Examination in Databases (TDA357/DIT620)

20 March 2015 at 8:30-12:30, Hörsalsvägen 5

CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Department of Computer Science and Engineering

Examiner: Aarne Ranta tel. 772 10 82. The examiner will visit the exam room at 9:30 and 11:30.

Results: Will be published 10 April at the latest.

Exam review: In April after the Easter break. To be announced at course web page and in the Google group.

Grades: Chalmers: 24 for 3, 36 for 4, 48 for 5. GU: 24 for G, 42 for VG.

Help material: One “cheat sheet”, which is a 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. One English language dictionary is also allowed.

Specific instructions:

- 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 (numbers 1 to 6) on a new page (the a,b,c,... parts with the same number can be on the same 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.
- In particular: in SQL questions, use standard SQL or Oracle. If you use any other variant (such as MySQL), say this; but full points are not guaranteed.
- Write the question number on every page. If you need many pages for one question, number the pages as well, for instance, “Question 3 p. 2”.

Most questions in this exam will be about one and the same domain: flights. This domain makes heavy use of databases, familiar from web applications. The exam will go through different aspects having to do with airlines, flight connections, and bookings.

Table 1. We start with a table listing some flights. It is an example of a relation that links together flight codes, airlines, airports, and aircraft. The airlines and airports are real ones, whereas the codes are fake.

flight code	airline	prime flight	operating airline	departure city	departure airport	destination city	destination airport	aircraft type	seats
SK111	SAS	SK111	SAS	Gothenburg	GOT	Frankfurt	FRA	B737	140
LH555	Lufthansa	SK111	SAS	Gothenburg	GOT	Frankfurt	FRA	B737	140
AF111	Air France	AF111	Air France	Gothenburg	GOT	Paris	CDG	A320	170
LH111	Lufthansa	LH111	Lufthansa	Frankfurt	FRA	Paris	CDG	A321	200
LH222	Lufthansa	LH222	Lufthansa	Frankfurt	FRA	Malta	MLA	A320	170
AF222	Air France	AF222	Air France	Paris	ORY	Malta	MLA	A320	170
AB222	Air Berlin	AB222	Air Berlin	Frankfurt	FRA	Munich	MUC	A320	170
KM111	Air Malta	KM111	Air Malta	Munich	MUC	Malta	MLA	A319	140
LH333	Lufthansa	KM111	Air Malta	Munich	MUC	Malta	MLA	A319	140
SK222	SAS	KM111	Air Malta	Munich	MUC	Malta	MLA	A319	140
AF333	Air France	AF333	Air France	Paris	CDG	Frankfurt	FRA	A320	170

We assume the following (slightly simplified) conventions for this domain:

- the “flight code” attribute determines all other attributes on a row
- the “prime flight” is the flight code used by the airline operating the flight; the “flight code” in the first column can thus belong to another airline that has a code sharing agreement with the operating airline
- the “prime flight” appears in the table as a “flight code” as well, having itself as prime flight
- each airport has a unique code
- every aircraft of the same type has the same number of seats

(It is a common practice that one and the same flight can be booked using different airlines. Each airline uses a different "flight code", but the passengers end up in the same plane. The code used by the actual operating airline is called the "prime flight" code. For example, whether you book flight LH333 with Lufthansa or flight SK222 with SAS, you end up in the plane of Air Malta flight KM111.)

Question 1. Modelling and Design, basic (11p)

Question 1a (2p)
Find at least four redundancies in Table 1.

Question 1b (5p)
Draw an Entity-Relationship diagram that models the data in Table 1 in a meaningful way. The diagram must have some separate entities and relationships. Mark the keys by underlining them.

Question 1c (4p)
Convert your Entity-Relationship (E-R) diagram to a database schema. Mark all keys and references.

Question 2. Modelling and Design, advanced (9p)

This question refers to the same domain as Question 1.

Question 2a (3p)
Identify the functional dependencies and keys in the domain as described in Question 1. You must have some functional dependencies that are not superkeys. Consider the entire Table 1 as one relation. For functional dependencies, it is enough to list a base (a minimal set that implies all the others).

Question 2b (4p)
Starting with Table 1 and the functional dependencies and keys in (2a), decompose the relation into BCNF (Boyce-Codd Normal Form). Show all intermediate steps. **Notice**: if you find out that the relation is already in BCNF, then you have done something wrong in (2a).

Question 2c (2p)
Suppose you know the attributes of a relation and that it has no functional dependencies.

- Do you have enough information to bring it to BCNF. If yes, how? If no, why?
- Do you have enough information to bring it to the Fourth Normal Form (4NF). If yes, how? If no, why?

Question 3. Construction and Querying, basic (17p)

Question 3a (4p)
Convert the schema you designed in Question 1c to SQL. Make sure you use reasonable datatypes and express all referential constraints properly.

Questions 3b-3d

In the questions that follow, we need information about departure and arrival times. We assume the following relations have been implemented as tables in SQL:

Airports(code,city)

FlightCodes(code, airlineName)

Flights(departureAirport, destinationAirport, departureTime, arrivalTime, code)
departureAirport -> Airports.code
destinationAirport -> Airports.code
code -> FlightCodes.code

The listed flight code is the prime flight (i.e. the one used by the operating airline). For simplicity, we assume that departureTime and arrivalTime are integers denoting full hours, all in the same time zone, and that

$0 \leq \text{departureTime} < \text{arrivalTime} < 24$

Question 3b (3p)
Using this schema (not the one you defined in 1c), write an SQL query that finds all airports that have departures or arrivals (or both) of flights operated by Lufthansa or SAS (or both).

Question 3c (3p)
Using the schema from (3b), write an SQL query that shows the names of all cities together with the number of flights that depart from them, and sorts them by the number of flights in descending order (i.e. the city with the largest number of departures first).

Question 3d (7p)
Using the above schema (same as in 3b and 3c), write a view that lists all connections from any city X to any other city Y involving 1 or 2 legs (i.e. separate flights between two cities: if you fly from Gothenburg to Paris with a change in Frankfurt, the connection has two legs).

The query must return a table with the following information (and nothing else):

- the departure city X and destination city Y
- the departure time from X and the arrival time in Y
- the number of legs
- the total time from departure in X to arrival in Y
- the total time spent in the air

A change is possible if and only if (1) it happens at the same airport (2) the changing time is at least 1 hour, and (3) the connecting flight is on the same day.

Question 4. Construction and Querying, advanced (6p)

Note: In these questions, we don't care about the set/multiset distinction.

Question 4a (3p)
Express the query of question in (3b) by a relational algebra expression.

Question 4b (3p)
Translate the following relational algebra expression to an SQL query:

$$\pi_{\text{First.depatureTime,Second.arrivalTime}}$$
$$((\varrho_{\text{First}}(\text{Flights})) \bowtie_{\text{First.destinationAirport = Second.departureAirport}} (\varrho_{\text{Second}}(\text{Flights})))$$

Question 5. Triggers, basic (9p)

Assume the following two tables are added to the schema in (3b): a table listing for each flight the number of available seats and the price per ticket,

```
AvailableFlights(flight, numberOfFreeSeats, price)
  flight -> Flights.code
```

another one listing the passengers that have been booked for each flight, with the price they have paid and a unique booking reference number (an integer, to keep things simple),

```
Bookings(reference, flight, passenger, price),
  flight -> Flights.code
```

Question 5a (2p)
Create a view that lists booking references, passengers, flight codes, and departure and destination cities.

Question 5b (7p)
Create a trigger on the view defined in (5a). This trigger takes care of booking a new passenger to a flight. It is fired by an insertion of a passenger and a flight code to the view, for instance, “book Annie Adams for AF666”. Its effect should be the following:

- if the number of free seats on AF666 is positive, decrement it by one; the booking is successful
- if there are no free seats, the booking fails
- if the booking is successful, add Annie Adams and AF666 to Bookings, with the price given in AvailableFlights when booking her; also add a booking reference which is the maximum of the previous references (for all flights) plus one
- if the booking is successful, increment the price by 50 SEK for the next passenger (thus the fuller the flight, the more you pay)

Question 6. Miscellaneous, advanced (8p)

Question 6a (4p)
Bookings are typically made concurrently by many users. Assuming that the booking steps in (5b) are *not* an atomic transaction, show sequences of events (i.e. queries and updates from different customers) where

- a customer is suggested a flight but it turns out to be full
- a customer is told that a flight is full although it has seats
- a customer has to pay overprice i.e. a price that applies to customers booking later

For each sequence, also indicate which isolation level would prevent the undesired outcome from happening.

Question 6b (4p)
Write an XML document that contains a DTD corresponding to the database schema in (3b) together with data used on the first and the last row in the table of Table 1 (where you can invent the departure and arrival times yourself).

Solutions to Examination in Databases (TDA357/DIT620)

20 March 2015 at 8:30-12:30, Hörsalsvägen 5

CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG,
Department of Computer Science and Engineering
Teacher: Aarne Ranta

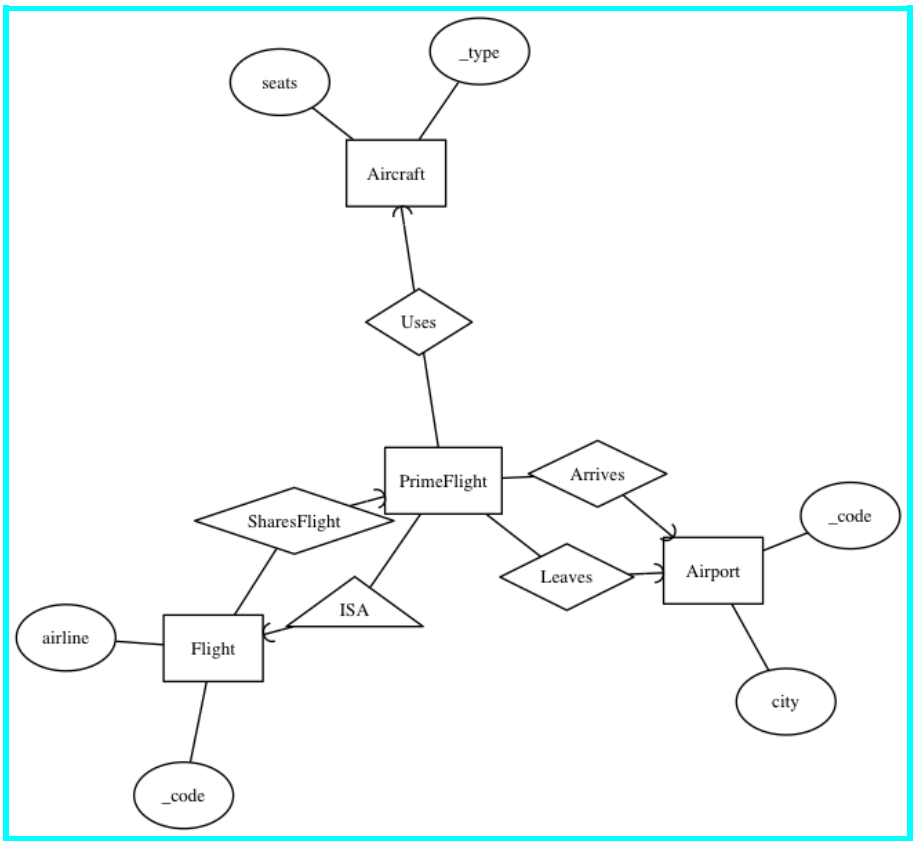
The answers are written in handwriting font on cyan background, and all other text is explanations. Thus the minimal answer can be seen from the cyan text.

Question 1a: four redundancies in Table 1.

1. number of seats repeated for each occurrence of aircraft
2. city repeated for each departure airport code
3. city repeated for each destination airport code
4. airline, cities, and aircraft of prime flight repeated for each flight code
5. airline is redundant if the flight code is given, because it can be computed from the two-letter prefix of the code

0p for the wrong idea about what a redundancy is

Question 1b: Entity-Relationship diagram for data in Table 1. There are several other correct answers. The picture is enough as an answer.



-2 for the wrong design of Primary flight
-1 for wrong types of relationships
-2 for suggesting storing lot of redundant data (could be same -2 as wrong design prime flight)

Question 1c: convert your Entity-Relationship (E-R) diagram to a database schema.

Airports(_code,city)

Aircraft(_type,seats)

Flights(_code,airline,primeFlight)
  primeFlight -> PrimeFlights.code

PrimeFlights(_code,depAirport,destAirport,aircraft)
  code -> Flights.code
  depAirport -> Airports.code
  destAirport -> Airports.code
  aircraft -> Aircraft.type

-3 if all relationships are translated incorrectly

Question 2a
functional dependencies

- flightCode -> (all attributes) (enough to say airline and primeFlight)
- departureAirport -> departureCity
- destinationAirport -> destinationCity
- aircraft -> seats
- primeFlight -> all attributes except flightcode and airline

optionally also:

- primeFlight airline -> (all attributes) (enough to write flightCode)
- primeFlight -> operatingAirline

keys:

- flightCode

optionally also (assuming another company uses just one code for sharing a flight)

- primeFlight, airline

The optional dependencies and keys are not required in the answer, but compensate for possibly missing other ones.
Keys missing: -1
-2 for giving FDs for their own schema instead of the given table
-1 for incorrectly calculated closure

Question 2b BCNF

R1(_aircraftType, seats)

R2 (_destinationAirport, destinationCity)

R3(_departureAirport, departureCity)

primeFlight -> operatingAirline brings another relation if considered

R4(_flightCode, airline, primeFlight, operatingAirline, departureAirport, destinationAirport, aircraftType)
departureAirport -> R3.departureAirport
destinationAirport -> R2.destinationAirport
aircraftType -> R1.aircraftType

0 if some of the violating FDs are not acted on
0 if acted on a FD that does not violate BCNF

Question 2c: a relation that has no functional dependencies.
Enough information to bring it to BCNF?

Yes. It is already in BCNF, so we need not do anything.

Enough information to bring it to 4NF?

No. There can be multivalued dependencies that are not functional dependencies.

0 if vague speculations instead of yes or no

Question 3a The schema of Question 1c in SQL.

```
CREATE TABLE Airports (
  code CHAR(3) PRIMARY KEY,
  name VARCHAR(32)
  ) ;
CREATE TABLE Aircraft(
  type VARCHAR(16) PRIMARY KEY,
  seats INT
  ) ;
CREATE TABLE PrimeFlights(
  code VARCHAR(8) PRIMARY KEY,
  depAirport CHAR(3) REFERENCES Airports(code),
  destAirport CHAR(3) REFERENCES Airports(code),
  aircraft VARCHAR(16) REFERENCES Aircraft(type)
  ) ;
CREATE TABLE Flights(
  code VARCHAR(8) PRIMARY KEY,
  airline VARCHAR(32),
  primeFlight VARCHAR(8) REFERENCES PrimeFlights(code)
  ) ;
```

optionally add this:

```
ALTER TABLE PrimeFlights ADD CONSTRAINT codeExists FOREIGN KEY (code)
REFERENCES Flights(code) ;
```

-2p if (most of) primary keys are missing
0 if introduces new tables

Question 3b: an SQL query that finds all airports that have departures or arrivals (or both) of flights operated by Lufthansa or SAS (or both).

```
SELECT DISTINCT
  served
FROM
  ((SELECT destinationAirport AS served,airlineName
    FROM FlightCodes JOIN Flights ON Flights.code = FlightCodes.code)
  UNION
  (SELECT departureAirport AS served,airlineName
    FROM FlightCodes JOIN Flights ON Flights.code = FlightCodes.code)
  ) AS D
WHERE D.airlineName = 'Lufthansa' OR D.airlineName = 'SAS' ;
```

-2p if the result is not an actual list of airports but e.g. all attributes.
-1 if the list has duplicates
There are many variants, for instance unions on other levels.

Question 3c: an SQL query that shows the names of all cities together with the number of flights that depart from them, and sorts them by the number of flights in descending order

```
SELECT Airports.city, COUNT(Flights.code) AS nflights
FROM Airports JOIN Flights ON Airports.code = Flights.departureAirport
GROUP BY Airports.city
ORDER BY nflights DESC ;
```

This answer does not list cities with 0 departing flights, but this is OK.
-1 if forgot DESC
-1 if SUM instead of COUNT
-2 if forgot GROUP BY

Question 3d: a view that lists all connections from any city X to any other city Y involving 1 or 2 legs.

```
CREATE VIEW Connections AS
WITH
  CityFlights AS
  (SELECT departureAirport, D.city AS departureCity, destinationAirport,
          A.city  AS destinationCity, departureTime, arrivalTime
  FROM Flights, Airports AS D, Airports AS A
  WHERE D.code = departureAirport AND A.code = destinationAirport)
SELECT departure, arrival, totalTime, airTime, legs
FROM
  ((SELECT departureCity AS departure, destinationCity AS arrival,
         arrivalTime - departureTime AS totalTime,
         arrivalTime - departureTime AS airTime,
         1 AS legs
   FROM CityFlights)
  UNION
  (SELECT F1.departureCity AS departure, F2.destinationCity AS arrival,
         F2.arrivalTime - F1.departureTime AS totalTime,
         (F2.arrivalTime - F1.departureTime) - (F2.departureTime - F1.arrivalTime)
as airTime,
         2 AS legs
   FROM CityFlights F1 JOIN CityFlights F2 ON F1.destinationAirport =
F2.departureAirport
   WHERE F1.arrivalTime < F2.departureTime
  )) AS F ;
```

The last “AS F” is required in SQL because subqueries in FROM must have aliases. But we do not require this.
Small errors:

- show airports instead of cities: -1
- show 2 or 3 legs instead of 1 or 2: -1
- miscomputing airTime: -1
- only 1 leg: 2
- lack of comparison between arrivalTime and departureTime: -2

Question 4a: express 3b in relational algebra

$$\pi_{served}\ (\sigma_{D.airlineName = \text{'Lufthansa'}\ OR\ D.airlineName = \text{'SAS'}}\ (\varrho_{D}\ (\pi_{served,\ airlineName}\ (\varrho_{(served/destinationAirport)}\ (FlightCodes \bowtie_{Flights.code\ =\ FlightCodes.code} Flights)) \cup \pi_{served,\ airlineName}\ (\varrho_{(served/departureAirport)}\ (FlightCodes \bowtie_{Flights.code\ =\ FlightCodes.code} Flights))$$

The subscript (b/a) to ϱ means that the attribute a is renamed to b whereas the other attributes keep their old name. Variant notations for this renaming will be accepted if we understand them.

Question 4b: translate an algebra expression to SQL

```
SELECT First.depTime, Second.arrivalTime
FROM Flights AS First JOIN Flights AS Second
ON First.destAirport = Second.depAirport ;
```

-1 if translated as SELECT * FROM
- no deduction if doing renaming via WITH First AS SELECT * FROM Flights

The printed exam pdf had a slightly different-looking symbol for ϱ. This may have been a problem for some participants, but not many. The single-name subscript to ϱ means that the whole relation is renamed.
Variations:

- use cartesian rather than JOIN: OK

Question 5a View with booking references, passengers, flight codes, and departure and destination cities.

```
CREATE VIEW Passengers AS
SELECT B.reference, B.passenger, B.flight, D.city as dep, A.city as dest
FROM
  Bookings B JOIN Flights F ON F.code = B.flight,
  Airports D JOIN Flights G ON G.departureAirport = D.code,
  Airports A JOIN Flights H ON H.destinationAirport = A.code
  ;
```

-2 if something of the form SELECT … departure FROM… UNION SELECT … destination FROM ….
-1 for airports instead of cities

Question 5b Trigger for booking a flight

```
CREATE TRIGGER makeBooking
INSTEAD OF INSERT ON Passengers
REFERENCING NEW AS new
  FOR EACH ROW
  DECLARE
    nseats INT;
    price INT;
    maxRef INT;
  BEGIN
    SELECT numberOfFreeSeats INTO nseats FROM AvailableFlights
        WHERE flight = :new.flightcode;
    IF nseats > 0 THEN
      UPDATE AvailableFlights
          SET numberOfFreeSeats = numberOfFreeSeats - 1
          WHERE flight = :new.flightcode;
      SELECT price INTO price FROM AvailableFlights
          WHERE flight = :new.flightcode;
      SELECT MAX(reference) INTO maxRef FROM Bookings;
      INSERT INTO Bookings (reference, flight, passenger, price)
          VALUES  (maxRef + 1, :new.flightcode, :new.passenger, price);
      UPDATE AvailableFlights SET price = price + 50
          WHERE flight = :new.flightcode;
    ELSE
      RAISE_APPLICATION_ERROR(-20001, 'No seats available');
    END IF;
  END ;
```

We are not picky about the syntax when grading, but focus on checking that all the correct actions are performed.

BEFORE instead of INSTEAD OF: -1
-1 if forgot to increase the price
-1 if used COUNT instead of MAX for reference

Question 6a: concurrency problems with isolation levels solving them.

A customer is suggested a flight but it turns out to be full

Customer 1: read

Database: freeseats=1 freeseats=0

Customer 2: read book commit

- nonrepeatable read, avoided by REPEATABLE READ

A customer is told that a flight is full although it has seats

Customer 1: read

Database: freeseats=1 freeseats=0

Customer 2: read book rollback

- dirty read, avoided by READ COMMITTED

A customer has to pay overprice i.e. a price that applies to customers booking later

Customer 1: read book commit

Database: price=1000 price=1050

Customer 2: read book commit

- nonrepeatable read, avoided by REPEATABLE READ

The last one was left out from grading.

-3 if situations are described correctly but no isolation levels given

Question 6b: XML with DTD for the schema in 3b with the first and the last flights in Table 1. There are many solutions; this is the one implemented in qconv.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE FlightData [
 <!ELEMENT FlightData (Airports | FlightCodes | Flights)*>
 <!ELEMENT Airports EMPTY>
 <!ATTLIST Airports
    code ID #REQUIRED
    city CDATA #REQUIRED
    >
 <!ELEMENT FlightCodes EMPTY>
 <!ATTLIST FlightCodes
    code ID #REQUIRED
    airlineName CDATA #REQUIRED
    >
 <!ELEMENT Flights EMPTY>
 <!ATTLIST Flights
    departureAirport IDREF #REQUIRED
    destinationAirport IDREF #REQUIRED
    departureTime CDATA #REQUIRED
    arrivalTime CDATA #REQUIRED
    code IDREF #REQUIRED
    >
]>
<FlightData>
 <Airports code="CGD" city="Paris" />
 <Airports code="FRA" city="Frankfurt" />
 <Airports code="GOT" city="Gothenburg" />
 <FlightCodes code="AF333" airlineName="Air France" />
 <FlightCodes code="SK111" airlineName="SAS" />
 <Flights departureAirport="CDG" destinationAirport="FRA" departureTime="6"
arrivalTime="8" code="AF333" />
 <Flights departureAirport="GOT" destinationAirport="FRA" departureTime="6"
arrivalTime="7" code="SK111" />
</FlightData>
```

Not using ID / IDREF correctly: -1
Missing DTD: one can get max 1
No root element: -1