# Collection of database exams

Rasmus Pagh

October 25, 2011

This collection of database exams is meant to provide students of the course *Introduction to Database Design* extensive practice towards achieving the intended learning outcomes of the course relating to databases. The first exam covers all intended learning outcomes, including those related to XML. Several E-R notations are used that differ from the one used in our textbook, by Kifer, Bernstein and Lewis (KBL).

- In the fall 2003 and January 2004 exams the relationship R2 means that for each entity of type BRAVO there is at most one entity of type CHARLIE.
- In the january 2005 exam, arrows in relationships are placed next to the entity types, and not the relationships. However, the meaning of an arrow in each direction is the same as in KBL.
- In the June 2006 exam the notation has been changed to coincide with that in KBL.

In the January 2007 exam, the syntax for data accesses is different from that of KBL, but straightforward: R(A) denotes a read of database element A, and so on. In the January 2006 exam, $\Gamma$ is the grouping operator of extended relational algebra, and not described in KBL.

# 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 7 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.

**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.

## 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.

**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.

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`.

**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).

## 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`.

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

## 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.

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

**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.

## 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.

**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.

## 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.

# Database Systems

IT University of Copenhagen

January 2, 2007

This exam consists of 5 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 6 numbered pages (including this page).

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.

RG refers to *Database Management Systems* by Raghu Ramakrishnan and Johannes Gehrke, McGraw-Hill, 2002.

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

## 1 SQL DDL and normalization (20%)

The national railroad company in the Republic of Delalaya maintains a relational database of information on (physical) trains, train personnel, and manning. The database stores key numbers for each train, and information on which employees work on a given train departure. To simplify administrative procedures, employees always work on the same physical train, and always work all the way from the departure station to the final destination. The database has the following schema:

`Train(`<u>`Tid`</u>`,type,productionYear,capacity)`
`Personnel(`<u>`Pid`</u>`,worksOnTrain,worksAs,hiredDate,salary)`
`Manning(Tid,`<u>`Pid`</u>`,`<u>`Sid`</u>`,`<u>`onDate`</u>`)`

The attribute `type` of `Train` refers to the train manufacturer's code for a specific kind of trains. All trains with the same code are identical. The attribute `worksOnTrain` of `Personnel` is a foreign key reference to `Train(Tid)`. This reflects the fact that each person always works on the same train (but of course not all the time). Any tuple in `Personnel` must contain a valid reference to a tuple in `Train`. The `Sid` attribute of `Manning` is a reference to a code for a specific departure in the time table. However, the time table is not part of the database. The attribute pair `(Pid,Tid)` in `Manning` is a foreign key reference to `Personnel(Pid,worksOnTrain)`.

**a)** Write SQL statements that create the above relations, including key and foreign key constraints. You can make any reasonable assumption about the data types.

**b)** Identify all functional dependencies in the relations that do not have a superkey on the left hand side (i.e., are "avoidable"). Use the dependencies to decompose the schema into BCNF, and state the resulting database schema. Briefly discuss the quality of the new schema (ignoring efficiency issues).

## 2 Data modeling (30%)

The transportation authority in the Republic of Delalaya has decided to implement a database to keep statistics on public transportation (fuel-driven buses and electricity-driven trains), with emphasis on keeping track of delays and the number of travelers. In particular, it should be used to identify particular weaknesses in the transportation systems (e.g., stretches of railroad that often cause delays, or employees who have trouble keeping the schedule).

First of all, data from the railroad company, as described in Problem 1, should be integrated into the system, but not necessarily using the schema stated there.

**a)** Draw an ER diagram corresponding to the data described in Problem 1, including, if possible, all integrity constraints stated there. You should follow general rules for making a good ER design — your design does *not* need to translate to the three relations stated in Problem 1.

In addition to the above, the following information is needed:

- **Information on buses.** Similar to the information on trains, but in addition the *range* of each bus (the number of kilometers it will drive on a full tank) should be recorded.
- **Information on bus drivers.** Similar to to the information on train personnel, with the following changes: All bus drivers have the same work (driving the bus). A bus driver does not drive a specific bus, but may drive any bus.
- **Route information.** The sequence of stops on each train/bus route. Each route has a unique *route number*.
- **Vehicle usage.** For each route and departure time, on each day, record which physical train/bus was used. (In Delalaya, this is always a single bus or train – several trains can't be coupled together.)
- **Timetable information.** Information on *planned* arrival and departure times for each route and every stop.
- **Timetable statistics.** Information on *actual* arrival and departure times for every train/bus and every stop, on every day.
- **Traveler statistics.** Periodically, surveys are being made that record the destinations of all travelers in a given bus/train at a given time (between two stops).
- **Manning.** Who has worked on a particular vehicle at every time. It should be taken into account that manning may change at any time during a route.

**b)** Draw an ER diagram that integrates the additional data described above with the ER diagram of Problem 2 a). You should follow general rules for making a good ER design. If you need to make any assumptions, state them.

**c)** Suppose that we desire the database to evolve over time (e.g. with new time tables), but we also want to be able to store and query historical data. Outline how the ER diagram of Problem 2 b) could be changed to achieve this.

## 3 SQL (25 %)

Consider a database of flight departures and airplanes, with the following schema:

```
Departure(departureID,airplaneID,destination,departureTime,bookedSeats)
Airplane(airplaneID,modelID,fabricationYear)
Model(ModelID,name,capacity)
```

**a)** Write an SQL query that lists the `airplaneID` of all airplanes made before 1960.

**b)** Write an SQL query that lists the `departureID` for all departures bounded for destinations starting with the letter "D".

**c)** Write an SQL query that lists the average `capacity` of airplanes fabricated 1970 or later.

**d)** Write an SQL query that lists the `departureID` for every overbooked departure (i.e. where the number of bookings exceed the capacity of the plane).

**e)** Write a query in **relational algebra** that lists the model-`name` of every airplane that was fabricated in 1970.

**f)** Write an SQL query that lists the `fabricationYear` of the oldest and second oldest plane. You may assume that the two oldest planes have different values on the attribute `fabricationYear`.
(**Note:** Some versions of SQL have a special syntax for this kind of query – however, you are required to write the query using only SQL features found in RG.)

**g)** Write an SQL query that lists all `destinations` that has more empty seats than the average number of empty seat on all departures (we assume that the number of empty seats is the number of booked seats subtracted from the capacity of the plane).

## 4 Transactions (10 %)

**a)** Consider an initially empty table `T` with the schema `T(number)` and transactions running at isolation level `READ COMMITTED`

| | Transaction 1 | Transaction 2 |
|---|---|---|
| 1 | | INSERT INTO t VALUES (1) |
| 2 | INSERT INTO t VALUES (2) | |
| 3 | | SELECT * FROM T |
| 4 | SELECT * FROM T | |
| 5 | | ROLLBACK |
| 6 | SELECT * FROM T | |
| 7 | | SELECT * FROM T |

State what is returned from each of the SELECT queries at line 3, 4, 6, and 7. If there are several possibilities, you may state any of them.

**b)** Consider the two schedules below.

**Schedule 1:**

| Transaction 1 | Transaction 2 |
|---|---|
| R(A) | |
| | R(A) |
| R(B) | |
| | W(B) |
| rollback | |
| | W(A) |
| | commit |

**Schedule 2:**

| Transaction 1 | Transaction 2 |
|---|---|
| | R(B) |
| R(B) | |
| R(A) | |
| W(A) | |
| commit | |
| | W(B) |
| | commit |

State for each of the two schedules whether it is serializable or not. If the schedule is serializable write a serialization of the schedule, otherwise give a brief explanation of why the schedule is not serializable.

## 5 Database efficiency (15%)

Consider the relation `T(id,name)` with 100,000 tuples. Values of `id` are positive integers, and values of `name` are strings of length at most 30. The following queries are of interest:

1. `SELECT * FROM t WHERE id = 100`
2. `SELECT * FROM t WHERE id > 10`
3. `SELECT * FROM T WHERE id > 100 and id < 9000 and name = 'Mads'`

**a)** State for each of the above queries whether an index would speed it up. In the cases where the answer is "yes" you also have to specify:

- Would a Hash-index or a B-tree index be the fastest index?
- Which attribute(s) should be indexed?
- Would a clustered index make the query significantly faster than an unclustered index?

# Database Systems

IT University of Copenhagen

June 8, 2006

This exam consists of 5 problems with 16 questions, printed on 7 numbered pages. The weight of each problem is stated. You have 4 hours to answer all questions. If you are unable to answer a question, try to give a partial answer. You may choos to write in English or Danish.

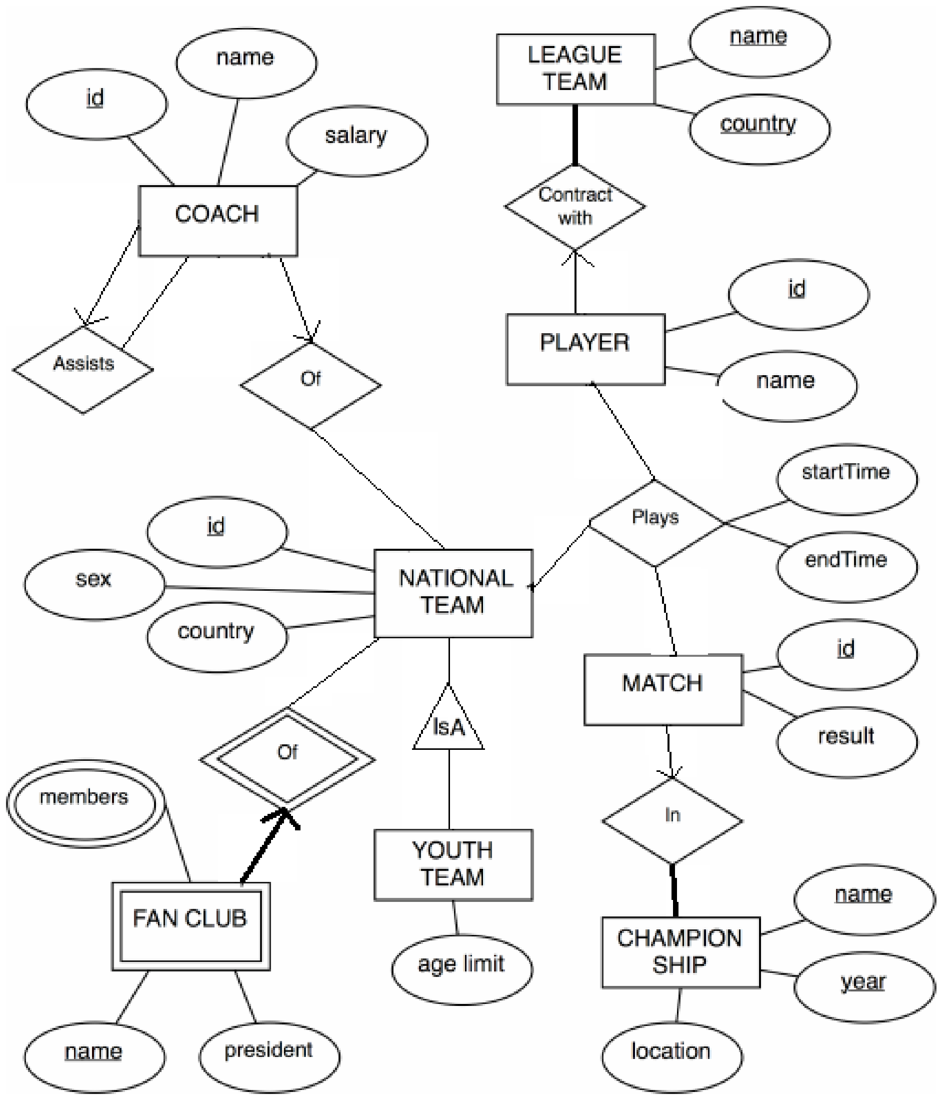
Pages in your answer should be numbered and contain name, CPR number, and course code (DBS). Write only on the front page of pages, and put the pages in order before handing in.

"MDM" refers to the course book "Modern Database Management 7th edition" by Jeffery A. Hoffer, Mary B. Prescott, and Fred R. McFadden.

All written aids are allowed.

## 1 Data modeling (35%)

Consider the below EER diagram (extended Chen notation), modeling data on national team soccer: Coaches, fan clubs, matches, championships, and players. For coaches, it is recorded who is assisting (Assists). For players it is modeled with national league team (LEAGUE TEAM) they have a contract with (Contract with). Some national teams are youth teams – the YOUTH TEAM and NATIONAL TEAM entity sets are connected by an "IsA" relationship. For fan clubs it models who are members, and who is president. For each game it models which players were active (Plays), and in what time period (between startTime and endTime). If the whole match is played, these numbers are 0 and 90, respectively



**a)** Indicate for each of the following statements if they agree with the EER diagram. (Observe that the diagram does not necessarily model reality exactly.)

1. A national team always has at least 1 coach.
2. The assistant of a coach can have an assitant herself.
3. A player has a contract with at most 1 league team.
4. A player cna take part in matches for more than 1 country.
5. A player can be substituted in and out several times in a match, and hence have several starting times.
6. A youth team can take part of a championship.
7. There can be 20 players on court for each team in a match.
8. There can be two fan clubs of the same name.

**b)** Convert the EER diagram to relations. When there are several choices, you should choose a method that minimizes the number of relations. Write the schemas of the resulting relations, with primary key attributes underlined.

The EER diagram does not model historic data on player careers (what teams they have played for, in what periods, and for what salary). Further, a playing coach will correspond to an instance of the COACH entity as well as a PLAYER entity, with no information that this is the same person. A new data model is sought where these restrictions do not apply.

Further, the new data model should make it possible to register not only the result of the match, but also the most important events in a match:

- Goals (who is goal scorer is, and in what minute the goal was scored).
- Penalties (what minute, who committed the penalty, and against whom).
- Red and yellow cards (who and when).
- Substitutions – as in the present ER diagram.

**c)** Draw a revised ER model in your chosen notation, taking the above wishes into account. You should strive to make a flexible data model, which can easily be extended with more detailed information. Write explanatory text if needed to understand your reasoning.

## 2 Normalization (15%)

Consider a relation with the schema: `Sales(seller,producer,product,amount)`. The following is a legal instance of `Sales`:

| seller | producer | product | amount |
|---|---|---|---|
| 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)** Which of the following potential FDs do *not* hold, based on the instance above?
1. `amount` → `product`  2. `amount` → `product seller`
3. `product` → `producer`  4. `producer` → `product`
5. `seller product` → `amount`

The instance above can be computed as the join of these relations:

| seller | producer |
|---|---|
| Silman | SoftFloor AB |
| Bjarnes Tæpper | Bøgetæpper |
| Top Tæpper | Bøgetæpper |

| seller | product | amount |
|---|---|---|
| Silman | Velour | 101000 |
| Bjarnes Tæpper | Berber | 207000 |
| Top Tæpper | Kashmir | 77000 |
| Silman | Berber | 72000 |
| Bjarnes Tæpper | Valnød | 17000 |

**b)** State a functional dependency (FD) that ensure that `Sales` can be split as in the example given with no loss of information. In other words, the FD should ensure that the SQL statement

```
(SELECT seller, producer FROM Sales) NATURAL JOIN
(SELECT seller, product, amount FROM Sales)
```

always returns a relation that is identical with `Sales`. Further, give an explanation in words of what the FD expresses.

**c)** Give an instance of `Sales` where the chosen split does not work, i.e., where the SQL statement in question b) does *not* return the same instance.

## 3 SQL (30 %)

Consider the relations `fan(id,name,cprnr,memberSince,favorite)` og `player(id,name,country)`, and instance with the following data:

| **id** | **name** | **cprnr** | **memberSince** | **favorite** |
|---|---|---|---|---|
| 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** | **name** | **country** |
|---|---|---|
| 1 | Peter Ijeh | Nigeria |
| 2 | Marcus Allbäck | Sverige |
| 3 | Martin Bernburg | Danmark |
| 4 | Jesper Christiansen | Danmark |
| 5 | Michael Gravgaard | Danmark |

The relations contain data on members in a fan club, and their favorite players.

**a)** How many tuples are returned for each of the following queries, when run on the instances above?

1. `SELECT * FROM fan WHERE memberSince = 2003;`

2. `SELECT * FROM fan WHERE memberSince >= 2000 AND favorite <> 5;`

3. `SELECT COUNT(*), memberSince FROM fan GROUP BY memberSince;`

4. `SELECT * FROM fan WHERE name LIKE 'Hans%';`

5.
   ```
   SELECT R1.name, R2.name FROM fan R1, fan R2
   WHERE R1.favorite = R2.favorite and R1.id < R2.id;
   ```

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

7.
   ```
   SELECT name FROM fan WHERE favorite NOT IN
   (SELECT id FROM player WHERE country='Danmark');
   ```

**b)** Write an SQL command that, for each tuple in the relation `fan` where `cprnr` is larger than `3112999999` or less than `0101000000`, sets `cprnr` to the value `NULL`.

**c)** Write an SQL command that deletes all tuples in `fan` where `cprnr` has the value `NULL`.

**d)** Write a query that, for each member in the fan club, computes the name of the member and the name of his/her favorite player.

**e)** Write an SQL query that computes the average of the column `memberSince` in the relation `fan`.

**f)** Define an SQL view that for each member in the fan club shows the name of the member, and the name of the members' favorite player. Use your view to write a query that computes the number of fans of each player (the name of the player must be shown).

**g)** Write an SQL query that returns a relation with a *single* attribute, containing all names in `fan` and `player`. You can assume that the data types for the `name` attributes are identical.

**h)** Write an SQL query that returns the names of all players that have more female than male fans. A person in `fan` is male if and only if the expression `cprnr % 2 = 1` is true.

## 4 Transactions (10 %)

Consider two database connections that make updates and queries on the relation `MyFan(id, name)`:

| **Connection 1** | **Connection 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)** Assume that `MyFan` does not contain any tuples, that the transactions are running at isolation level `READ COMMITED`, and that the individual SQL commands are sent to the DBMS in the sequence shown above. Which tuples are returned for each of the 3 `SELECT` statements?

## 5 Constraints (10%)

Assume that the relations `fan` and `player` have been created with no constraints, and that the tables contain the data shown in problem 3. We now add constraints with the following commands:

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

**a)** State for each of the following commands which of the three above constraints (if any) are violated, i.e., result in an error message.

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%';`

# 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.

**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.

## 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.

**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` $\rightarrow$ `title`; 2. `startpage` $\rightarrow$ `endpage`; 3. `journal issue` $\rightarrow$ `year`
4. `title` $\rightarrow$ `ID`; 5. `ID` $\rightarrow$ `startpage endpage journal issue`; 6. `TR-ID` $\rightarrow$ `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.

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

## 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;`

**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';`

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.

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

**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. $\gamma_{\texttt{year,COUNT(ID)}}(\texttt{Articles})$

## 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);
```

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?

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

## 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.

**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 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

IT Universitetet i København

7. juni 2005

Eksamenssættet består af 5 opgaver med 13 spørgsmål, fordelt på 6 sider (inklusiv denne side).

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 (30%)

**a)** Udarbejd en datamodel for blodbanken i Sengeløse Sygehus ved brug af EER notationen beskrevet i MDM kapitel 3 og 4. Nedenfor er en beskrivelse af det, som systemet skal dække. I det omfang beskrivelsen ikke er fyldestgørende, må du selv gøre dig nogle antagelser. (Sådanne antagelser skal fremgå af besvarelsen.) Ved vurderingen lægges der vægt på, at der er mindst mulig redundans i databasen, og at attributter med samme indhold og betydning ikke unødigt bruges på flere entitetstyper.

**b)** Konvertér din EER model fra spørgsmål a) til en relationel datamodel. Du skal blot angive attributnavne for hver relation, og f.eks. *ikke* specificere datatyper, fremmednøgler, etc. Angiv hvilken metode, du anvender til konverteringen (fra MDM eller fra kursets forelæsningsslides).

**Casebeskrivelse:**
Sengeløse Sygehus er et mindre lokalt sygehus, der har skadestue, tilbyder ambulante behandlinger, og desuden driver en blodbank, der tapper lokale donorer og er leverandør af blodportioner til regionens større sygehuse. Administrationen af blodbanken er hidtil foregået ved hjælp af et papirbaseret journalsystem. I forbindelse med pensioneringen af blodbankens mangeårige sekretær, er det besluttet at erstatte systemet med et moderne IT system, der desuden skal have en række nye funktioner.

For hver donor findes et donoroplysningskort med personinformation samt information om tapninger. Flg. er et eksempel på et sådant donorkort:

**DONOROPLYSNINGSKORT**

**CPR:** 060275-1133
**Navn:** Rasmus Berg
**Adresse:** Bakketoppen 1
**Postnr:** 1313 Bjergby
**Tlf.:** 1223 3344
**Blodtype:** A pos.

**Tapninger (dato, tapning/kontrol):**

| | | |
|---|---|---|
| 1. | 11/2-2001 | ABP/AG |
| 2. | 13/5-2001 | PR/AG |
| 3. | 29/9-2001 | AG/ABP |
| . . . | | |

Oplysningerne om tapning og kontrol handler om, hvem fra personalet, der har foretaget selve tapningen, og hvem der har lavet den obligatoriske kontrol af oplysningerne. Det er tanken, at den information om tapninger, der findes i det eksisterende journalsystem *ikke* skal overføres. Dog ønskes det, at information om, hvor mange gange en donor er tappet i alt, gemmes i det nye system. Ud over disse oplysninger skal det for hver donor registreres, hvornår denne senest er blevet indkaldt til tapning (dette sker ved brev), og hvornår det evt. er aftalt, at tapningen skal foretages.

For personalet skal gemmes standardoplysninger om CPR nummer, navn, adresse, og telefon. Desuden skal det registreres, hvornår en person er ansat (og evt. ophørt med ansættelsen), hvilken uddannelse vedkommende har, samt hvad personens unikke initialer er. (Det er disse initialer, der anvendes på donoroplysningskortene.) Det er *ikke* nødvendigt at gemme oplysninger om tidligere ansættelser, hvis personen har været ansat flere gange. For bioanalytikere skal der yderligere gemmes information om, hvorvidt de har uddannelsesfunktion eller ej. For læger skal der yderligere gemmes information om, hvilke specialer de har (f.eks. genetik eller mikrobiologi). For sygeplejersker skal der ikke gemmes yderligere information.

Den sidste type af information, der skal gemmes i systemet, omhandler blodportionerne. For hver blodportion skal det registreres, hvilken tapning portionen stammer fra, hvornår tapningen fandt sted, hvilken blodtype det drejer sig om, og hvad blodprocenten er. Når en blodportion sendes til en afdeling på et andet sygehus, skal leverancen registreres, så man efterfølgende kan se hvor portionerne blev sendt hen, og hvornår det skete. Til brug ved leverancerne skal systemet registrere navn og adresse for alle sygehuse, der aftager blod fra Sengeløse Blodbank, samt navn på den person på hver afdeling, der er ansvarlig for modtagelse af blod.

## 2 Normalisering (10%)

I denne opgave betragter vi relationen `Behandler`, der indeholder information om medicinske behandlere, og hvilke sygdomme de behandler:

```
Behandler(id,adresse,postnr,by,speciale,sygdom)
```

Hver behandler har en unik `id`, og desuden præcis én registreret adresse. Til ethvert postnummer svarer netop ét bynavn. En behandler kan have mere end ét speciale, men i hver by er der højst én behandler med et givet speciale. En sygdom hører til præcis ét speciale, men et speciale kan dække mange sygdomme.

**a)** Angiv alle kandidatnøgler i `Behandler`. Redegør for eventuelle antagelser om, hvordan data kan se ud.

**b)** Foretag normalisering af `Behandler` til 3. normalform. Angiv det resulterende relationsskema, samt hvilke funktionelle afhængigheder, der er anvendt ved normaliseringen. Eventuelle antagelser om, hvordan data kan se ud, skal også angives.

## 3 SQL (30 %)

Denne opgave omhandler SQL forespørgsler på `Behandler` relationen fra opgave 2. (Bemærk at forespørgslerne skal være på `Behandler` og altså *ikke* på de normaliserede relationer fra spørgsmål 2.b.) Ved løsningen skal du bruge oplysningerne fra opgave 2 om indholdet i `Behandler`. Betragt flg. SQL forespørgsler:

```
1:     SELECT speciale
       FROM Behandler
       WHERE postnr=1000 AND sygdom='hundegalskab';

2:     SELECT speciale
       FROM Behandler
       WHERE postnr=1000 AND id IN (SELECT id
                                    FROM Behandler
                                    WHERE sygdom='hundegalskab');
```

**a)** Giv en kort forklaring i ord på, hvad de to forespørgsler returnerer – herunder hvad der er forskellen mellem dem.

I kurset har vi stiftet bekendtskab med SQLs `COUNT(*)` funktion, der bruges til at tælle tupler. SQL har desuden funktionen `COUNT (DISTINCT <attribut>)`, der bruges til at tælle antallet af *forskellige* værdier af en attribut.

**b)** Skriv en SQL forespørgsel, der returnerer en relation med tre attributter: Der skal være et tupel for hvert speciale i `Behandler` med angivelse af antallet af behandlere og antallet af forskellige sygdomme indenfor specialet.

Antag at vi nu yderligere har relationen `Patient(id,sygdom,behandler_id)`, der indeholder information om patienter, hvilke(n) sygdom(me) de har, og for hver sygdom hvilken behandler (fra `Behandler` relationen), de er henvist til.

**c)** Skriv en SQL forespørgsel der returnerer `id` for de patienter, der (fejlagtigt) er henvist til en behandler, som ikke ifølge `Behandler` kan behandle deres sygdom.

For at forebygge fejlagtige referencer ønskes det at erstatte `Patient` med relationen `Patient2(id,sygdom,behandler_by)`. Idéen er, at {`sygdom,behandler_by`} bruges som fremmednøgle. (Da der kun er én behandler for hver sygdom i hver by, kan man referere til en unik behandler på denne måde.)

**d)** Skriv en SQL kommando, der opretter relationen `Patient2`, og erklærer ovennævnte fremmednøgle. Skriv herefter en eller flere SQL kommandoer, der indsætter informationen fra `Patient` i `Patient2`, således at alle patienter beholder deres behandler(e). Patienter og sygdomme, der ikke er henvist til en kvalificeret behandler, indsættes med værdien `NULL` på attributten `behandler_by`.

## 4 Transaktioner (15 %)

Relationen `Seats(seatID,class,reserved)` bruges til at håndtere sædereservationer i et fly. Den indeholder er tupel for hvert sæde, og attributten `reserved` er 0 eller 1 afhængigt af, om sædet er ledigt eller reserveret. 15 minutter før afgang lukkes der for reservationer til business class ved at eventuelle ledige sæder på business class bliver overført til kunder på "economy plus". Nedenstående transaktion (skrevet i Oracle SQL) foretager overførslen, ved hjælp af to relationer `FreeBusinessSeats(seatID,reserved)` og `Upgrades(seatID)`, der bruges til at gemme mellemresultater.

```
1. DELETE FROM FreeBusinessSeats;
2. DELETE FROM Upgrades;
3. INSERT INTO FreeBusinessSeats (SELECT seatID, reserved FROM Seats
                                   WHERE class='business' AND reserved=0);
4. INSERT INTO Upgrades (SELECT *
                         FROM (SELECT seatID FROM Seats
                               WHERE class='economy plus' AND reserved=1)
                         WHERE rownum<=(SELECT COUNT(*) FROM FreeBusinessSeats));
5. UPDATE FreeBusinessSeats SET reserved=1
   WHERE rownum<=(SELECT COUNT(*) FROM Upgrades);
6. UPDATE Seats SET reserved=0 WHERE seatID IN (SELECT * FROM Upgrades);
7. UPDATE Seats SET reserved=1
   WHERE (seatID,1) IN (SELECT seatID,reserved FROM FreebusinessSeats);
```

**Forklaring:** De først to SQL sætninger sletter evt. gamle mellemresultater. Linie 3 indsætter de ledige business class sæder i relationen `FreeBusinessSeats`. I linie 4 udvælges reservationer på "economy plus", som skal opgraderes. Antallet af opgraderinger holdes under antallet af ledige pladser ved brug af `rownum` variablen, der returnerer nummeret på den aktuelle række. Linie 5 markerer det rette antal ledige sæder i `FreeBusinessSeats` som reserverede. I linie 6 og 7 overføres informationen om de nye reservationer til `Seats` relationen.

**a)** Antag at ovenstående transaktion kører på SQL isoleringsniveau `READ COMMITTED`. Argumentér for, at hvis der samtidig foretages en reservation på en business class plads (dvs. en transaktion, der ændrer en `reserved` værdi fra 0 til 1), kan der forekomme en dobbeltbooking, dvs. at antallet af reserverede sæder er mindre end antallet af passagerer.

På grund af ovenstående problem er det oplagt at overveje et højere isoleringsniveau. Vi betragter SQLs `REPEATABLE READ` og `SERIALIZABLE`, samt "snapshot isolation" beskrevet i artiklen *A Critique of ANSI SQL Isolation Levels.*

**b)** Overvej for hver af de tre ovennævnte isoleringsniveauer, hvorvidt der kan forekomme en dobbeltbooking. Argumentér for dine svar.

## 5 Indeksering (15 %)

De to forespørgsler angivet i opgave 3 refererer til attributterne `speciale`, `postnr`, `sygdom` og `id`. For at forbedre forespørgslernes køretid kan det overvejes at oprette et eller flere indekser. Denne opgave går ud på at vurdere forskellige muligheder for indeksering.

**a)** Vurdér for hver af de fire ovennævnte attributter effekten af at have et indeks på denne attribut som *eneste* indeks. Det skal angives hvilke af de to forespørsgler (om nogen), indekset vil gøre hurtigere. Hvad en den eller de bedste af disse muligheder, med henblik på at gøre begge forespørgsler hurtige? Argumentér for dit svar.

**b)** Betragt følgende fire muligheder for at lave et "composite" indeks:

- `CREATE INDEX idx1 ON Behandler(postnr,sygdom);`
- `CREATE INDEX idx2 ON Behandler(sygdom,postnr);`
- `CREATE INDEX idx3 ON Behandler(id,postnr);`
- `CREATE INDEX idx4 ON Behandler(postnr,id);`

Angiv hvilke af de to forespørgsler (om nogen), hvert indeks vil gøre hurtigere, hvis det oprettes som eneste indeks.

**c)** Foreslå en kombination af to indekser, der får forespørgslerne til at køre så hurtigt som muligt, samlet set. Argumentér for dit valg.

# Introduction to Databases

IT University of Copenhagen

January 7, 2005

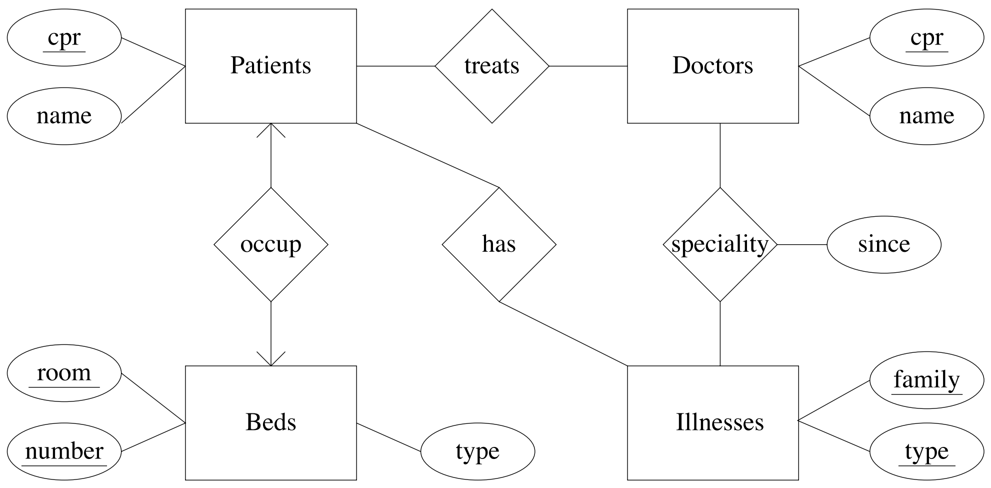
This exam consists of 6 problems with a total of 16 questions. The weight of each problem is stated. You have 4 hours to answer all 16 questions. If you cannot give a complete answer to a question, try to give a partial answer. You may choose to write your answer in Danish or English. Your answers should appear in the same order as the corresponding questions. Remember to write the page number and your CPR-number on each page of your written answer. The complete assignment consists of 5 numbered pages (including this page).

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 (20%)

Consider the following E/R diagram, modeling data about patients in a hospital:



**a)** Perform a conversion of the E/R diagram into relation schemas, using the method described in GUW. You should eliminate relations that are not necessary (e.g., by combining relations).

Imagine you are given the task of integrating the database for the data of the E/R diagram with a database used for managing beds. It has the following relational database schema:

```
Beds(room_id,bed_number,type,buy_date)
Rooms(room_id,type,capacity)
BedBookings(room_id,bed_number,from_date,to_date,patient_cpr)
```

The `Beds` relation contains information on all beds in the hospital, including which room they are in. It corresponds to the "Beds" entity set, plus one additional attribute. The `Rooms` relation contains information about each room. Finally, the `BedBookings` relation contains information about what beds will be used in future, planned hospitalizations. It contains information on at most one hospitalization for each patient.

**b)** Integrate the relational database schema in the E/R diagram above, and draw the complete E/R diagram. The resulting E/R diagram should have the property that it can be converted into relations using the method described in GUW, so that the relations `Beds`, `Rooms`, and `BedBookings` have exactly the above schemas.

## 2 Normalization (15%)

Consider the relations `Beds`, `Rooms`, and `BedBookings` from Problem 1. After questioning the facilities management of the hospital, you have the following information about what kind of data can occur in these relations:

- The only key of `Beds` is `(room_id,bed_number)`.
- All beds bought on the same date are of the same type.
- All beds in the same room were bought on the same date.
- The only key of `Rooms` is `(room_id)`.
- Rooms of the same type can have different capacities.
- There can be rooms of different types with the same capacity.
- The only key of `BedBookings` is `(room_id,bed_number,patient_cpr)`.
- The attribute `to_date` is either `NULL` or larger than `from_date`.

**a)** Based on the above information, give an example of redundancy and an example of an update anomaly in the relations.

**b)** Identify all avoidable functional dependencies in the three relation schemas (i.e., nontrivial functional dependencies that do not have a superkey on the left hand side).

**c)** Based on the functional dependencies from b), perform a decomposition of the relations into BCNF.

## 3 Transactions (10%)

Relation `BedBookings` from Problem 1 has three types of transactions performed on it:

1. Booking a bed. This involves finding a bed in `Beds` that only occurs in `BedBookings` with `to_date` before a certain date, and using it for a new tuple in `BedBookings`.
2. Signing out a patient, i.e., setting the `to_date` attribute to a certain date.
3. Gathering statistics, i.e., computing an aggregate on the relation.

**a)** Suggest an appropriate SQL isolation level for each type of transaction. Argue in favour of your choices.

## 4 SQL and relational algebra (30%)

Consider the relations `Beds`, `Rooms`, and `BedBookings` from Problem 1.

**a)** Write an SQL query that computes the total capacity of all rooms where `type` is `'T'`.

**b)** Write a query in SQL that computes the `room_id`s and `type`s of rooms with at least one bed having `buy_date < '1990'`. (If desired, you may use the fact from Problem 2 that all beds in the same room were bought on the same date.)

Consider the following SQL expression:

```
UPDATE BedBookings
SET room_id=NULL, bed_number=NULL
WHERE (room_id,bed_number) IN (SELECT room_id,bed_number
                               FROM Beds
                               WHERE type='OLD');
```

**c)** Give a short and precise explanation of what changes are performed on the data when the above expression is run.

**d)** Write an SQL statement that sets the capacity of every room in `Rooms` to the number of beds that are currently in the room (as registered in the `Beds` relation).

**e)** Write relational algebra expressions corresponding to the SQL of questions a) and b).

## 5 OLAP (10%)

This problem concerns the construction of a relational OLAP system for data about traffic. Data for the system comes from a sensor that registers passing vehicles, and measures their speed and their type (“bike”, “car”, “van”, or “truck”). The sensor data is combined with information about the time and day of week, and the weather (“snowy”, “rainy”, or “dry”).

**a)** Identify the facts, measures and dimensions to be used in an OLAP system for the traffic data.

**b)** Give a star schema for the data.

## 6 SQL privileges (15%)

Consider the relation `BedBookings` from Problem 1. Suppose that it is created by the user `dba`, who executes the following statements:

```
GRANT SELECT ON BedBookings TO adm WITH GRANT OPTION;
GRANT UPDATE ON BedBookings TO adm WITH GRANT OPTION;
GRANT DELETE ON BedBookings TO adm;
```

Subsequently, the user `adm` executes these statements (some of which may result in error messages from the DBMS):

```
GRANT SELECT ON BedBookings TO doc;
GRANT UPDATE(from_date,to_date) ON BedBookings TO doc WITH GRANT OPTION;
GRANT DELETE ON BedBookings TO doc;
```

**a)** State what kinds of rights (SELECT, UPDATE, DELETE) the user `doc` has on the relation `BedBookings`.

Now assume that the user `dba` executes the following statements (some of which may result in error messages from the DBMS):

```
REVOKE SELECT ON BedBookings FROM adm CASCADE;
REVOKE UPDATE(from_date) ON BedBookings FROM adm CASCADE;
```

**b)** State the rights of the user `doc` after the above REVOKE statements.

The following SQL query returns all tuples in `BedBookings` concerning female patients, omitting the `patient_cpr` attribute. (It uses the fact that females have even CPR numbers.)

```
SELECT room_id,bed_number,from_date,to_date
FROM BedBookings
WHERE (patient_cpr%2=0);
```

**c)** Write SQL statements that, if executed by the user `dba`, allows the user `public` to retrieve the information produced by the above query, but does *not* allow `public` to access any CPR numbers, or any tuples concerning males. **Hint:** First define a view.

# Introduction to Databases

IT University of Copenhagen

January 20, 2004

This exam consists of 6 problems with a total of 12 questions. The weight of each problem is stated. You have 4 hours to answer all 12 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. Remember to write the page number, your name, and your CPR-number on each page of your written answer. The complete assignment consists of 5 numbered pages (including this page).

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%)

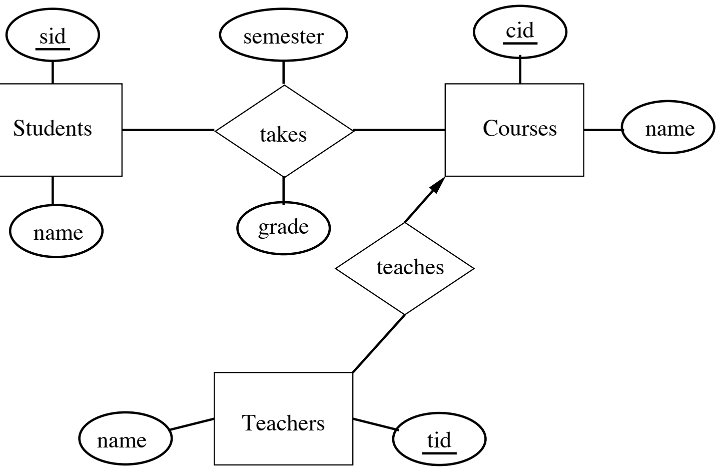
This problem has two unrelated parts. In the first part we consider the task of designing a database for characters in the *Lord of the Rings* books. The Tolkien trilogy contains characters belonging to many different people, e.g., hobbits, elves, dwarves, and men. The database should record:

- The name of each character, and the people he/she belongs to (you may assume that there is exactly one such people for each character).
- The places in the books (volume and page number) featuring this character. (The page number would refer to a specific edition.)
- For every pair of characters, the places in the book where these two characters meet.
- For each people, information on what place(s) it resides in (or has resided in), including the start (and possibly the end) of the period of residence.

For example, it should be recorded that the elves resided in Rivendell since the Second Age, and that Bilbo Baggins and Frodo Baggins are hobbits, and meet on page 3, say, of volume 1.

**a)** Draw an E/R diagram for the database. Remember to include appropriate keys and constraints. Emphasis should be put on using the design principles described in GUW.

In the second part we consider the following E/R diagram:



**b)** Perform a conversion of the E/R diagram into relation schemas, using the method described in GUW. You should combine relations when possible. Write SQL statements to create the relations, including key constraints. The attributes *sid, cid, tid*, and *grade* should be of type integer, and the remaining attributes text strings of length at most 30.

## 2 SQL queries and relational algebra (25%)

Consider the following relation schemas, used for examples in GUW:

```
Movie(title,year,length,inColor,studioName,producerC#)
StarsIn(movietitle,movieyear,starname)
MovieStar(name,address,gender,birthdate)
```

We assume (as in GUW) that the title and year uniquely identify a movie, and that the name uniquely identifies a movie star. Consider the following SQL queries, `Q1` and `Q2`:

```
Q1:  SELECT DISTINCT title, studioName
     FROM Movie, StarsIn
     WHERE starname='Meryl Streep' AND
           title=movietitle AND
           year=movieyear;

Q2:  SELECT DISTINCT title, studioName
     FROM Movie, StarsIn, (SELECT starname
                           FROM StarsIn
                           HAVING count(*)>10
                           GROUP BY starname) Productive
     WHERE title=movietitle AND
           year=movieyear AND
           Productive.starname=StarsIn.starname;
```

**a)** Give a description in words of what each of the queries computes. Emphasis should be put on giving a short and clear description.

**b)** Write a relational algebra expression (using extended operators if needed) corresponding to each of the queries. You may use the aggregation operator `COUNT(*)` to obtain a count of all tuples. **Hint:** First rewrite the subquery to avoid the `HAVING` clause.

The following SQL query computes the title and year of all color movies in the database from before 1939:

```
     SELECT title, year
     FROM Movie
     WHERE year<1939 AND inColor=1;
```

**c)** Write a sequence of SQL statements that permanently remove from the database information on all color movies from before 1939, and all actors in the database starring *only* in such movies.

## 3 Indexing (10%)

Consider again query `Q1` of Problem 2.

**a)** Suggest an index which could speed up `Q1`, and write SQL (using the syntax presented in GUW) to create the index.

Suppose that `Q1` takes 100 ms with an index, and 1100 ms without an index (at that the size of the relations is not changing too much, so this number can be regarded as fixed). Also, assume that updating the index takes 100 ms per update (insertion or deletion).

**b)** How many updates must there be for each query before the time used for updating the index exceeds the time saved on queries.

## 4 Normalization (15%)

Consider the following instance of a relation R:

| `saleID` | `salesman` | `regNo` | `make` | `office` |
|---|---|---|---|---|
| 42 | B. Honest | VY 34718 | Opel | City |
| 53 | W. Gates | PQ 11112 | Ford | Redwood |
| 87 | B. Honest | MX 32781 | Ford | City |
| 99 | L. R. Harald | AB 12345 | Porche | City |

The functional dependencies of R, not including trivial ones, are:

1. `saleID` $\rightarrow$ `salesman regNo make office`
2. `salesman` $\rightarrow$ `office`
3. `regNo` $\rightarrow$ `make`

**a)** Decompose the relation into BCNF. For each step of the decomposition procedure, state what functional dependency it is based on, and give the relation schemas after the step has been carried out.

**b)** State the relation instances in your BCNF schema corresponding to the above instance of R. Give an example of an update anomaly of the original relation schema that has been eliminated in the BCNF schema.

## 5 Database constraints (15%)

We again consider the relation schemas from Problem 2. In this problem we suppose that the schema for `StarsIn` contains the declarations

```
FOREIGN KEY (movietitle,movieyear) REFERENCES Movie(title,year) ON DELETE CASCADE,
FOREIGN KEY (starname) REFERENCES MovieStar(name) ON DELETE SET NULL
```

and that corresponding `UNIQUE` constraints are set on `Movie` and `MovieStar`.

**a)** Explain what happens, if anything, to maintain the referential integrity constraints in each of the following cases:

1. A tuple in `Movie` is deleted.
2. A tuple in `MovieStar` is deleted.
3. A tuple in `StarsIn` is deleted.

Suppose we issue the SQL command:

```
INSERT INTO StarsIn VALUES ('Total Recall',1990,'Arnold Schwarzenegger');
```

**b)** Explain what is the result of the insertion command in each of the following cases:

1. The movie `Total Recall` does not exist in `Movie`.
2. The name `Arnold Schwarzenegger` does not exist in `MovieStar`.

## 6 Transactions (10%)

Consider the following three transactions on the relation `students(name,grade)`:

| Transaction A |
|---|
| `INSERT INTO students VALUES ('F. Student',5);` |
| `INSERT INTO students VALUES ('A. Student',13);` |
| `INSERT INTO students VALUES ('C. Student',8);` |

| Transaction B |
|---|
| `UPDATE students SET grade=grade+1 WHERE grade<11 AND grade>3;` |

| Transaction C |
|---|
| `UPDATE students SET grade=3 WHERE grade=5;` |

**a)** Suppose that the transactions run more or less simultaneously at isolation level `READ COMMITTED`, and that `students` is initially empty. List all 4 possible instances of `students` after all transactions have committed (assuming that no transaction is rolled back).

# Introduction to Databases

IT University of Copenhagen

Trial exam, Fall 2003

This exam consists of 6 problems with a total of 12 questions. The weight of each problem is stated. You have 4 hours to answer all 12 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. Remember to write the page number, your name, and your CPR-number on each page of your written answer. The complete assignment consists of 5 numbered pages (including this page).

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.

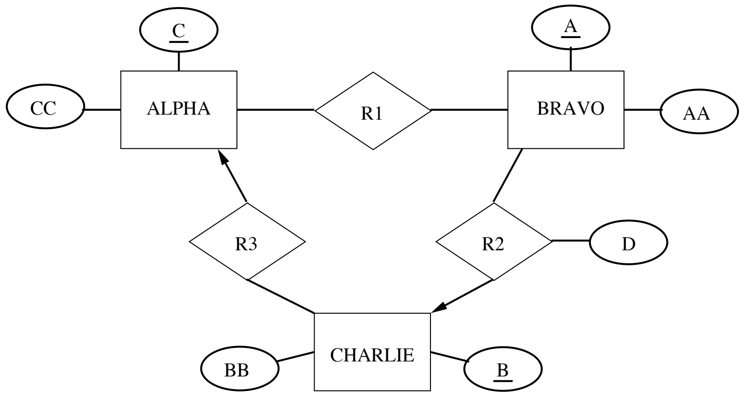
## 1 Database design (25%)

This problem has two unrelated parts. In the first part we consider the task of designing a database for a book club. The club has a selection of books for sale to its members. There is a "book of the month" every month, which is automatically shipped to all members who have not cancelled the shipment before a certain date. The database should contain information about:

- The selection of books, including information on titles, authors, publishers, and ISBN numbers (each book has a unique ISBN number).
- The members, including information on names, addresses, and payment due.
- The books selected as "book of the month".
- Which books have been shipped to each customer, and in what quantity.
- Which "books of the month" have been cancelled by each customer.

**a)** Draw an E/R diagram for the database. Emphasis should be put on using the design principles of GUW section 2.2. Remember to include appropriate keys and constraints.

In the second part we consider the following E/R diagram:



**b)** Perform a conversion of the E/R diagram into relation schemas, using the method described in GUW. You should combine relations when possible. Write SQL statements to create the relations, including key constraints. Assume that all attributes are integers.

## 2 Normalization (15%)

Consider the following instance of a relation R:

| `student` | `course` | `grade` | `address` |
|---|---|---|---|
| William Gates | Operating systems | 6 | Microsoft Way 1 |
| Jakob Nielsen | User interfaces | 8 | Silicon Valley 22 |
| Mads Tofte | IT Management | 10 | Glentevej 67 |
| Jakob Nielsen | User interfaces | 8 | Glentevej 38 |
| William Gates | Intro. to databases | 7 | Microsoft Way 1 |
| Steve Jobs | Intro. to databases | 10 | Apple Drive 10 |
| Steve Jobs | Operating systems | 10 | Apple Drive 10 |

The functional and multivalued dependencies of a relation schema can be determined only from knowledge of the data that the relation is supposed to contain. However, from a given relation instance we may be able to say that a given FD or MVD does **not** hold.

**a)** Which of the following functional dependencies can be seen to **not** hold for R: `student` $\rightarrow$ `address`, `address` $\rightarrow$ `student`, `course` $\rightarrow$ `grade`.
Argue in each case why a functional dependency is possible or impossible.

**b)** Which of the following multivalued dependencies can be seen to **not** hold for R: `student` $\twoheadrightarrow$ `address`, `address` $\twoheadrightarrow$ `student`, `course` $\twoheadrightarrow$ `grade`.
Argue in each case why a multivalued dependency is possible or impossible.

## 3 SQL queries and relational algebra (20%)

Again consider the relation R from Problem 2. The following SQL query is used in the statistics office:

```
SELECT student, avg(grade) AS GPA
FROM R
HAVING count(*) > 1
GROUP BY student;
```

**a)** What is the result of the query when run on the instance of R from Problem 2? Describe in words what the query computes in general.

**b)** Rewrite the query to avoid the `HAVING` clause. That is, write an equivalent SQL query that does not contain the keyword `HAVING`. **Hint:** Create a new attribute to contain a count of tuples, and select those tuples where this attribute has value $> 1$.

**c)** Write an expression in relational algebra which is equivalent to the above query. **Hint:** Use your answer from b).

## 4 Authorization (15%)

The user `alice` has just created a relation `R(user,info)`, and issues the following SQL commands:

```
GRANT INSERT ON R TO bob WITH GRANT OPTION;
GRANT SELECT ON R TO bob WITH GRANT OPTION;
GRANT SELECT ON R TO claire;
```

Consider the following SQL commands:

1. `SELECT * FROM alice.R WHERE user='claire';`
2. `INSERT INTO alice.R VALUES ('claire','clairvoyant');`
3. `GRANT SELECT ON alice.R TO dorothy;`

**a)** State for each of the three users `bob`, `claire`, and `dorothy`, which of the above SQL commands he/she has autorization to execute.

Now assume that `bob` executes the command

```
GRANT INSERT ON R TO claire;
```

and `alice` then executes the command

```
REVOKE INSERT ON R FROM bob CASCADE;
```

**b)** Again, state for each of the three users `bob`, `claire`, and `dorothy`, which of the above SQL commands he/she has autorization to execute at this point.

## 5 Transactions (10%)

Consider the following three transactions on the relation `accounts(no,balance,type)`:

**Transaction A**

```
UPDATE accounts SET balance=balance*1.02 WHERE type='savings';
UPDATE accounts SET balance=balance*1.01 WHERE type='salary' AND balance<0;
UPDATE accounts SET balance=balance*1.07 WHERE type='salary' AND balance>0;
```

**Transaction B**

```
UPDATE accounts SET type='salary' WHERE no=12345;
```

**Transaction C**

```
UPDATE accounts SET balance=balance-1000 WHERE no=12345;
```

The purpose of Transaction A is to add interest to the balance of accounts, depending on the type and balance. Transaction B changes the type of a particular account. Transaction C makes a withdrawal from a particular account.

**a)** Suppose that the transactions are run more or less simultaneously at isolation level `READ COMMITTED`. What results of running the transactions are possible in this case, but not if the transactions had been run at isolation level `SERIALIZABLE`?

## 6 Database efficiency (15%)

You are appointed the administrator of a new DBMS that is used to register business transactions of a large multinational company, for use by the top management. Every day about 10,000 new business transactions are registered, and there are about 10 queries for old business transactions (identified by a transaction code). Having learnt about indexes on IDB, you consider placing an index on the transaction code to speed up queries. A full table scan processes 10,000 business transactions per second, while an index lookup can be done in 100 ms. The time used to insert is 40 ms without an index, and 100 ms with an index.

**a)** With 100,000 business transactions registered, what is the daily processing time (registering new + looking up old business transactions) with and without an index?

**b)** When will it become an advantage (in terms of total processing time) to use an index?