COMP9120 Database Management Systems

Tutorial Week 8 Solution: Schema Refinement and Normalisation

Exercise 1. Interpreting Functional Dependencies

Consider the following schema regarding the transport of important passengers directly from a specified pickup point at the airport entrance to the appropriate departure gate for their flights:

VipTransfers(destination,departs,airline,gate,name,contact,pickup)

A tuple such as ('Berlin', '11:25 01/06/2012', 'Lufthansa', 3, 'Justin Thyme', '0413456789', 1) means that Justin Thyme has a flight to Berlin at 11:25 on June 1, 2012, with Lufthansa Airlines, departing from Gate 3, and must be taken there from pickup point 1, and Justin can be contacted by phone on his number 0413456789. The schema has the following functional dependencies:

I. destination, departs, airline → gate

II. gate → airline

III. contact → name

IV. name, departs → gate, pickup

V. gate, departs → destination

a) Express the above functional dependencies in simple English.

I. An airline never runs more than one flight departing at the same time to the same destination. Each flight departs from a single gate.

II. All flights from a gate are provided by the same airline.

III. Each phone is owned by at most one VIP.

IV. A VIP cannot depart from two gates at the same time, and cannot be picked up from more than one location for a departure.

V. Flights can depart to at most one destination from a gate at a given time.

b) Consider the following collection of tuples. Why is this instance not a legal state for the database?

Destination	Departs	Airline	Gate	Name	Contact	Pickup
Berlin	1/06/2012 11:25	Lufthansa	3	Justin Thyme	0416594563	1
Madrid	1/07/2012 14:30	Iberian	4	Willy Makit	0497699256	2
London	3/05/2012 6:10	British Airways	7	Hugo First	0433574387	5
Moscow	1/07/2012 17:50	Aeroflot	6	Rick OhChet	0416594563	7
Berlin	1/06/2012 11:25	Qantas	1	Dick Taite	0469254233	4
Kuala Lumpur	1/08/2012 14:30	Cathay	7	Hugo First	0433574387	2
Singapore	1/08/2012 14:30	Qantas	2	Hugo First	0433574387	2
London	1/07/2012 17:50	Lufthansa	3	Justin Thyme	0413456789	4

British Airways and Cathay can't share the same gate 7.

Rick can't use Justin's phone number (but it's fine that Justin has 2 numbers).

Hugo can't make two flights at the same time on 1/8/12. Maybe he changed flights and the old flight didn't get removed.

Exercise 2. Candidate keys and Normal Forms

Before answering the questions below, I'll lead you through the basic steps to analyse the problem. Instead of going through the process below to find out if (contact, departs, airline) is a key for question 2(a), you could just find the closure of these attributes and of subsets of these attributes as shown in the lecture, but the process below will hopefully help you see a bigger picture.

First of all, you cannot determine the normal form of a relation without knowing all the keys. Recall that for the purposes of schema normalization we make no distinction between candidate keys and primary keys. There are simply keys to a relation.

All keys must have full attribute closure (i.e., all attributes of the relation must follow from knowing the key) and it must be minimal (i.e., you can't get full attribute closure from any subset of the proposed key; more than the minimal set of keys is just a super key – not a candidate key).

Look back at the functional dependencies. Some attributes are determined by others, because they appear on the right of a FD. Other attributes only appear on the left side of the FDs, in which case they must be known directly rather than inferred from others, and so must be part of a key. => To help you recall later – we will call this statement Z.

The only attributes of the original relation that never appear on the right of an FD are *departs* and *contact*, so these must be part of all keys.

So start with (*departs*, *contact*). What is its closure (*departs*, *contact*)$^+$? The following steps grow the set of implied attributes by incorporating the FDs listed at the start.

Implied attributes	**Updated closure**	**Comment**
departs, contact	*{departs, contact}*	Trivial attributes
Name	*{departs, name, contact}*	From *contact* via FD III
gate, pickup	*{departs, gate, name, contact, pickup}*	From *name*, *departs* via FD IV
Destination	*{destination, departs, gate, name, contact, pickup}*	From *gate*, *departs* via FD V
Airline	*{destination, departs, airline, gate, name, contact, pickup}*	From *gate* via FD II

So we get the full set of attributes from the closure of (*departs*, *contact*). Is it minimal? No FD involves just *departs*, so its closure (*departs*)$^+$ is just {*departs*}. You should be able to work out that the closure (*contact*)$^+$ = {*contact* ,*name* }. So no subset of (*departs*, *contact*) is a key, meaning (*departs*, *contact*) **is** a key. You should be able to see also that since these attributes must appear in all keys (see statement Z), this is the **only** key for the relation.

a) Is (contact, departs, airline) a candidate key from the above functional dependencies. Can you find an alternative?

Almost. The closure of the attributes is the full relation, but a candidate key must also be minimal. The inclusion of *airline* means it is not a candidate key, however it is a superkey.

b) Is the relation in 3NF?

gate → airline: does NOT meet the 3NF restriction that either the LHS is a superkey or at least for the RHS to be part of a key. (hence cannot be in 3NF)

c) Explain whether it is a lossless-join decomposition to decompose the relation into the following:

R1(destination, departs, gate)

R2(contact, departs, pickup)

R3(gate, airline)

R4(contact, name)

No. It is not a lossless-join decomposition:

Recall, when using the lecture decomposition process to decompose a relation into 2 smaller relations based on an FD, where one of the smaller relations has all attributes of the FD.

R3 and R4 correspond to two such smaller relations that match FDs II and III respectively.

Once R3 and R4 are removed from the original relation, this leaves a relation with all attributes of the original relation minus the RHS of FDs II and III ie: R7 (*dest, dep, gate, contact, pickup).* If we decompose R7 into R1 and R2 shown in this question, this is NOT a lossless-join decomposition since the common attributes of R1 and R2 (*departs)* is NOT a superkey for either R1 and R2 (lecture slides mention this requirement). (see closure of departs to check that it is not a superkey of R1 or R2).

d) Give a lossless-join decomposition of the original relation into BCNF relations.

Our original relation is:

R(destination, departs, airline, gate, name, contact, pickup).

Suppose we have decomposed R into

R3(gate, airline),

R4(contact, name)

R7(destination, departs, gate, contact, pickup)

as discussed in Exercise 2(c), which is a loss-less decomposition. R3 and R4 are in BNCF, while R7 is not. The FDs that are now hold for R7 are

contact, departs → gate, pickup

gate, departs → destination

You can see that (contact, departs) is a key of R7, while (gate, departs) is not. Thus, we decompose R7, based on the FD gate, departs → destination, into

R5(destination, departs, gate)

R6(departs, gate, contact, pickup)

Now, all R3, R4, R5, and R6 are in BCNF, the decomposition terminates.

Exercise 3. Relation Decomposition

Download the vip_transfers.sql file from Canvas which contains a schema for the above relation, along with example data. Execute the statements within this file. You can insert projections of this original relation into decomposed relations. For instance, the relation Example(contact, name) can be created as follows:

```
CREATE TABLE Example (
   contact VARCHAR(10),
   name VARCHAR(30)
);
INSERT INTO Example SELECT DISTINCT contact, name FROM VipTransfers;
COMMIT;
```

For your proposed decomposition, try creating the decomposed relations and a query to reconstruct the whole relation. Is the query result identical to the original relation?

Create relations for R3, R4, R5, and R6, populate them with data corresponding to that in the vip_transfers.sql file, then try to run this join to see if you can recover exactly the original relation:
select * from ((R3 natural join R4) natural join R5) natural join r6;
Solution file is available in Canvas