Data Storage and Formats

IT University of Copenhagen

January 4, 2011

(This is a translated version of the Danish-language exam. Answers may be given in Danish or English.)

The exam consists of 6 problems with 15 questions in total. It has 11 pages (including this page). **It is recommended to read the problems in order**, but it is not important that you solve them in any specific order.

The weight of each problem is stated. You have 4 hours to answer all questions. If you cannot give a complete answer to a question, try to give a partial answer.

All pages in your submission have to be numbered and include your name, social security number and the code of the course (BDLF). Only write on the front of each sheet and make sure to sort them so the problems will appear in order.

"KBL" refers to the course book "Database Systems - an application approach, 2nd edition", by Michael Kifer, Arthur Bernstein and Philip M. Lewis.

All written aids are allowed.

1 Data modeling (25%)

A database containing information about flight departures needs to be developed. The goal is for it to support a number of queries not supported by traditional booking systems. Instead of a specification of the data that the system should contain, and examples of the queries the system should support are provided below. You need *not* explain how to concretely implement the queries.

Connection queries:

- Find all direct flights from A to B departing between time t_1 and t_2.
- Find all direct flights from A to B arriving between time t_1 and t_2.
- Find all connections from A to B, involving exactly 1 stopover of maximum 3 hours.
- Find the fastest connection between A and B (total traveltime).

Specified connection queries:

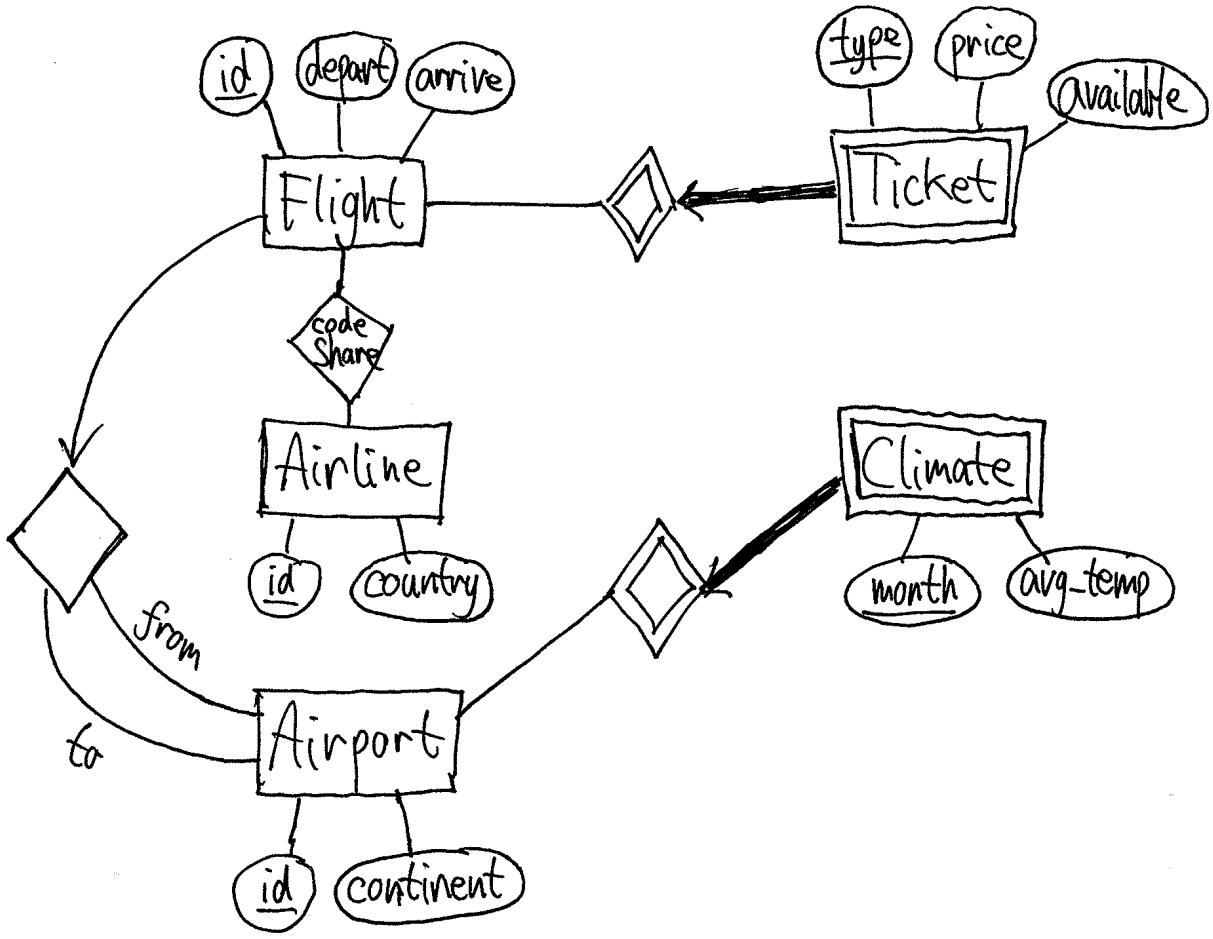
- Find all flights from A to a place where the average temperature in the given month is at least 25 degrees.
- Find all flights from A to Africa (or any continent of your choice) arriving between t_1 and t_2.
- Find one direct connection from A to B with an airline from USA (or any country of your choice). A flight can be associated with more than one airline ("code share").

Ticket queries:

- Find available ticket types and their prices for a flight
- Find the cheapest available ticket type for a given connection query.
- Find the cheapest available business class ticket for a given connection query.

a) Create an E-R model for a database containing enough information to answer all of the above queries. If you make any assumptions about data that has not been described in this problem, you have to describe them. Use the same E-R notation as KBL. Your model should express as many properties about the data as possible, such as participation constraints and a normalized databasedesign. Data not needed for the above (e.g., detailed information about bookings) should not appear in the model.

Example answer:

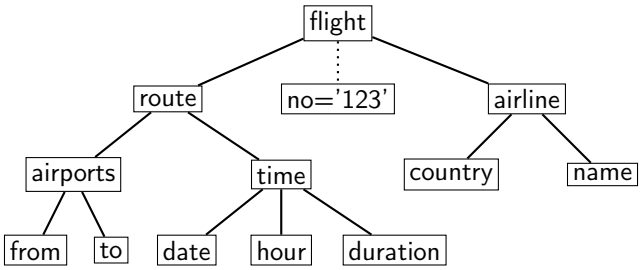


b) Create a relational datamodel that represents the E-R model. For each relation you need to state attributes, primary keys (underline) and foreign keys (dashed underline).

Example answer:
`Airport(id,continent)`
`Flight(id,from_id,depart,to_id,arrive)`
`Codeshare(flight_id,airline_id)`
`Airline(id,country)`
`Climate(airport_id,month,avg_temperature)`
`Ticket(flight_id,type,price,available)`
All attributes with a name ending with `_id` are foreign keys.

2 XML (25%)

XML documents can be represented as trees. In this problem we look at a tree representing an XML document without text nodes. Following the notation in KBL a dashed line indicates an attribute node.



a) Give a textual representation of the XML document. It is of importance that it is well-formed. You do not need to indicate processing elements $<?\cdots>$.

Example answer:

```
<flight no='123'>
   <route>
      <airports><from/><to/></airports>
      <time><date/><hour/><duration/></time>
   </route>
   <airline><country/><name/></airline>
</flight>
```

XPath expressions returns a set of XML elements. In the above document, each name only exists in one element. These sets can therefore be indicated as sets of names, like: {date,time}.

b) Specify the sets returned by the following XPath expressions:

1. `flight/route`
2. `flight/*`
3. `*/route/*`
4. `//time[name]`
5. `time/../*`
6. `//country/*/..`

Example answer:

1. {route}.
2. {route, airline}.
3. {airports, time}.

4. {}.

5. {}.

6. {}.

c) Create an XML Schema that defines a language that includes the above XML document. It should allow text elements in the from, to, date, hour, duration, country and name elements. There can be more than one airline element in the flight element. You do not need to have a country element under airline. Furthermore the time element can appear before airports. Apart from these instructions the structure of the schema should be as the example. You do *not* need to specify specific datatypes for text elements or use namespaces.

Example answer:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:f="http://flightinfo.org"
  targetNamespace="http://flightinfo.org">

  <element name="flight">
    <complexType>
      <sequence>
        <element ref="f:route"/>
        <element ref="f:airline" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="no" type="string"/>
    </complexType>
  </element>

  <element name="route">
    <complexType>
      <all>
        <element ref="f:airports"/>
        <element ref="f:time"/>
      </all>
    </complexType>
  </element>

  <element name="airline">
    <complexType>
      <sequence>
        <element name="country" type="string"/>
```

```
        <element name="name" minOccurs="0" type="string"/>
      </sequence>
    </complexType>
  </element>

  <element name="airports">
    <complexType>
      <sequence>
        <element name="from" type="string"/>
        <element name="to" type="string"/>
      </sequence>
    </complexType>
  </element>

  <element name="time">
    <complexType>
      <sequence>
        <element name="date" type="string"/>
        <element name="hour" type="string"/>
        <element name="duration" type="string"/>
      </sequence>
    </complexType>
  </element>

</schema>
```

In the last question we look at an XML document of the form:

```
<flightinfo>
   <flight no='123'>... </flight>
   <flight no='456'>... </flight>
   ...
</flightinfo>
```

In XQuery the document is refered to as `doc("flightinfo.xml")`. We assume that it is possible to change from one flight to another, if and only if: 1) The dates for the departures are identical and 2) The end time (`hour + duration`) of the first flight is between 1 and 4 hours from the start time (`hour`) of the next flight.

d) Write XQuery expressions that return:

1. A list of all numbers (`no`) on flights from CPH to FRA.
2. A list of all airports that SAS flies to or from. (Duplicates are OK.)
3. A list of all pairs of airports where you can get from one to the other with exactly one stop-over. Use the format: `<conn><from>CPH</from><to>EWR</to></conn>`.

Example answer:

1. `doc("flightinfo.xml")//airports[from="CPH" and to="FRA"]/../../@no/string()`
2. `doc("flightinfo.xml")//airline[name='SAS']/../route/airports/*/string()`
3.
```
for $a in doc("flightinfo.xml")//flight
    for $b in doc("flightinfo.xml")//flight
      let $aa := $a/route/airports
      let $ba := $b/route/airports
      let $at := $a/route/time
      let $bt := $b/route/time
      let $transfer := $bt/hour - ($at/hour + $at/duration)
      where ($aa/to=$ba/from) and ($at/date=$bt/date) and
            ($transfer>=1) and ($transfer<=4)
      return <conn>{($aa/from, $ba/to)}</conn>
```

3 SQL (20 %)

Consider the following relations containing information on the inspections of airplanes:

```
Inspection(iid,planeId,date)
Employee(eid,name,title,employedSince)
InspectedBy(employeeId,inspectionId)
```

Plane(pid,airline,modelId,buildDate)
Model(mid,name,manufacturer,engineType)

Attributes in primary keys are underlined. In the following we assume that no foreign keys have been assigned in the database.

a) Write an SQL query that returns pid on all SAS airplanes built after 2000-01-01 (buildDate>'2000-01-01').

Example answer:

```
SELECT pid FROM Plane WHERE airline="SAS" and buildDate>'2000-01-01'
```

b) Wirte an SQL query that calculates the number of employees who have inspected an SAS airplane (airline='SAS').

Example answer:

```
SELECT COUNT(DISTINCT IB.employeeId)
FROM Inspection I, InspectedBy IB, Plane P
WHERE I.iid=IB.inspectionId and I.planeId=Plane.pid and Plane.airline='SAS'
```

c) Write one or more SQL commands that will delete all information on inspections before 2000-01-01.

Example answer:

```
DELETE FROM Inspection WHERE date<"2000-01-01";
DELETE FROM InspectedBy WHERE inspectionId NOT IN Inspection
```

d) Write an SQL query that returns the names of the employees who have only worked with airplanes from one manufacturer (manufacturer).

Example answer:

```
SELECT name FROM Employee E
WHERE 1=
  (SELECT COUNT(DISTINCT M.manufacturer)
   FROM Inspection I, InspectedBy IB, Plane P, Model M
   WHERE E.eid=IB.employeeId and I.iid=IB.inspectionId
         and I.planeId=P.pid and P.modelId=M.mid)
```

4 Normalization (10%)

Consider the following relations from problem 3:

```
Plane(pid,airline,modelId,buildDate)
Model(mid,name,manufacturer,engineType)
```

`pid` and `mid` are primary keys of the relations. Furthermore `modelId` is a foreign key that references `Model(mid)`. Now assume that each manufacturer `manufacturer`) only builds one plane per date (`buildDate`).

a)
Argue that {`modelId, buildDate`} is a key in `Plane`.

Example answer:
Each manufacturer belongs to exactly one value of `modelId`. In connection with `buildDate` they uniquely identify an airplane.

Someone is considering to merge the relations into one (by a join operation):

```
Plane2(pid,airline,buildDate,modelId,modelName,manufacturer,engineType)
```

b) Argue that this relation is not in BCNF. Your argumentation should use the concept of functional dependencies (FDs).

Example answer:
According to question a) we have the following FD:

`modelId buildDate` $\rightarrow$ `pid, airline`.

But {`modelId, buildDate`} is not a key for `Plane2`, since different producers can build planes on the same day.

5 Relational algebra (10 %)

Consider the relations from problem 3 once again.

a) Using words, describe briefly what the following expression will return:

$$\pi_{\text{title}}(\sigma_{\text{date}>\text{'2010-12-31'}}(\pi_{\text{iid,date}}(\text{Inspection})) \underset{\text{iid=inspectionId}}{\bowtie} \text{InspectedBy} \underset{\text{eid=employeeId}}{\bowtie} \text{Employee})$$

Example answer: Titles of the inspectors who have made an inspection after 2010-12-31.

b) Write an expression in relational algebra corresponding to the following SQL query:

```
SELECT airline, manufacturer
FROM Plane, Model
WHERE mid = pid and name = '747'
```

Example answer: $\pi_{\text{airline,manufacturer}}(\text{Plane} \underset{\text{mid=pid}}{\bowtie} \sigma_{\text{name='747'}}(\text{Model}))$

6 Indexing (10%)

Consider the following SQL Queries:

1. `SELECT COUNT(*) FROM Employee WHERE title='apprentice' and employedSince<'2008-01-01'`
2. `SELECT * FROM Employee WHERE name>title`
3. `SELECT name FROM Employee WHERE eid=42`
4. `SELECT * FROM Employee, InspectedBy WHERE title='apprentice' and eid=employeeId`
5. `SELECT DISTINCT name,title FROM Employee`

A colleague is considering the following (B-tree) indexes:

(a) `Employee(name)`

(b) `Employee(title,employedSince)`

(c) `Employee(employedSince,title)`

(d) `Employee(name,eid,title)`

(e) `Employee(eid,name)`

(f) `Employee(eid,title)`

a) For each of the SQL queries, indicate which of the above indexes (if any) you would assume to be used in a fast query plan. Underline the index you believe to result in the best performance. (All answers that could be correct within reason will be accepted). As an example, write "`0. a,b`" if you think index `a` and `b` can be used to conduct query `0` faster, and that `b` is the better one of the two.

Example answer:

1. `b,c`
2. No B-tree index can help.
3. `e,f`
4. `e,f`
5. None of the given indexes help.