarjan';
```

e) Write an SQL query that returns the number of co-authors of `'Robert Tarjan'`. (I.e., the number of authors who have written at least one article together with him.)

Answer:

```
SELECT COUNT(DISTINCT A2.authorID)
FROM Authors, Authoring A1, Authoring A2
WHERE A1.authorID=auID AND name='Robert Tarjan' AND
     A2.authorID<>auID AND A1.articleID=A2.articleID;
```

f) Write SQL statements that correspond to the following two relational algebra expressions. Duplicate elimination should be performed.

1. $\pi_{\texttt{title,year}}(\sigma_{\text{year}=2005}(\texttt{Articles}))$
2. $\gamma_{\texttt{year,COUNT(ID)}}(\texttt{Articles})$

Answer:

```
1.  SELECT DISTINCT title, authorID FROM Articles WHERE
year=2005;
2.  SELECT year,COUNT(ID) FROM Articles GROUP BY year;
```

4 Efficiency and transactions (15%)

Consider the following six queries on `Articles` from problem 2:

```
1.  SELECT title FROM Articles WHERE year=2005;
2.  SELECT title FROM Articles WHERE endpage=100;
3.  SELECT title FROM Articles WHERE year>1995 AND year<2000;
4.  SELECT title FROM Articles WHERE journal='JACM' AND issue=55;
5.  SELECT title FROM Articles WHERE issue=55 AND journal='JACM';
6.  SELECT title FROM Articles WHERE endpage-startpage>50;
```

a) Indicate which of the above queries would likely be faster (based on the knowledge you have from the course), if *all* of the following indexes were created. Use the answer sheet of the exam for your answer.

```
    CREATE INDEX Idx1 ON Articles(year,startpage);
    CREATE INDEX Idx2 ON Articles(startpage,endpage);
    CREATE INDEX Idx3 ON Articles(journal,issue,year);
```

Answer:

#	**1**	**2**	**3**	**4**	**5**	**6**
Faster	**X**		**X**	**X**	**X**	
Same		**X**				**X**

In the following we consider the below transactions on the `Authors(auID,name)` relation.

Time	User A	User B
1	`INSERT INTO Authors VALUES (42,'Donald Knuth');`	
2		`INSERT INTO Authors VALUES (43,'Guy Threepwood');`
3		`DELETE FROM Authors WHERE name LIKE 'Don%';`
4		`INSERT INTO Authors VALUES (44,'Donald E. Knuth');`
5	`DELETE FROM Authors WHERE name LIKE 'Guy%';`	
6	`COMMIT;`	
7		`COMMIT;`

b) Suppose that `Authors` is initially empty, that the transactions are run at isolation level `READ COMMITTED`, and that the commands are issued in the order indicated above. What is the content of `Authors` after the execution?

Answer:

auID	name
42	Donald Knuth
43	Guy Threepwood
44	Donald E. Knuth

c) Suppose that `Authors` is initially empty. What are the possible contents of `Authors` after each *serial* execution of the two transactions?

Answer:

auID	name
43	Guy Threepwood
44	Donald E. Knuth

auID	name
44	Donald E. Knuth
42	Donald Knuth

5 Constraints (10%)

Suppose that the `Authoring` relation of problem 3 relation was created as follows:

```
CREATE TABLE Authoring(
  articleID INT REFERENCES Article(ID) ON DELETE SET NULL,
  authorID INT REFERENCES Author(ID) ON DELETE CASCADE
)
```

a) Indicate which of the following statements are true, and which are not. Use the answer sheet of the exam for your answer.

1. If we try to delete a tuple from `Authoring`, the tuple is not deleted. Instead, `articleID` is set to `NULL`.
2. If we delete a tuple from `Authoring`, any tuples in `Author` referred to by this tuple are also deleted.
3. If we delete a tuple from `Article`, some attributes of `Authoring` may have their values set to `NULL`.
4. If we try to insert a tuple into `Author`, with an `ID` that is not referred to in `Authoring`, the operation is rejected.
5. If we try to insert a tuple into `Authoring`, with an `ID` that does not exist in `Author`, the operation is rejected.

Answer:

#	**1**	**2**	**3**	**4**	**5**
True			**X**		**X**
False	**X**	**X**		**X**	

b) Write `CHECK` constraints for `Articles` of Problem 2 that ensure the following:

1. Values of the `journal` attribute does *not* start with '`Journal`'.
2. The value of the `endpage` attribute is never smaller than that of `startpage`.
3. The value of `year` is given in full (e.g. 1999 is not abbreviated as 99). You may assume that `year` is of type integer, and that there are no articles more than 200 years old.

Answer:

```
1.  CHECK NOT (journal LIKE 'Journal%');
2.  CHECK endpage>=startpage;
3.  CHECK year>=1000;
```

Answer sheet (to be handed in)

Name		**Page number**	
CPR		**Total pages**	

Instructions. For all questions except 3.b (which asks for numbers), you must place exactly one X in each column. Note that the grading will be done in a way such that random answering does not pay. For example, two correct answers and one incorrect answer will be worth the same as one correct answer and two question marks.

Question 2.a	**1**	**2**	**3**	**4**	**5**	**6**
Key						
Not a key						
?						

Question 2.b	**1**	**2**	**3**	**4**	**5**	**6**
FD						
Not an FD						
?						

Question 3.a	**1**	**2**	**3**	**4**	**5**	**6**
Valid						
Invalid						
?						

Question 3.b	**1**	**2**	**3**	**4**
Number of tuples				

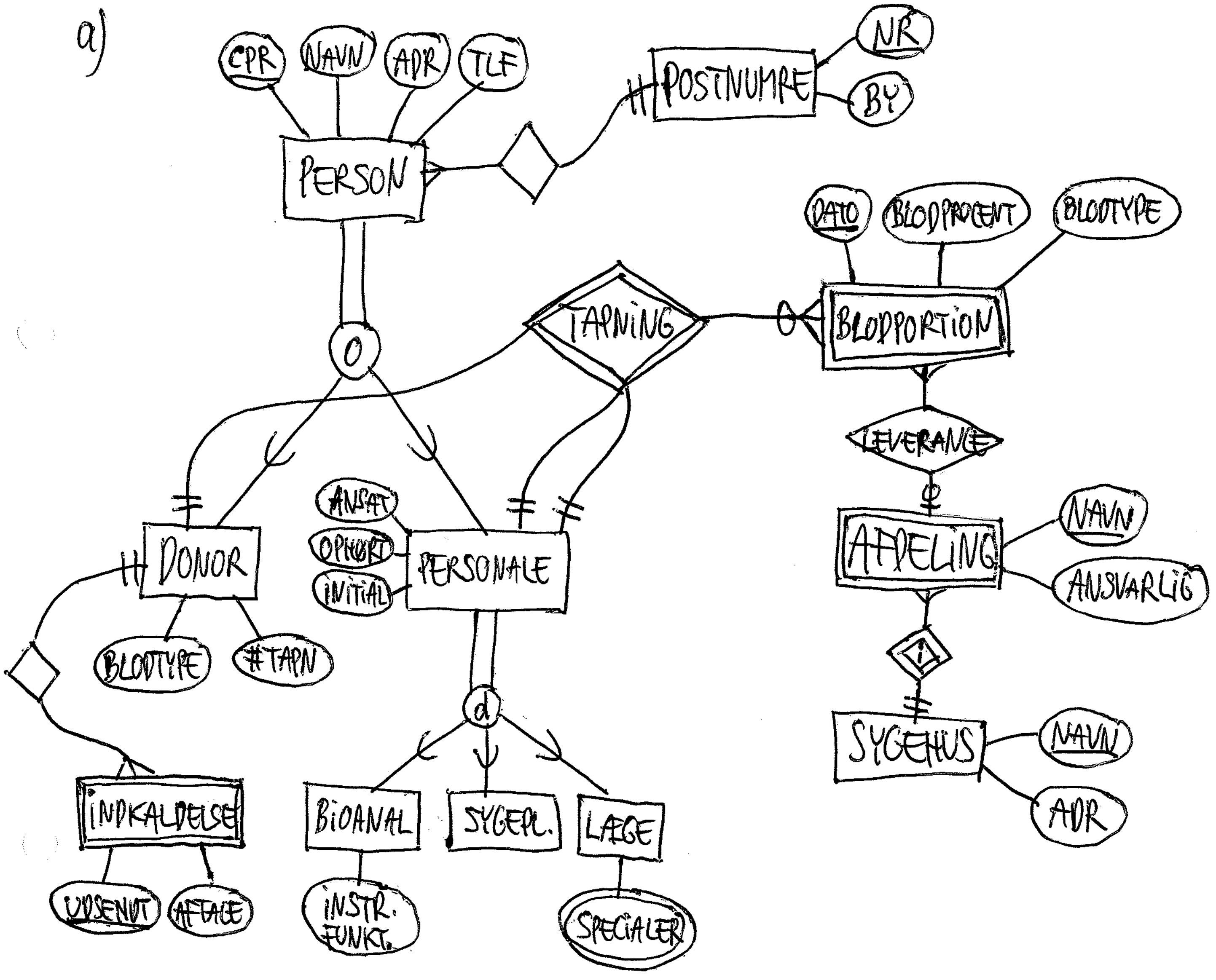
Question 4.a	**1**	**2**	**3**	**4**	**5**	**6**
Faster						
Same						
?						

Question 5.a	**1**	**2**	**3**	**4**	**5**
True					
False					
?					

DATABASESYSTEMER 7/6 - 2005

OPGAVE 1

a)



b) VED BRUG AF METODE FRA FORELÆSNINGSSLIDES:

PERSON(CPR, NAVN, ADR, TLF, POSTNR)

POSTNUMRE(NR, BY)

DONOR(CPR, BLODTYPE, #TAPN)

PERSONALE(CPR, ANSAT, OPHØRT, INITIAL)

INDKALDELSE(CPR, UDSENDT, AFTALE)

BIOANAL(CPR, INSTRFUNKT.)

LÆGE(CPR, SPECIALE)

KAN EVT. INDEHOLDE "SUBTYPE DISCRIMINATOR"

BLODPORTION(DONORCPR, PERSONALECPR1, PERSCPR2, DATO, BLODPROCENT, BLODTYPE, LEVERETTIL)

AFDELING(NAVN, SYGEHUS, ANSVARLIG)

SYGEHUS(NAVN, ADR)

1

OPGAVE 2

a) KANDIDATNØGLER: {id, sygdom}, {sygdom, by}, {sygdom, postnr}.

b) FLG. FUNKTIONELLE AFHÆNGIGHEDER HAR IKKE EN KANDIDATNØGLE PÅ VENSTE SIDE:

id → adresse, postnr, by
postnr → by
sygdom → speciale

DA postnr OG by BEGGE ER EN ÆGTE DELMÆNGDE AF EN KANDIDATNØGLE, GIVER KUN FD'ERNE id→adresse OG sygdom→ speiale ANLEDNING TIL DEKOMPOSITION:

Adr(id, adresse)
Spec(sygdom, speciale)
Behandler2(id, postnr, by, sygdom)

OPGAVE 3

a) 1 RETURNERER ALLE BEHANDLERE AF HUNDEGALSKAB i POSTNR 1000 (iFØLGE OPG.2 ~~ER~~ DER HØJST ÉN).
SPECIALET SOM HUNDEGALSKAB HØRER UNDER FOR

2 RETURNERER ALLE SPECIALER SOM DEN PÅGÆLDENDE BEHANDLER HAR.

b)
```
SELECT speciale, COUNT(DISTINCT id), COUNT(DISTINCT sygdom)
FROM Behandler
GROUP BY speciale;
```

c)
```
SELECT id
FROM Patient
WHERE sygdom NOT IN (SELECT sygdom
                     FROM Behandler
                     WHERE id=behandler_id);
```

d)
```
CREATE TABLE Patient2(
    id INT, sygdom CHAR(32), behandler_by CHAR(16),
    FOREIGN KEY (sygdom,behandler_by) REFERENCES Behandler(sygdom,by));

UPDATE Patient
 SET behandler_id=NULL
 WHERE sygdom NOT IN (SELECT sygdom FROM Behandler
                      WHERE id=behandler_id);
```
ÆNDRING ~~AF~~ PATIENT KAN UNDGÅS VED FØRST AT LAVE EN KOPI

```
INSERT INTO Patient2 VALUES
    (SELECT Patient.id, Patient.sygdom, Behandler.by
     FROM Patient, Behandler
     WHERE behandler_id = Behandler.id);
```

OPGAVE 4

a) PÅ ISOLERINGSNIVEAU READ COMMITTED KAN ÆNDRINGER LAVET AF ANDRE TRANSAKTIONER FOREKOMME MELLEM TO SQL SÆTNINGER.

TRANSAKTIONEN FINDER FRIE SÆDER I SÆTNING 3, OG ÉT AF DEM KAN SAGTENS BLIVE BOOKET INDEN SÆTNING 7.

b) PÅ "REPEATABLE READ" OG "SERIALIZABLE" KAN INGEN LÆST VÆRDI ÆNDRE SIG UNDER TRANSAKTIONEN. DERFOR KAN Seats IKKE OPDATERES MELLEM SÆTNING 3 OG 7.

VED "SNAPSHOT ISOLATION" VIL TRANSAKTIONEN ABORTE HVIS DATA, DEN HAR SKREVET TIL, ER BLEVET ÆNDRET UNDERVEJS AF EN ANDEN TRANSAKTION.

DOBBELTBOOKING ER ALTSÅ UMULIG I ALLE 3 TILFÆLDE.

OPGAVE 5

a) INDEKS PÅ speciale HJÆLPER IKKE, DA speciale IKKE ER DEL AF NOGEN BETINGELSE.

INDEKS PÅ id HJÆLPER IKKE – id FOREKOMMER KUN i FORBINDELSE MED "in" i BETINGELSEN. (EN SMART DBMS KUNNE MÅSKE UDNYTTE ET INDEKS PÅ id, HVIS SUBQUERYEN i 2 HAR ET LILLE RESULTAT).

INDEKS PÅ postnr ELLER sygdom ER EN GOD IDÉ, DA BEGGE BLIVER BRUGT i HØJT SELEKTIVE SELECTS i 1 SÅVEL SOM 2.

b)

idx1	GØR BÅDE 1 OG 2 HURTIGERE. (SÆRLIGT 1)
idx2	GØR 1 OG SUBQUERYEN i 2 HURTIGERE (DVS. 2 LIDT HURTIGERE)
idx3	HJÆLPER IKKE PÅ 1, OG FORMENTLIG IKKE PÅ 2 (EN SMART DBMS KUNNE BRUGE idx3 VED 2 HVIS SUBQUERY HAR LILLE RES.)
idx4	HJÆLPER BÅDE i 1 OG 2

c)

FØRSTE INDEKS PÅ {sygdom, postnr} (idx 2)

ANDET INDEKS PÅ {postnr}

GØR 1 MEGET HURTIG. OG KAN BRUGES AF SUBQUERY i 2.

SELEKTIV ATTRIBUT i 2

(EN SMART DBMS KUNNE UDNYTTE {postnr, id} HVIS SUBQUERYEN GIVER ET LILLE RESULTAT).