06-34140/34141/34139/36989 *Data Structures, Algorithms & Databases* The University of Birmingham
Spring Semester 2022-23 School of Computer Science

Solutions to Exercise Sheet 3

Question 1: Suggesting functional dependencies

a. Some possible dependencies:

owner_contact_phone	$\rightarrow$	owner
owner_contact_phone	$\rightarrow$	owner_address
registration_number	$\rightarrow$	model
registration_number	$\rightarrow$	year_of_first_registration
registration_number	$\rightarrow$	diesel_or_petrol
registration_number	$\rightarrow$	date_of_previous_MOT
engineer	$\rightarrow$	garage
garage	$\rightarrow$	garage_address
garage_MOT_licence	$\rightarrow$	garage

It has to be said that for most of these one can construct a counterexample, e.g., a garage may relocate during the time covered by the database.

b. The pair (registration_number, date_of_inspection) is a candidate key.

Question 2: Deriving functional dependencies

a. Does not imply $A \rightarrow D$ because A alone does not suffice to apply any of the given dependencies. Counterexample:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

b. Does imply $A \rightarrow D$:

i. $A \rightarrow A$ (trivially)
ii. $A \rightarrow B$ (assumption)
iii. $AB \rightarrow C$ (assumption)
iv. $A \rightarrow C$ (transitivity applied to (i), (ii), and (iii))
v. $BC \rightarrow D$ (assumption)
vi. $A \rightarrow D$ (transitivity applied to (iv), (ii), and (iv))

c. Does imply $A \rightarrow D$. Derivation is almost the same as for (b).

Question 3: Checking functional dependencies against a table

Because under C we have five different values, this attribute can functionally determine every other:

$$C \rightarrow A \qquad C \rightarrow B \qquad C \rightarrow D$$

No other single attribute determines anything because in each case one can find an example of two rows for which that attribute has the same value but the other has different ones. None of the pairs AB, BD, or AD determines anything. The only other possible dependency is

$$ABD \rightarrow C$$

Question 4: Working with functional dependencies

a. The minimal candidate keys are:

$$BCD,\ ACD,\ CDE$$

Taking BCD for example, we can see that BC determine E and ED determine A. So the other two attributes are determined by BCD. So, it is a candidate key.

To remove any single attribute from this collection, we would need to show that the other attributes determine the removed attribute. But there is no attribute that can be removed.

b. The table is evidently not in BCNF. The functional dependency $A \to B$, for example, would need that the left hand side A should be a superkey (superset of a candidate key). Since A is not a candidate key, this is not possible.

c. Let us choose $ED \to A$ as the harmful functional dependency to be removed from the table. The closure of ED, using the algorithm in Section 10, is $EDAB$. So, we can decompose the table T into two tables:

$$T_1(ABDE),\ T_2(CDE)$$

Now, the functional dependency $ED \to A$ exists only in T_1 and it is on a candidate key. We have fortuitously removed the other functional dependency $BC \to E$, because the combination BC does not occur eith either table.

The remaining functional dependency is $A \to B$. To remove it, we find the closure of A, which is just AB. So, we decompose T_1 into

$$T_{11}(AB),\ T_{12}(ADE)$$

So, T_{11}, T_{12}, T_2 is the final collection of tables obtained by normalisation procedure.

In the above, we chose to remove the functional dependency $ED \to A$ first. If we choose some other functional dependency to be removed first, we might obtain another decomposition. Two other possible decompositions arise in this way:

$$ADE,\ CDE,\ AB$$
$$BCE,\ AB,\ ACD$$

Note: The schema AB in the second decomposition was printed as AD in an earlier version of the solutions.

Question 5: Judging decompositions

a. AB together form a (candidate) key as together these attributes determine CD by assumption and themselves trivially.

   i. This is not lossless because the common attribute B is not a key for either table.

   ii. This is lossless because the common attributes BC are a key for the second table. We can assume that BC is not a key for the original schema, or otherwise we would have the functional dependency $BC \to A$. Therefore the decomposition is redundancy reducing. It is also dependency preserving because the functional dependency $AB \to D$ follows from those in the smaller tables by applying the transitivity rule from Armstrong's Axioms.

   The resulting tables are in Boyce-Codd Normal Form because the left hand sides of the valid functional dependencies are in fact keys for the tables in which they arise.

   iii. This is not lossless because the common attributes BD are not a key for either table.

   iv. This is lossless because BCD form a superkey of the second table. It is not redundancy reducing, though, because the first table is the same as the original one. For this stupid reason it is, of course, dependency preserving.

b. Only A is a candidate key.

   i. The decomposition of (A, B, C) into (A, B) and (B, C) is lossless. But the decomposition of (A, B, C, D) into (A, B, C) and (C, D, E) is not lossless.

   ii. This is lossless because we can reconstruct (A, B, C, D) by virtue of $B \to C$. Now we observe that by the transitivity rule from Armstrong's Axioms also $BD \to E$ must hold, and then the whole table can be reconstructed because the common attributes are B and D. However, it is not dependency preserving because we cannot reconstruct $CD \to E$ from the dependencies which hold in the smaller tables.

iii. This is lossless because (A, B, C, D) and (C, D, E) overlap in C and D, and (B, C) is a subtable of the first. It is also redundancy reducing but not to the maximal possible extent. It is dependency preserving because all functional dependencies are based on one of the small tables, except for $A \to E$ but this follows by transitivity. The result is not in Boyce-Codd Normal Form because (A, B, C, D) still contains the non-trivial functional dependency $B \to C$.

Question 6: Normalisation example

a. Using the normalisation procedure, we attempt to remove the functional depencies with IATAno as their left hand side. The closure of IATAno is the collection

   IATAno, name, maxscore

   So, we decompose the original table into two:

   testtype(<u>IATAno</u>, name, maxscore)
   test(<u>IATAno</u>, <u>airplane</u>, <u>date</u>, technician, duration, score)

   All the functional dependencies are now on primary keys. So, the tables are in BCNF. Note that "testtype" is quite a recognisable entity. It describes the valid types of tests that are conducted.

b. In the original table, the attributes "name" and "maxscore" would be duplicated over and over, representing data redundancy.

   Insertion anomalies would occur if a new record is inserted for a particular conduction of test, but the name or maxscore attribute are entered wrongly. In that case, different records would have different names associated with the same IATAno. Update anomalies might arise similarly.

   Aggregation anomaly is theoretically possible, but it is hard to see it arising in realistic applications.

   Deletion anomaly might occur if we archive old data. If a particular test was not performed in recent years, then its name and maxscore attributes would be deleted inadvertantly.

Question 7: Evaluating functional dependencies

a. **A** We might want to count how many customers have been served in a given period. In the given setup this could be tricky because one would have to recognise when two customers are actually the same. (Counting the credit card numbers instead is no perfect solution either because people use more than one card.)

   U If a person's address details change, say, then it would be quite awkward to update the information in the booking table, again because of the problem of identifying a given customer. Also, it is not clear whether one should update historical records. One would really like to have a separate customer table for this.

   D In order that the table does not get too big, one can imagine that records older than a month (say) are relegated to a "storage" table. If a customer hasn't made a booking for a month, his/her details would then get lost.

   I There is no problem when we enter a new customer as it is natural for him/her to provide this information when making the booking. If we enter a customer who has made a booking before but now uses a different card, then it is possible that his/her personal details are entered in a slightly different way (mis-spellings, for example). It will then again be difficult later to recognise them as one and the same person.

   It would really be much more natural if bookings and customer details were kept in separate tables. Also, from bad experience (even with reputable companies) perhaps a business should never store credit card details for longer than necessary for completing the transaction.

b. **A** I can't think of an interesting query that involves clock speed.

   U This could be a problem as cpu's get constantly upgraded and one would want the higher clock speed to be listed for each model that uses a particular processor.

   D This could be a problem if a particular processor is no longer used for any model but is later re-introduced, though the information would always be available from other sources.

   I I can't see a big problem here as the information is available when a model is specified.

   Decomposition is the right remedy because the tables will be small and re-joining for querying will be fast.

c. **A** Can't think of an aggregation query that would run into problems.

   U Definitely a problem as with every change in tuition fee one has to carefully apply the update query to all affected records. A lot of money is involved and people have no patience for mistakes in this area.

   D It is indeed possible that there are no students on a particular programme in some year (as with some MSc programmes) and the tuition fee information could get lost.

   I It's a problem as tuition fee is determined at a different time from when students are entered.

   Should be decomposed. Also, there is no performance issue which would require us to consider the other two solutions.

d. **A** If you wanted to compute the number of volumes published in a given year, then this table would not easily relinquish that information. But this may not be the most important question in life.

   U Not a problem as this information will never be updated (unless it was entered erroneously).

   D We will probably never delete a record from this table, though if we do, we would be able to reconstruct the information from the publishers website.

   I This can be a problem as the year of publication is determined later than the volume for a given article. (Publishers assemble papers, sometimes around a specific topic, but can't quite predict when the volume will be ready for publication.)

   I would not decompose.