CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

Examination in Databases, TDA357/DIT620

Tuesday 13 January 2015, 14:00-18:00

Examiner:	Graham Kemp (telephone 772 54 11, room 6475 EDIT) The examiner will visit the exam room at 15:00 and 17:00.
Results:	Will be published by the end of January at the latest.
Exam review:	See course web page for time and place: http://www.cse.chalmers.se/edu/year/2014/course/TDA357/HT2014/
Grades:	Grades for Chalmers students (TDA357) are normally determined as follows: ≥ 48 for grade 5; ≥ 36 for grade 4; ≥ 24 for grade 3. Grades for GU students (DIT620) are normally determined as follows: ≥ 42 for grade VG; ≥ 24 for grade G.
Help material:	One A4 sheet with hand-written notes. You may write on both sides of that sheet. If you bring a sheet, it must be handed in with your answers to the exam questions. English language dictionaries are allowed.

Specific instructions:

- Please answer in English where possible. You may clarify your answers in Swedish if you are not confident you have expressed yourself correctly in English.
- Begin the answer to each question on a new page.
- Write clearly; unreadable = wrong!
- Fewer points are given for unnecessarily complicated solutions.
- Indicate clearly if you make any assumptions that are not given in the question.
- Write the page number and question number on every page.

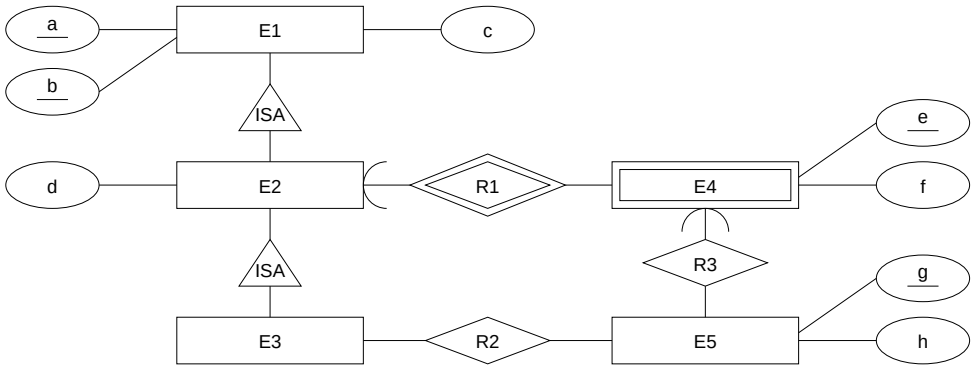
Question 1. 12 p

a) Consider the following domain description.

> Students and staff at a university can book rooms for meetings. Each person is identified by their university e-mail address. People's real names are also stored. The office telephone number of each member of staff is stored. Rooms are identified by a combination of the room number and the name of the building where the room is located. Building names are unique at the university, but two or more buildings can have rooms with the same number. Each booking is assigned a unique identifier. The year, week number, day (e.g. Monday), start time and end time of each room booking is stored, and also the person who made the booking. Assume that bookings are automatically deleted from the system after the meeting time has passed. Members of staff are allowed to make many room bookings, but each student can only have one room booking in the system at any time. For each booking made by a student, a course code must be recorded with the booking.

Draw an E-R diagram that correctly models this domain.
(6p)

b) Translate the following E-R diagram into a set of relations, clearly marking all references and keys.
Use the E-R approach to translate subclass-superclass (ISA) relationships.



(6p)

Question 2. 10 p

Suppose we have relation $R(A, B, C, D, E, F, G)$ with keys ABC, ACD and ACG, and functional dependencies $A \to E$, $AB \to D$, $ABC \to F$, $ABC \to G$, $CD \to G$, $E \to F$, $G \to B$.

a) i) State, with reasons, which of the FDs listed above violate BCNF. (1p)
ii) Decompose relation R to BCNF. Show each step in the normalisation process, and at each step indicate which functional dependency is being used. Indicate keys and references for the resulting relations. (4p)

b) i) Which attributes of R are prime? (1p)
ii) State, with reasons, which of the FDs listed above violate 3NF. (1p)
iii) Decompose relation R to 3NF. (3p)

Question 3. A university uses a database to manage information about applications to its Master's programmes. This database has the following relations:

10 p

Programmes(<u>code</u>, name, department, numPlaces)
Applicants(name, address, <u>appNumber</u>)
AppliesFor(<u>applicant</u>, <u>programme</u>, choiceNumber, meritScore, status)

Each programme is identified by its code. The programme name, the name of the department that arranges the programme, and the number of available places for students in the programme this year are stored.

Each applicant's name, address and (unique) applicant number are stored.

Each applicant can apply to up to four programmes, and attribute *choiceNumber* records the applicant's prioritisation of the programmes (e.g. someone might apply to two programmes: Computer Science as choice 1 and Physics as choice 2). Possible values for this attribute are 1, 2, 3 and 4. The choice numbers for a particular applicant must be unique (e.g. an applicant cannot apply to two programmes giving both as choice number 1).

When applications to a programme are processed by the university, the applications are ranked, and each application is assigned a merit score between 1 and 1000 (e.g. the top three applications for a programme might be assigned merit scores 920, 890 and 870). No two applicants can get the same merit score for the same programme (you can assume that no programme has more than 1000 applicants). A default merit score of '0' indicates that the application has not been evaluated yet.

Attribute *status* has the value 'unprocessed', 'offered' (a place on the programme has been offered to the applicant), 'accepted' (the applicant received an offer and accepted it), 'declined' (the applicant received an offer but declined it), 'offer withdrawn' (an offer was made, but the applicant did not respond before the deadline for accepting the offer), or 'rejected' (the university decided not to offer a place on this programme to this applicant). The default status is 'unprocessed'.

a) Suggest references for these relations.
Write SQL statements that create these relations with constraints in a DBMS.
(4p)

b) Choice numbers for the applications from a particular applicant must be consecutive numbers starting from 1 (i.e. if someone applies to four programmes, their choice numbers will be 1, 2, 3 and 4; to three programmes the choice numbers are 1, 2 and 3; to two programmes the choice numbers are 1 and 2; to one programme the choice number is 1).
Write an assertion that checks this. (You can assume that any constraints in your solution to part (a) are enforced.)
(2p)

c) If the number of students who have accepted places in a programme has reached the number of available places in that programme, then all unprocessed applications to that programme can be rejected.
Write a trigger that implements this.
What privileges must a user have in order to fire this trigger?
(4p)

Question 4. Assume the same relations as in Question 3:

6 p

Programmes(code, name, department, numPlaces)
Applicants(name, address, appNumber)
AppliesFor(applicant, programme, choiceNumber, meritScore, status)

a) Write a relational algebra expression that finds the names of applicants and programme names where the applicant applied to a programme arranged by the Physics department as their first choice.
(3p)

b) Write a relational algebra expression that finds the code of the programme(s) selected as first choice by the largest number of applicants.
(3p)

Question 5. Assume the same relations as in Question 3:

9 p

Programmes(code, name, department, numPlaces)
Applicants(name, address, appNumber)
AppliesFor(applicant, programme, choiceNumber, meritScore, status)

a) Write an SQL query that finds the names of applicants and programme names where the applicant applied to a programme arranged by the Physics department as their first choice.
(3p)

b) Write an SQL query that finds the code of the programme(s) selected as first choice by the largest number of applicants.
(3p)

c) Write an SQL query that finds how many applicants chose the programme with code 'MPALG' in preference to the programme with code 'MPCSN'. The total should include those applicants who applied to both programmes but gave a lower choice number for 'MPALG', and also applicants who applied to 'MPALG' but did not apply to 'MPCSN'.
(3p)

Question 6. Suppose a system for handling applications to a university's Master's programmes (as described in Question 3) has a transaction, T, with the following steps:
5 p

T_1: get a programme code from the user (store this in p), and find the unprocessed application to this programme with the highest merit score
T_2: count how many applicants have accepted a place on programme p
T_3: offer the applicant identified in step T_1 a place on programme p, and set the *status* attribute in the relevant row of relation *AppliesFor* to 'offered'
T_4: get response from the applicant ('accepted' or 'declined') and set the *status* attribute in the relevant row of relation *AppliesFor* accordingly
T_5: count how many applicants have accepted a place on programme p

a) In database transactions, what are *phantoms*? Refer to transaction T in your answer. (2p)

b) Discuss how the results of steps T_2 and T_5 could differ if transaction T is run with different isolation levels.
Discuss what isolation level(s) would be most appropriate for running transaction T. (3p)

Question 7. Consider the following piece of XML:
8 p

```
<Question7>
 <Applicants>
  <Applicant name="Andersson" appNum="a1" />
  <Applicant name="Jonsson"   appNum="a2" />
  <Applicant name="Larsson"   appNum="a3" />
 </Applicants>
 <Choices>
  <Choice applicant="a1" code="MPSOF" choiceNum="1" meritScore="750" />
  <Choice applicant="a1" code="MPALG" choiceNum="2" meritScore="750" />
  <Choice applicant="a1" code="MPCSN" choiceNum="3" meritScore="800" />
  <Choice applicant="a2" code="MPALG" choiceNum="1" meritScore="700" />
  <Choice applicant="a3" code="MPCSN" choiceNum="1" meritScore="850" />
  <Choice applicant="a3" code="MPALG" choiceNum="2" meritScore="850" />
 </Choices>
</Question7>
```

a) Write a Document Type Definition (DTD) for the XML that is given above. (2p)

b) Write an XPath expression that finds Choice elements where the choice number is 1 and the merit score is greater than 800. (1p)

c) The flexibility of XML enables us to nest elements in a more natural way than in the example shown at the top of this question. Write a piece of XML that contains the same information as in the example shown above, but which uses nesting, and avoids duplication of applicant identifiers. (2p)

d) Assuming that the XML shown above is in file *exam.xml*, write an XQuery expression that constructs your solution to part (c). (3p)

CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

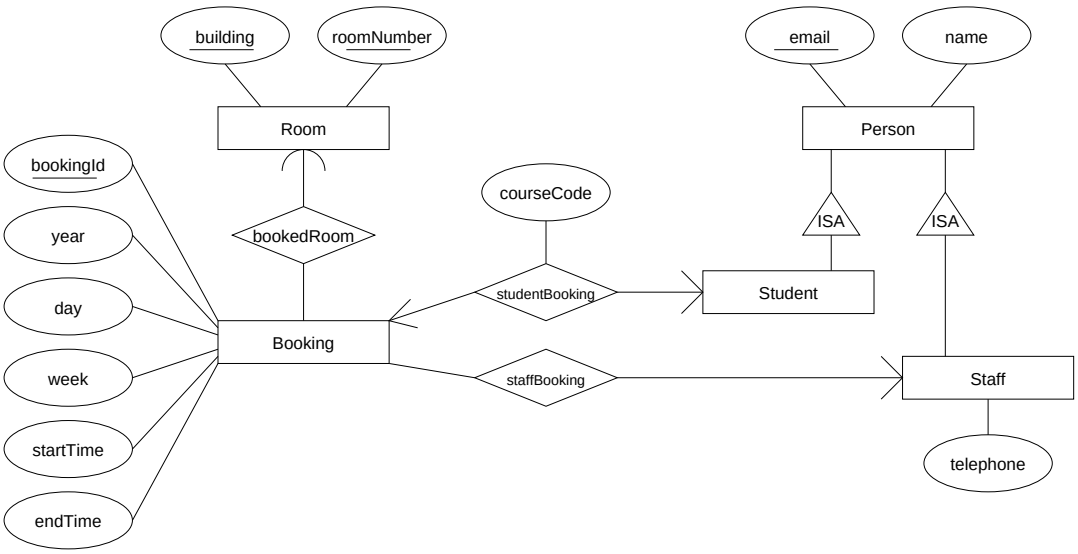
Examination in Databases, TDA357/DIT620

Tuesday 13 January 2015, 14:00-18:00

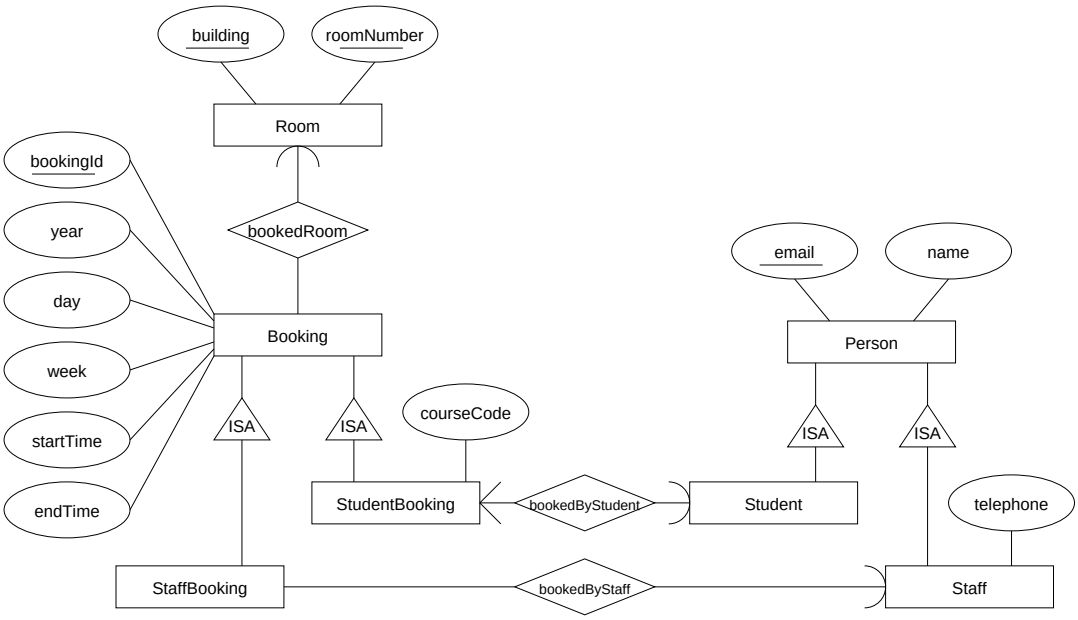
Solutions

Updated 2015-02-09

Question 1. a) This suggestion is acceptable:
12 p



However, that suggestion does not model the multiplicity of the relationships between booking, students and staff in a good way. These can be represented in a better way with subclasses of booking:



b) $E1(\underline{a}, \underline{b}, c)$
$E2(\underline{a}, \underline{b}, d)$
$\quad (a, b) \rightarrow E1.(a, b)$
$E3(\underline{a}, \underline{b})$
$\quad (a, b) \rightarrow E2.(a, b)$
$E4(\underline{a}, \underline{b}, \underline{e}, f)$
$\quad (a, b) \rightarrow E2.(a, b)$
$E5(\underline{g}, h, a, b, e)$
$\quad (a, b, e) \rightarrow E4.(a, b, e)$
$R2(\underline{a}, \underline{b}, \underline{g})$
$\quad (a, b) \rightarrow E3.(a, b)$
$\quad g \rightarrow E5.g$

Question 2. a) i) ABC->F and ABC->G do not violate BCNF since their left sides are keys.
10 p The other 5 FDs violate BCNF.

ii) Decompose R on A->E
{A}+ = {AEF}

R1(_A,E,F)
R2(A,B,C,D,G)
A -> R1.A

Decompose R2 on AB->D
{AB}+ = {ABD}

R21(_A,_B,D)
A -> R1.A
R22(A,B,C,G)
(A,B) -> R21(A,B)

Decompose R1 on E->F
{E}+ = {E,F}

R11(_E,F)
R12(E,_A)
E -> R11.E

Decompose R22 on G->B
{G}+ = {GB}

R221(_G,B)
R222(_A,_C,_G)
G -> R221.G

Update reference for R21: A -> R12.A

b) i) A, B, C, D, G

ii) A->E, E->F

iii) R1(_A,E)
R2(_A,_B,D)
R3(_A,_B,_C,F,G)
R4(_C,_D,G)
R5(_E,F)
R6(_G,B)

Question 3. 10 p

a)
```
CREATE TABLE Programmes (
    code        CHAR(5) PRIMARY KEY,
    name        VARCHAR(50),
    department  VARCHAR(50),
    numPlaces   INT
);

CREATE TABLE Applicants (
    name        VARCHAR(30),
    address     VARCHAR(50),
    appNumber   INT PRIMARY KEY
);

CREATE TABLE AppliesFor (
    applicant REFERENCES Applicants(appNumber),
    programme REFERENCES Programmes(code),
    choiceNumber INT CHECK (choiceNumber BETWEEN 1 AND 4),
    meritScore   INT DEFAULT 0 CHECK (meritScore BETWEEN 0 AND 1000),
    status VARCHAR(30) DEFAULT 'unprocessed'
                       CHECK (status IN ('unprocessed', 'offered',
                                'accepted', 'declined',
                                'offer withdrawn', 'rejected') ),
    PRIMARY KEY (applicant, programme),
    CONSTRAINT choices_unique UNIQUE (applicant, choiceNumber)
);
```

b)
```
CREATE ASSERTION ConsecutiveChoices CHECK
   ( NOT EXISTS (
        SELECT   applicant
        FROM     AppliesFor
        GROUP BY applicant
        HAVING   MAX(choiceNumber) > COUNT(choiceNumber) ) )
```

c)
```
CREATE OR REPLACE TRIGGER CourseFull
AFTER UPDATE OF status ON AppliesFor
REFERENCING NEW AS newrow
FOR EACH ROW
WHEN ( newrow.status = "accepted" )
BEGIN
  IF ( ( SELECT COUNT(applicant)
      FROM   AppliesFor
      WHERE  programme = :newrow.programme
         AND status = "accepted" ) >= ( SELECT  numPlaces
                                        FROM Programmes
                                        WHERE code = :newrow.programme ) ) THEN

        UPDATE AppliesFor
        SET    status = "rejected"
        WHERE  status = "unprocessed" AND programme = :newrow.programme;
    END IF;
END;
```

Privilege UPDATE of attribute status in table AppliesFor is needed.

Question 4. 6 p

a) $\pi_{Applicants.name,Programmes.name}(Applicants \bowtie_{applicant=appNumber} ((\sigma_{department="Physics"} Programmes) \bowtie_{code=programme} (\sigma_{choiceNumber=1} AppliesFor)))$

b) $R := \gamma_{programme,COUNT(appliocant)\rightarrow numApplicants}(\sigma_{choiceNumber=1} AppliesFor)$

$\pi_{programme}(\sigma_{numApplicants=maxApplicants(\gamma_{MAX(numApplicants)\rightarrow maxApplicants} R)} R)$

Question 5. 9 p

a)
```
SELECT  Applicants.name, Programmes.name
FROM    Applicants, Programmes, AppliesFor
WHERE   applicant = appNumber
        AND code = programme
        AND department = 'Physics'
        AND choiceNumber = 1
```

b)
```
WITH R AS ( SELECT  programme, COUNT(applicant) AS numApplicants
            FROM    AppliesFor
            WHERE   choiceNumber = 1 )
SELECT Programme
FROM   R
WHERE  numApplicants = ( SELECT  MAX(numApplicants)
                         FROM    R )
```

c)
```
WITH R1 AS
    ( SELECT  A.applicant AS name
      FROM    AppliesFor A JOIN AppliesFor B ON A.applicant = B.applicant
      WHERE   A.programme = 'MPALG'
              AND B.programme = 'MPCSN'
              AND A.choiceNumber < B.choiceNumber )
WITH R2 AS
    ( SELECT  name
      FROM    Applicants
      WHERE   'MPALG' IN (
                  SELECT  programme
                  FROM    AppliesFor
                  WHERE   applicant = name )
              AND 'MPCSN' NOT IN (
                  SELECT  programme
                  FROM    AppliesFor
                  WHERE   applicant = name )
SELECT  COUNT(name)
FROM    R1 UNION R2
```

Question 6. 5 p

a) See the lecture slides on transactions. In short phantoms can occur when (i) transaction A reads data satisfying some <search conditions>, then (ii) transaction B creates data items satisfying A's <search conditions>, then A repeats a read with the same <search conditions>.

b) In the normal case, T_5 returns a value one larger than T_2 (if place is accepted) or the same as T_2 (if place is declined).

Larger values for T_5 can occur due to phantoms (see part (a)) for transactions run with isoltion levels REPEATABLE READ, READ COMMITTED or READ UNCOMMITTED.

Running transactions with isolation level SERIALIZABLE is the only way to avoid possible problems with phantoms. But step T_4 involves waiting for a reply from the applicant, and it would be unacceptable for other transactions to have to wait.

Question 7. 8 p

a)
```
<!DOCTYPE Question7 [

<!ELEMENT Question7 (Applicants, Choices)>

<!ELEMENT Applicants (Applicant*)>
  <!ELEMENT Applicant EMPTY>
    <!ATTLIST Applicant
      name   CDATA #REQUIRED
      appNum ID    #REQUIRED >

<!ELEMENT Choices (Choice*)>
  <!ELEMENT Choice EMPTY>
    <!ATTLIST Choice
      applicant  IDREF #REQUIRED
      code       CDATA #REQUIRED
      choiceNum  CDATA #REQUIRED
      meritScore CDATA #REQUIRED>

]>
```

b)
```
//Choice[@choiceNum="1" and @meritScore>800]
```

c)
```
<Question7>
  <Applicant appNum="a1" name="Andersson">
    <Choice meritScore="750" choiceNum="1" code="MPSOF"/>
    <Choice meritScore="750" choiceNum="2" code="MPALG"/>
    <Choice meritScore="800" choiceNum="3" code="MPCSN"/>
  </Applicant>
  <Applicant appNum="a2" name="Jonsson">
    <Choice meritScore="700" choiceNum="1" code="MPALG"/>
  </Applicant>
  <Applicant appNum="a3" name="Larsson">
    <Choice meritScore="850" choiceNum="1" code="MPCSN"/>
    <Choice meritScore="850" choiceNum="2" code="MPALG"/>
  </Applicant>
</Question7>
```

d)
```
<Question7>
  {
    let $d := doc("exam.xml")
    for $a in $d//Applicant
    let $choices := (
        for $c in $d//Choices/Choice[@applicant = $a/@appNum]
        return <Choice code="{$c/@code}"
                       choiceNum="{$c/@choiceNum}"
                       meritScore="{$c/@meritScore}" /> )
    return <Applicant name="{$a/@name}" appNum="{$a/@appNum}" >
             {$choices}
           </Applicant>
  }
</Question7>
```