**COMP9120 Database Management Systems** **Semester 2, 2016**

# Tutorial Week 7 Example Solution: Design Theory 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. Each flight departs from a single gate. An airline never runs more than one flight departing at the same time to the same destination.

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 though 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. Conversely, *pickup* is only ever on the right of any FD, so it is never going to be part of a key.

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) What normal form is the relation in, and why? Looking at the restrictions imposed by 2NF, 3NF and BCNF on functional dependencies will help you decide.

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)

Also, since contact → name only includes part of the key on the left hand side (and name is a non-key attribute), it is a partial dependency, so the relation also cannot be in Second Normal Form (2NF). (see 2NF condition in lecture slides)

But each attribute is atomic, so it is 1NF.

c) Explain whether it be a good idea to decompose the relation into the following:

`R1(destination, departs, gate)`

`R2(gate, airline)`

`R3(contact, name)`

`R4(contact, departs, pickup)`

No. Firstly, 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.

R2 and R3 correspond to two such smaller relations that match FDs II and III respectively.

Once R2 and R3 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: R9 (*dest, dep, gate, contact, pickup).* If we decompose R9 into R1 and R4 shown in this question, this is NOT a lossless join decomposition since the common attributes of R1 and R4 (*departs)* is NOT a key for either R1 and R4 (lecture slides mention this requirement). (see closure of departs to check that it is not a key of R1 or R4).

Also, the decomposition does not preserve dependencies. See lecture slides: “All attributes of an FD must appear in a single relation.” This does not hold for FDI, IV.

d) Give a lossless-join, dependency-preserving decomposition of the original relation in the highest normal form relations possible. Is the decomposition 3NF or BCNF?

Our usual process for decomposition mentioned in the lecture (see Theorem on slide 29) can only guarantee the *lossless join* property, but now we must also consider if decomposition is *dependency preserving*. Hence, for this question, we must find decompositions that satisfy the conditions mentioned in the lecture for both:

(a) *Lossless join decomposition:* common attributes for 2 relations resulting from a decomposition contain a key for one of these relations.

*(b) Dependency preserving:* taking each FD “x” in turn, all properties in “x” must all be found in one (not across both) of the relations resulting from decomposition.

Our original relation is:

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

Trying to decompose along FD III using the theorem on slide 29 of the lecture would yield *R10(destination, departs, airline, gate, contact, pickup) and R11(contact, name).*This is a lossless join decomposition, but is not dependency preserving since FD IV is not preserved (see condition (b) above).

One lossless-join decomposition that is also dependency preserving is to decompose R into *R1(contact, name, departs)* and *R2(destination, departs, airline, gate, name, pickup).* It is dependency preserving, since "name" is now also in the larger relation, allowing dependency preservation to hold for all FDs. Further, the lossless join decomposition property holds since (name, departs) can determine all attributes of R2 (see discussion of attribute closure in lecture). We can further decompose R1 along FD III into *R3(contact, name)* and *R4(contact, departs),* maintaining properties for (a) and (b). R2 can also be further decomposed into *R5(name, departs, gate, pickup)* and *R6(destination, departs, airline, gate)* since the common properties are a key for R6, and (b) can also still be observed to hold. Our final relations *are* R3, R4, R5, and R6. Of these, R3 and R5 have only one FD that applies to their attributes, and they meet the BCNF condition mentioned in the lecture. No FD applies to the attributes of R4, so that is also in BCNF.

The tricky relation is R6, whose attributes are covered by FD I, FD II and FD V. Performing an analysis using attribute closure, you will find that the keys of R6 are (*destination, departs, airline*) and (*departs, gate*). Hence every attribute is part of a key (all attributes are "prime attributes"), which means the relation is in Third Normal Form (3NF) – see 3NF condition in the lecture. Because the two keys overlap – they share the *departs* attribute – it is impossible to decompose further while still preserving dependencies. So although the other relations are in BCNF, the overall decomposition is in 3NF. R6 is not in BCNF since it does not meet the rule that for every X determines Y, X is a superkey (eg: Left hand side of FDII is gate, where {gate}+ is not a superkey of R6).

One may ask the question of why not decompose R2 along FD IV using the theorem in slide 29 of the lecture. The answer is that, unlike our decomposition into R5, and R6, this would not allow us to preserve FDI.

**Exercise 3. Relation Decomposition**

Download the `vip_transfers.sql` file, (from the Schemas page on eLearning), 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;