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

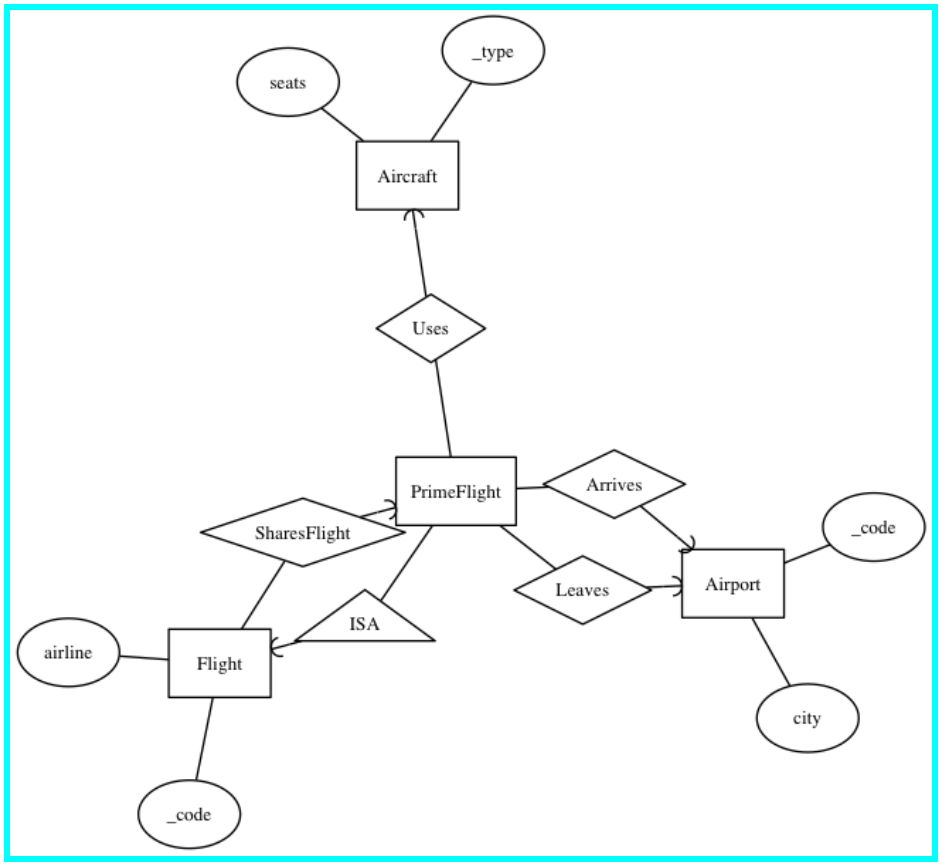
The answers are written in handwriting font on cyan background, and all other text is explanations. Thus the minimal answer can be seen from the cyan text.

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

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

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