Database Systems

IT University of Copenhagen

January 2, 2007

This exam consists of 5 problems with a total of 15 questions. The weight of each problem is stated. You have 4 hours to answer all questions. The complete assignment consists of 6 numbered pages (including this page).

If you cannot give a complete answer to a question, try to give a partial answer. You may choose to write your answer in Danish or English. Write only on the front of sheets, and remember to write your CPR-number on each page. Please start your answer to each question at the top of a *new* page. Please order and number the pages before handing in.

RG refers to *Database Management Systems* by Raghu Ramakrishnan and Johannes Gehrke, McGraw-Hill, 2002.

All written aids are allowed / Alle skriftlige hjælpemidler er tilladt.

1 SQL DDL and normalization (20%)

The national railroad company in the Republic of Delalaya maintains a relational database of information on (physical) trains, train personnel, and manning. The database stores key numbers for each train, and information on which employees work on a given train departure. To simplify administrative procedures, employees always work on the same physical train, and always work all the way from the departure station to the final destination. The database has the following schema:

```
Train(Tid,type,productionYear,capacity)
Personnel(Pid,worksOnTrain,worksAs,hiredDate,salary)
Manning(Tid,Pid,Sid,onDate)
```

The attribute `type` of `Train` refers to the train manufacturer's code for a specific kind of trains. All trains with the same code are identical. The attribute `worksOnTrain` of `Personnel` is a foreign key reference to `Train(Tid)`. This reflects the fact that each person always works on the same train (but of course not all the time). Any tuple in `Personnel` must contain a valid reference to a tuple in `Train`. The `Sid` attribute of `Manning` is a reference to a code for a specific departure in the time table. However, the time table is not part of the database. The attribute pair `(Pid,Tid)` in `Manning` is a foreign key reference to `Personnel(Pid,worksOnTrain)`.

a) Write SQL statements that create the above relations, including key and foreign key constraints. You can make any reasonable assumption about the data types.

b) Identify all functional dependencies in the relations that do not have a superkey on the left hand side (i.e., are "avoidable"). Use the dependencies to decompose the schema into BCNF, and state the resulting database schema. Briefly discuss the quality of the new schema (ignoring efficiency issues).

2 Data modeling (30%)

The transportation authority in the Republic of Delalaya has decided to implement a database to keep statistics on public transportation (fuel-driven buses and electricity-driven trains), with emphasis on keeping track of delays and the number of travelers. In particular, it should be used to identify particular weaknesses in the transportation systems (e.g., stretches of railroad that often cause delays, or employees who have trouble keeping the schedule).

First of all, data from the railroad company, as described in Problem 1, should be integrated into the system, but not necessarily using the schema stated there.

a) Draw an ER diagram corresponding to the data described in Problem 1, including, if possible, all integrity constraints stated there. You should follow general rules for making a good ER design — your design does *not* need to translate to the three relations stated in Problem 1.

In addition to the above, the following information is needed:

- **Information on buses.** Similar to the information on trains, but in addition the *range* of each bus (the number of kilometers it will drive on a full tank) should be recorded.
- **Information on bus drivers.** Similar to to the information on train personnel, with the following changes: All bus drivers have the same work (driving the bus). A bus driver does not drive a specific bus, but may drive any bus.
- **Route information.** The sequence of stops on each train/bus route. Each route has a unique *route number*.
- **Vehicle usage.** For each route and departure time, on each day, record which physical train/bus was used. (In Delalaya, this is always a single bus or train – several trains can't be coupled together.)
- **Timetable information.** Information on *planned* arrival and departure times for each route and every stop.
- **Timetable statistics.** Information on *actual* arrival and departure times for every train/bus and every stop, on every day.
- **Traveler statistics.** Periodically, surveys are being made that record the destinations of all travelers in a given bus/train at a given time (between two stops).
- **Manning.** Who has worked on a particular vehicle at every time. It should be taken into account that manning may change at any time during a route.

b) Draw an ER diagram that integrates the additional data described above with the ER diagram of Problem 2 a). You should follow general rules for making a good ER design. If you need to make any assumptions, state them.

c) Suppose that we desire the database to evolve over time (e.g. with new time tables), but we also want to be able to store and query historical data. Outline how the ER diagram of Problem 2 b) could be changed to achieve this.

3 SQL (25 %)

Consider a database of flight departures and airplanes, with the following schema:

```
Departure(departureID,airplaneID,destination,departureTime,bookedSeats)
Airplane(airplaneID,modelID,fabricationYear)
Model(ModelID,name,capacity)
```

a) Write an SQL query that lists the `airplaneID` of all airplanes made before 1960.

b) Write an SQL query that lists the `departureID` for all departures bounded for destinations starting with the letter “D”.

c) Write an SQL query that lists the average `capacity` of airplanes fabricated 1970 or later.

d) Write an SQL query that lists the `departureID` for every overbooked departure (i.e. where the number of bookings exceed the capacity of the plane).

e) Write a query in **relational algebra** that lists the model-`name` of every airplane that was fabricated in 1970.

f) Write an SQL query that lists the `fabricationYear` of the oldest and second oldest plane. You may assume that the two oldest planes have different values on the attribute `fabricationYear`.
(**Note:** Some versions of SQL have a special syntax for this kind of query – however, you are required to write the query using only SQL features found in RG.)

g) Write an SQL query that lists all `destinations` that has more empty seats than the average number of empty seat on all departures (we assume that the number of empty seats is the number of booked seats subtracted from the capacity of the plane).

4 Transactions (10 %)

a) Consider an initially empty table `T` with the schema `T(number)` and transactions running at isolation level `READ COMMITTED`

	Transaction 1	Transaction 2
1		INSERT INTO t VALUES (1)
2	INSERT INTO t VALUES (2)	
3		SELECT * FROM T
4	SELECT * FROM T	
5		ROLLBACK
6	SELECT * FROM T	
7		SELECT * FROM T

State what is returned from each of the SELECT queries at line 3, 4, 6, and 7. If there are several possibilities, you may state any of them.

b) Consider the two schedules below.

Schedule 1:

Transaction 1	Transaction 2
R(A)	
	R(A)
R(B)	
	W(B)
rollback	
	W(A)
	commit

Schedule 2:

Transaction 1	Transaction 2
	R(B)
R(B)	
R(A)	
W(A)	
commit	
	W(B)
	commit

State for each of the two schedules whether it is serializable or not. If the schedule is serializable write a serialization of the schedule, otherwise give a brief explanation of why the schedule is not serializable.

5 Database efficiency (15%)

Consider the relation `T(id,name)` with 100,000 tuples. Values of `id` are positive integers, and values of `name` are strings of length at most 30. The following queries are of interest:

1. `SELECT * FROM t WHERE id = 100`
2. `SELECT * FROM t WHERE id > 10`
3. `SELECT * FROM T WHERE id > 100 and id < 9000 and name = 'Mads'`

a) State for each of the above queries whether an index would speed it up. In the cases where the answer is "yes" you also have to specify:

- Would a Hash-index or a B-tree index be the fastest index?
- Which attribute(s) should be indexed?
- Would a clustered index make the query significantly faster than an unclustered index?