CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

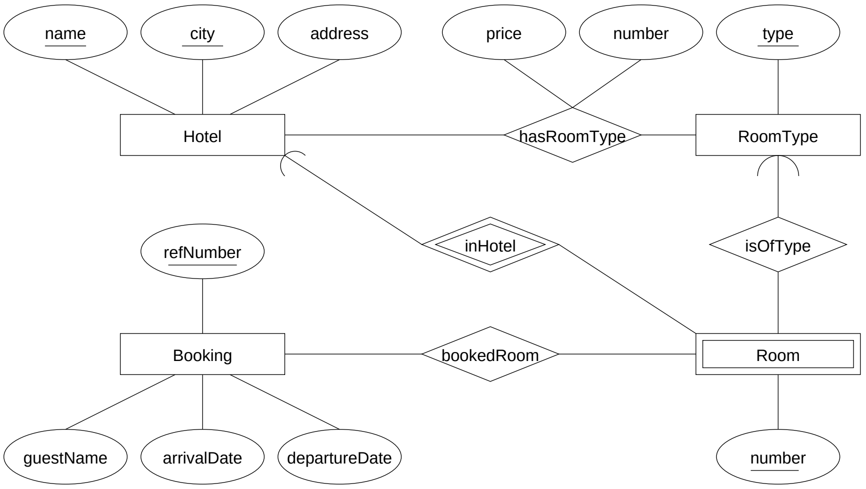
Examination in Databases, TDA357/DIT620

Friday 19 December 2008, 14:00-18:00

Solutions

Updated 2011-12-09

Question 1. a) E-R diagram:
10 p



b) $Hotels(\underline{name}, \underline{city}, address)$

$RoomTypes(\underline{type})$

$HasRoomType(\underline{name}, \underline{city}, \underline{roomType}, price, number)$
    $(name, city) \rightarrow Hotels.(name, city)$
    $roomType \rightarrow HasRoomType.type$

$Rooms(\underline{name}, \underline{city}, \underline{number}, roomType)$
    $(name, city) \rightarrow Hotels.(name, city)$
    $roomType \rightarrow HasRoomType.type$

$Bookings(\underline{refNumber}, guestName, arrivalDate, departureDate)$

$BookedRooms(\underline{name}, \underline{city}, \underline{number}, \underline{refNumber})$
    $(name, city, number) \rightarrow Rooms.(name, city, number)$
    $refNumber \rightarrow Bookings.refNumber$

Question 2. a) i) After considering the closures of all subsets of attributes, we find the following additional non-trivial FDs:
10 p

$D \rightarrow C$
$AB \rightarrow C$
$AD \rightarrow B$
$AD \rightarrow C$
$BD \rightarrow C$
$CD \rightarrow B$
$ABD \rightarrow C$
$ACD \rightarrow B$

Superkeys are: AD, ABD, ACD, ABCD. There is one key: AD.
FDs that violate BCNF:

$B \rightarrow C$
$D \rightarrow B$
$D \rightarrow C$
$AB \rightarrow C$
$BD \rightarrow C$
$CD \rightarrow B$

ii) — By first decomposing on $B \rightarrow C$, we first get $R_1(B, C)$ and $R_2(A, B, D)$. R_2 is not in BCNF, so we must decompose further.

— By first decomposing on $D \rightarrow B$, we first get $R_1(B, C, D)$ and $R_2(A, D)$. R_1 is not in BCNF, so we must decompose further.

In both cases, we end up with three relations, $R_a(A, D)$, $R_b(B, C)$ and $R_c(B, D)$.

b) FD:

$$room, day, hour \rightarrow courseCode$$

MVDs:

$$courseCode \twoheadrightarrow room, day, hour$$
$$courseCode \twoheadrightarrow student$$

Question 3. a) $\tau_{health\_centre}(\pi_{health\_centre}($
$Doctors \bowtie_{person\_number=doctor} (\sigma_{patient=\text{“}6006064444''ANDyear\geq 2000}(Appointments))))$

4 p

b) $\gamma_{health\_centre,month,count(*)\rightarrow numApps}($
$Doctors \bowtie_{person\_number=doctor} (\sigma_{year=2007}(Appointments)))$

Question 4. a)

8 p

```
SELECT   name, points
FROM     Student NATURAL LEFT OUTER JOIN (
             SELECT student AS id, points
             FROM   Points
             WHERE code = 'TDA357' )
ORDER BY name;
```

b) i)

```
WITH DBStudents AS (
    SELECT id, name, points
    FROM   Students JOIN Points ON id = student
    WHERE  code = 'TDA357' AND year = 2007 AND month = 'December' )
SELECT name
FROM   DBStudents
WHERE  points = ( SELECT MAX(points) FROM DBStudents );
```

ii)

```
WITH DBStudents AS (
    SELECT id, name, points
    FROM   Students JOIN Points ON id = student
    WHERE  code = 'TDA357' AND year = 2007 AND month = 'December' )
SELECT name
FROM   DBStudents
WHERE  points >= ALL ( SELECT points FROM DBStudents );
```

c)

```
CREATE VIEW V AS
  SELECT   course, month, year, AVG(points) AS avgPoints
  FROM     Points
  GROUP BY course, month, year
  HAVING   COUNT(student) > 100;
```

Question 5. a) i) *Flight(flightNumber, day, month, year, numSeats, price)*
10 p

Passengers(passengerId, name, address)

Booking(bookingReference, flightNumber, day, month, year, passenger)

$flightNumber \rightarrow Flight.flightNumber$

$(flightNumber, day, month, year) \rightarrow Flight.(flightNumber, day, month, year)$

Solutions that make different assumptions about the keys and foreign keys might be accepted.

```
CREATE TABLE Flight (
    flightNumber     VARCHAR(8),
    day              INT,
    month            INT,
    year             INT,
    numSeats         INT CHECK (numSeats BETWEEN 50 and 200),
    price            INT DEFAULT 2000,
    PRIMARY KEY (flightNumber, day, month, year)
);

CREATE TABLE Passengers (
    passengerId      VARCHAR(20) PRIMARY KEY,
    name             VARCHAR(30),
    address          VARCHAR(50)
);

CREATE TABLE Booking (
    bookingReference VARCHAR(20) PRIMARY KEY,
    flightNumber     VARCHAR(8),
    day              INT,
    month            INT,
    year             INT,
    passenger        VARCHAR(20),
    FOREIGN KEY (flightNumber, day, month, year)
        REFERENCES Flight(flightNumber, day, month, year)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY passenger REFERENCES Passengers.passengerId
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

ii) Here are some suggestions, but some other policies will be accepted if these are well motivated.

If a passenger's ID changes, then we want to change the passenger ID also in that passenger's bookings. If a passenger is deleted from the database, then we might want to delete all of that passenger's bookings.

Regarding the references between bookings and flights, what we want to happen on update will probably depend on which part of the flight's key changes. For example, if a new flight number is assigned to the flight, then it would seem reasonable to cascade that change to all bookings for the flight. However, if the day changes, then we might want to SET NULL, and inform the passenger that they should contact the airline. If the flight is deleted, we could simply delete all bookings for that flight. However, the airline might prefer to SET NULL in the Booking relation, until passengers can be informed and possibly be assigned to other flights.

b)

```
CREATE ASSERTION NotOverbooked CHECK
  ( NOT EXISTS (
        SELECT flightNumber
        FROM   Flight F
        WHERE  numSeats < (
                   SELECT COUNT(bookingReference)
                   FROM   Booking B
                   WHERE  B.flightNumber = F.flightNumber
                      AND B.day = F.day
                      AND B.month = F.month
                      AND B.year = F.Year )
  ) );
```

c)

```
CREATE TRIGGER SetPriceOfRemainingSeats
AFTER INSERT ON Booking
REFERENCING NEW ROW AS new
FOR EACH ROW
WHEN
  ( ( ( SELECT COUNT( bookingReference)
        FROM   Booking
        WHERE  flightNumber = new.flightNumber
           AND day = new.day
           AND month = new.month
           AND year = new.year ) -
      ( SELECT numSeats
        FROM   Flight
        WHERE  flightNumber = new.flightNumber
           AND day = new.day
           AND month = new.month
           AND year = new.year ) ) > 20 )
BEGIN
    UPDATE Flight
    SET    price = 4000
    WHERE  flightNumber = new.flightNumber
       AND day = new.day
       AND month = new.month
       AND year = new.year;
END;
```

Question 6.
4 p

a) Only T1 updates the database, and accounts A001 and A002 will end up with balances of 9000 and 21000, respectively, regardless of the order in which the steps are executed. However, the amount printed by transaction T2 will depend on the order in which steps $T1_B$, $T1_D$, $T2_A$ and $T2_B$ are carried out.

Orderings [$T1_B$, $T1_D$, $T2_A$, $T2_B$], [$T2_A$, $T2_B$, $T1_B$, $T1_D$], [$T1_B$, $T2_A$, $T1_D$, $T2_B$] and [$T2_A$, $T1_B$, $T2_B$, $T1_D$] all give the correct amount of 30000.

However, [$T2_A$, $T1_B$, $T1_D$, $T2_B$] gives 31000 and [$T1_B$, $T2_A$, $T2_B$, $T1_D$] gives 29000.

b) Possible outcomes are that T2 will print 30000 (orderings [$T1_B$, $T1_D$, $T2_A$, $T2_B$] and [$T2_A$, $T2_B$, $T1_B$, $T1_D$]) or 31000 (ordering [$T2_A$, $T1_B$, $T1_D$, $T2_B$]).

Question 7.
6 p

a)
i) task 1: 2, task 2: 10, task 3: 10.
ii) task 1: 4, task 2: 3, task 3: 10.
iii) task 1: 4, task 2: 10, task 3: 4.
iv) task 1: 6, task 2: 3, task 3: 4.

b) (i) and (iii) have cost of 5.2 and (ii) and (iv) have cost 5.0. There's not much difference between the four options, but according to the calculations either only an index on *course*, or both indexes, would be best.

Question 8.
8 p

a)

```
for $g in /LabReports/GroupMembers/GroupMember,
    $l in /LabReports/LabsPassed/LabPassed[@lab="2"]
where $l/[@group] = $g/[@group]
return <Result>{$g/student}</Result>
```

b) Here is one suggestion. Other solutions might be accepted.

```
<LabReports>
 <Group groupnumber="20">
  <Student>101</Student>
  <Student>102</Student>
  <LabPassed>1</LabPassed>
  <LabPassed>2</LabPassed>
 </Group>
 <Group groupnumber="21">
  <Student>103</LabPassed>
  <LabPassed>1</LabPassed>
 </Group>
 <Group groupnumber="22">
  <Student>104</LabPassed>
  <LabPassed>2</LabPassed>
 </Group>
</LabReports>
```

c)

```
<!ELEMENT LabReports (Group*)>
<!ELEMENT Group (Student+, LabPassed*)>
<!ELEMENT Student (#PCDATA)>
<!ELEMENT LabPassed (#PCDATA)>
```

d)

```
/LabReports/Group[LabPassed=2]
```