EXAM
Databases (DIT620/TDA355/TDA356/TDA357)

DAY: 15 Mar 2011 TIME: 8:30 – 12:30 PLACE: M

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

IMPORTANT

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

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

The problems in each block are ordered by increasing difficulty. Hence the A problems are easy, but aim to cover the full basics of its area. The B and C level problems are more difficult, and aim to test your knowledge of the areas beyond the mere basics. If you only solve A problems your maximum score is 35 points, and if you only solve the B problems where there are also C problems it is 40 points.

Please observe the following:

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

Good advice

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

Good Luck!

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

1A (8p)

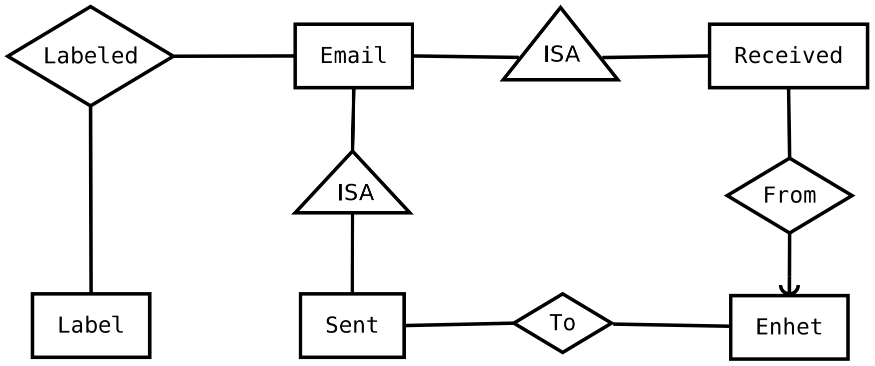
(i) (4p)

A small online book store want a database to keep track of books and authors, customers and sales.
Books are identified by their unique ISBN numbers. For each book, its title and author(s) should be stored, as well as its base price. Each author should be given a unique author ID, to distinguish between different authors with the same name.
Customers have a unique username, but also need to submit their name and shipping address. A customer may make purchases, and the records of those should be stored. Each purchase is assigned a unique serial number. A purchase can contain any number of books that the customer puts in their "shopping cart", possibly many copies of the same book. For each book, the purchase must also record the price at which it is bought, since (due to campaigns) it may differ from the book's base price.
The store frequently runs campaigns to increase sales. Each campaign includes one or more books for which the price is lowered. There are two different kinds of campaigns – either the books included are offered at a fixed lower price, or the books included are offered at a percentage discount. Information about these campaigns must also be included in the database, to help calculate the price for each purchase.

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

(ii) (4p)

The E/R diagram below is part of an E/R diagram for a database used for a simple mail program.



Translate the ER diagram above into a set of relations, using one of the three possible schemes for translating sub-entities. State which scheme you have used, and why it is better than the other two for this domain. Mark keys and references clearly in your answer.

1B (12p)

Below is a database schema used for the back-end of a blog server:

$Authors(\underline{login}, displayName, encPwd, description)$
$Posts(\underline{postId}, time, author, title, text)$
    $author \rightarrow Authors.login$
$Comments(\underline{post}, \underline{commentNr}, time, text)$
    $post \rightarrow Posts.postId$
$CommentsOnComments(\underline{post}, \underline{comment}, onComment, depth)$
    $(post, comment) \rightarrow Comments.(post, commentNr)$
    $(post, onComment) \rightarrow Comments.(post, commentNr)$
$AuthorComments(\underline{post}, \underline{comment}, author)$
    $(post, comment) \rightarrow Comments.(post, commentNr)$
    $author \rightarrow Authors.login$
$OtherComments(\underline{post}, \underline{comment}, name, url)$
    $(post, comment) \rightarrow Comments.(post, commentNr)$

All of this should be self-explanatory (or part of the problem), but if there is something that you don't understand, don't hesitate to ask. (Incidentally this is the same schema used in block 3 and onwards. A fuller explanation of the domain is given there, but won't really help with this particular problem.)
Reconstruct the ER diagram that led to these relations and constraints.

Note: At one point in the translation from diagram to relations, two attributes were unified, in lieu of forcing them to be equal through a constraint.

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

2A (8p)

A simple bug tracker software uses a database to manage filed issues, users and milestones. Each issue submitted is given a unique tracking number. It also records the user who reported the issue, the version of the software that the bug was found in, a bug category, a priority, a milestone where the bug should be handled, and the date for that milestone.
Categories, versions, priorities and milestones are customizable by the administrator of the software, so they are all simply stored in separate tables. Milestones are given a name and a date of completion, both of which are unique.
For users, the system simply stores a unique login, a password (encrypted, of course) and an email address.
Users can comment on issues through a simple commenting system, for which the system stores the issue number, a running comment number and the user who left the comment.
Issues become assigned to users, who are then responsible for fixing them. A responsible user is said to *own* the issue. Any comments left by other users for the issue in question are automatically sent to the owner, for which the system stores the owner's email address.
Other users can also opt to receive email updates for an issue, by explicitly subscribing to it.

The schema they use for their database looks like this:

$Users(\underline{username}, email)$
$Categories(\underline{category})$
$Versions(\underline{version})$
$Priorities(\underline{priority})$
$Milestones(\underline{dateDue}, \underline{name})$

$Issues(\underline{issueNr}, milestone, dateDue, reportedBy, timeReported, category, version, priority, text)$
  $reportedBy \rightarrow Users.username$
  $(milestone, dateDue) \rightarrow Milestones.(name, dateDue)$
  $category \rightarrow Categories.category$
  $version \rightarrow Versions.version$
  $priority \rightarrow Priorities.priority$

$AssignedIssues(\underline{issueNr}, owner, ownerEmail)$
  $issueNr \rightarrow Issues.issueNr$
  $owner \rightarrow Users.username$
$Subscriptions(\underline{issueNr}, user, userEmail)$
  $issueNr \rightarrow Issues.issueNr$
  $user \rightarrow Users.username$

$Comments(\underline{issueNr}, \underline{commentNr}, byUser, time, text)$
  $issueNr \rightarrow Issues.issueNr$
  $byUser \rightarrow Users.username$

This schema is not fully normalized, and thus suffers from a number of problems. It is your task to solve these by normalization of the schema.

(i) (4p)
For the given domain, identify all functional dependencies that are expected to hold.

(ii) (1p)
With the dependencies you have found, identify all BCNF violations in the relations of the database. For each violation, also specify what kinds of problems that could arise if it was not resolved.

(iii) (3p)
Do a complete normalization of the schema, so that all relations are in BCNF. (It's the end product that's important, not the steps you take to get there.)

2B (12p)

The tax department keeps track of where people live, and stores this information in a database. People are identified by a social security number (*ssNr*) and are assumed to have an address. An address consists of many components: a street name and a number, a postal code specifying an area, a city name, and a state.

Street names are not unique for the whole country, but can be assumed to be unique within a given city (we assume everyone lives in a city for the sake of the problem). An area identified by a postal code is always completely within one city, and a city is in a specific state. A postal code area typically includes many streets. In some cases, a given street could cross over into more than one postal code area, but each number on that street belongs to a specific area.

The following relation sums up all the attributes that should be stored in the database:

TaxRecord(*ssNr*, *firstName*, *lastName*, *streetName*, *streetNr*, *postalCode*, *city*, *state*)

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

(i) (4p)
Find all functional dependencies that are expected to hold for this domain given the domain description above.

(ii) (8p)
Show two decompositions of *TaxRecord*, one that satisfies BCNF and one that only satisfies 3NF. Discuss the difference betwen your two resulting schemas, highlighting the benefits and drawbacks of each choice.

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

The domain for this block, and for several following blocks as well, is that of a database for the back-end of a blog server. The database keeps tracks of registered authors who make posts to the blog. Anyone can then comment on those posts, or on the comments of others.

You are given the following schema of their intended database:

$\mathit{Authors}(\underline{\mathit{login}}, \mathit{displayName}, \mathit{encPwd}, \mathit{description})$
$\mathit{Posts}(\underline{\mathit{postId}}, \mathit{time}, \mathit{author}, \mathit{title}, \mathit{text})$
    $\mathit{author} \rightarrow \mathit{Authors.login}$
$\mathit{Comments}(\underline{\mathit{post}}, \underline{\mathit{commentNr}}, \mathit{time}, \mathit{text})$
    $\mathit{post} \rightarrow \mathit{Posts.postId}$
    text is at least 10 characters long
$\mathit{CommentsOnComments}(\underline{\mathit{post}}, \underline{\mathit{comment}}, \mathit{onComment}, \mathit{depth})$
    $(\mathit{post}, \mathit{comment}) \rightarrow \mathit{Comments}.(\mathit{post}, \mathit{commentNr})$
    $(\mathit{post}, \mathit{onComment}) \rightarrow \mathit{Comments}.(\mathit{post}, \mathit{commentNr})$
$\mathit{AuthorComments}(\underline{\mathit{post}}, \underline{\mathit{comment}}, \mathit{author})$
    $(\mathit{post}, \mathit{comment}) \rightarrow \mathit{Comments}.(\mathit{post}, \mathit{commentNr})$
    $\mathit{author} \rightarrow \mathit{Authors.login}$
$\mathit{OtherComments}(\underline{\mathit{post}}, \underline{\mathit{comment}}, \mathit{name}, \mathit{url})$
    $(\mathit{post}, \mathit{comment}) \rightarrow \mathit{Comments}.(\mathit{post}, \mathit{commentNr})$

The comment system requires some extra attention. Comments always belong to a specific post, and depend on that post for identification. Comments may be given directly on the post, or in reply to some other comment, in which case they will be displayed indented beneath that comment. For this reason, the *CommentsOnComments* relation stores if a comment (*comment*) is a reply to another comment (*onComment*). To simplify printing, each such comment also stores the *depth* at which it is placed – for instance, a comment placed on a comment placed on a comment placed on the post would be at depth 2. Comments placed directly on the post are at depth 0. (Technically this is derived information that could be calculated from the information already in the database – though see question 4C.)

The blog displays comments made by registered authors differently from those made by outsiders. If a comment is made by an author, this is registered in the *AuthorComments* table. If a comment is made by an outsider, the comment system asks for a name to display and (optionally) a url to the commenter's homepage.

3A (4p)

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

3B (6p)

Note the relation *Comments* storing basic information for comments. Its key consists of the post it belongs under, and a number to indentify it among the comments for that post. When creating this table, we should specify this key as a primary key constraint. Does it matter in which order we give the two attributes (*post* and *commentNr*) when writing this constraint? If yes, state how it matters, as well as which order would be preferable and why for this specific domain.

3C (8p)

When a comment is given through the blog front-end, it should be stored in the tables as befits a comment of that type. Each comment will generate information to store in several different tables: *Comments*, possibly *CommentsOnComments*, and either *AuthorComments* or *OtherComments*.

Sketch an overview of how to ensure, through the use of views and/or triggers, and privileges, that the database stores the correct information in the correct tables.

For any views you want to use, give the schema and explain its intended contents.

For any trigger you might include, list the trigger head ([BEFORE/AFTER/INSTEAD OF] [INSERT/DELETE/UPDATE] ON which element), and describe its intended operation in broad terms (a simple overview in plain English would be fine, you don't have to write any code).

Also specify what privileges the commenting front-end should be granted.

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

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

4A (4p)

(i) (2p)
Write an SQL query that lists the number of posts each author has made. All authors should be listed, including those that have not made any posts.

(ii) (2p)
Write an SQL query that lists all posts with at least 10 comments.

4B (6p)

Write a query that lists, for each author, all their posts *and* comments together, ordered by time. The query result should contain four columns: author login, time of post/comment, text of post/comment, and a column saying either 'post' or 'comment' to distinguish between the two.

4C (8p)

(i) (6p)
Write a query that finds the post(s) with the highest depth of comments.

(ii) (2p)
Without the "derivable" *depth* attribute in the schema, the query above could not be written using SQL. Why?

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

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

5A (3p)

(i) (2p)
What does the following relational algebra expression compute (answer in plain text):

$$\tau_{-y}(\gamma_{author,name,AVG(x)\rightarrow y}(\pi_{author,name,LENGTH(text)\rightarrow x}(\sigma_{author=login}(Posts \times Authors))))$$

(ii) (1p)
Translate the following relational algebra expression to SQL:

$$\gamma_{post,COUNT-DISTINCT(name)\rightarrow posters}(Comments \overset{o}{\bowtie} OtherComments)$$

5B (4p)

Translate the following SQL query to relational algebra:

```
SELECT post, onComment, COUNT(comment) AS comments
FROM CommentsOnComments
GROUP BY post, onComment
HAVING comments >= 10
```

5C (6p)

Write a relational algebra expression that for each post counts the number of comments made by *other* authors than the one writing the post.

Block 6 - Transactions **max 6p**

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

6A (3p)

Consider the following program (partly in pseudo-code), for inserting new author comments. In the code I prefix program variables with : just to distingush them from attributes (i.e. you don't need to worry about any connection to PSM or the like).

```
1 ... author (:author) writes a comment (:text) on a post (:post) ...
2 SELECT MAX(commentNr)+1 INTO newCommNr
   FROM Comments
   WHERE post = :post;
3 if (:newCommNr IS NULL) then commNr := 1;
4 INSERT INTO Comments VALUES
    (:post, :newCommNr, :text, SYSDATE)
6 INSERT INTO AuthorComments VALUES
    (:post, :newCommNr, :author)
```

(i) (1p)
For the program as specified above, what atomicity problems could arise if it was not run as a transaction?

(ii) (2p)
For the program as specified above, what isolation problems could arise if it was not run as a *serializable* transaction?

6B (6p)

Compare what would happen if the program above was run as a transaction with isolation level SERIALIZABLE compared to if it was run with isolation level READ COMMITTED. Point out benefits and drawbacks of the two choices for this particular problem.

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

The following DTD attempts to faithfully model the same domain and constraints for the blog server as the relations used in the previous blocks.

```
<!DOCTYPE BlogServer [

<!ELEMENT Blogserver (Author*, Post*)>
<!ELEMENT Author     (#CDATA)>
<!ELEMENT Post       (Text, Comment*)>
<!ELEMENT Text       (#CDATA)>
<!ELEMENT Comment    (Text, (AuthorComment | OtherComment), Comment*)>
<!ELEMENT AuthorComment (#EMPTY)>
<!ELEMENT OtherComment  (#EMPTY)>

<!ATTLIST Author
    login  ID    #REQUIRED
    encPwd CDATA #REQUIRED
    name   CDATA #REQUIRED>
<!ATTLIST Post
    postId ID    #REQUIRED
    time   CDATA #REQUIRED
    author IDREF #REQUIRED>
<!ATTLIST Comment
    time   CDATA #REQUIRED
    depth  CDATA #REQUIRED>
<!ATTLIST AuthorComment
    author IDREF #REQUIRED>
<!ATTLIST OtherComment
    name   CDATA #REQUIRED
    url    CDATA #IMPLIED>

]>
```

7A (5p)

(i) (2p)
Give an example XML document that is valid with respect to the DTD above, and that contains information about at least one post.

(ii) (3p)
Here's an alternative to parts of the DTD above:

```
<!ELEMENT Blogserver (Post*)>
<!ELEMENT Post    (Text, Author, Comment*)>
<!ELEMENT Author  (#CDATA)>
```

All other elements remain as before. Why is this solution less suitable?

7B (8p)

(i) (4p)
Explain the difference in expressive power between XPath and XQuery. Give an example of a query that could not be expressed using only XPath for the DTD given above.

(ii) (4p)
When answering this question, disregard types, usage (required vs NULL etc) and references (which are stupid in DTDs), and compare only the basic structure of the two schemas.
While the DTD above attempts to faithfully represent the same model as the relational schema listed in block 3, there are some small differences in what data that could be stored. List one such difference.