Handout 3: Sample Problems on Dependencies and Normalisation

CS 4604

October 25, 2010

Problem 1 (From the 2005 midterm examination) Consider the relation

`Inventory(Manufacturer, Brandname, Type, Weight, Store)`.

This relation stores the items that a grocery store stocks. Each tuple in relation `Inventory` represents the fact that a store sells an item of a particular type and brand name manufactured by a particular company. The relation also stores the weight of the item. Two tuples that such a relation may contain are:

`(Kellogg's Company, Frosted Flakes, Cereal, 14oz., Hokies Holesome Foods)` and
`(Kraft Foods, Philadelphia, Cream Cheese, 8oz., Healthy Hokies Store)`.

Convert each of the next three sentences in English about `Inventory` into a functional or a multi-valued dependency. *Consider each of these three sentences independently.*

1. A manufacturer holds the trademark for a brand name of an item of a particular type, i.e., no two manufacturers can use the same brand name for items of the same type. For example, two different manufacturers cannot use the brand name `Philadelphia` for the food type `Cream Cheese`.

   Solution: Brandname Type → Manufacturer.

   I gave partial marks for dependencies that came close to this one, as I did in all the other parts of this question. Many students swapped `Brandname` and `Manufacturer`; the dependency Manufacturer Type → Brandname implies that every manufacturer makes only one brand of a particular type, e.g., Kellogg's Company makes only one cereal.

2. For each type, each store sells only one brand name made by each manufacturer. For example, `Hokies Holesome Foods` does not sell any `Cereal` other than `Frosted Flakes` that is manufactured by `Kellogg's Company`.

   Solution: Manufacturer Store Type → Brandname.

   Some students suggested Brandname Store Type → Manufacturer as a solution. This dependency follows from the solution to part (a). However, the English statement in part (b) does not follow from the English statement in part (a).

3. If a particular item (specified by its manufacturer, brand name, and type) is available in a particular weight at a store, then that weight is available at all stores carrying that food item.

   Solution:

   Brandname Manufacturer Type ↠ Weight and its companion Brandname ManufacturerType ↠ Store. The phrase "is available at all stores" suggest that this dependency is multi-valued. If we consider all tuples with a fixed value the manufacturer, brand name, and type of an item, the weight and store appear in all possible combinations in these tuples.

   Some students listed all five attributes in this multi-valued dependency, forgetting the fact that such a dependency is trivial.

Problem continues on the next page.

For the rest of this question, assume that all the functional and/or multi-valued dependencies you specified in the previous three parts hold in `Inventory`, as do any dependencies that follow from them. However, no other dependencies hold in `Inventory`.

4. What are the keys for `Inventory`?

   Solution:

   The two keys are {`Brandname, Store, Type, Weight`} and {`Manufacturer, Store, Type, Weight`}. The attribute `Weight` must appear in the key since it does not participate in any functional dependency. Any key that contains `Brandname` and `Type` need not contain the attribute `Manufacturer` but must contain the attribute `Store`. Any key that contains the attributes `Manufacturer`, `Store`, and `Type` need not contain the attribute `Brandname`.

5. What normal forms does `Inventory` satisfy?

   Solution: It is in 3NF but not in BCNF or in 4NF.

6. Consider the decomposition of `Inventory` into

   `Inventory1(Manufacturer, Brandname, Type, Store)` and

   `Inventory2(Manufacturer, Brandname, Type, Weight)`.

   Use the chase to show that this decomposition of `Inventory` is not lossless-join.

   Solution:

7. Modify one of the attributes in either `Inventory1` or in `Inventory2` to obtain a lossless-join decomposition. Use the chase to prove that this new decomposition is lossless-join.

   Solution:

Problem continues on the next page.

8. State all the normal forms that the decomposition in part 6 satisfies.

   Solution: `Inventory1` satisfies 3NF. `Inventory2` does not satisfy any normal form. The closure of the functional dependencies in part (a) and part (b) does not contain any new dependencies, other than trivial ones and ones that follow by augmentation. Therefore, we see that both these dependencies hold in `Inventory1` and the dependency of part (a) holds in `Inventory2`. The keys for `Inventory1` are {`Brandname, Store, Type`} and {`Manufacturer, Store, Type`} and the key for `Inventory2` is {`Brandname, Type Weight`}. Since `Brandname Type → Manufacturer` still holds in both `Inventory1` and `Inventory2`, neither of them is in BCNF (and, as a result, not in 4NF). However, both `Brandname` and `Manufacturer` are attributes in keys for `Inventory1`, so this relation is in 3NF.

   Most students did not consider each relation separately. Many students were mislead by the fact that it appeared that this decomposition was a result of using the multi-valued dependency in part (c); they thought that the relation was in 4NF. Some students did not use the fact that if a relation is in 4NF, then it is in BCNF, and that if a relation is in BCNF, then it is in 3NF.

9. Decompose `Inventory` into a set of relations that are in BCNF such that the decomposition is lossless join. Is this decomposition dependency-preserving?

   Solution:

10. Use the 3NF synthesis algorithm to compute a lossless-join dependency-preserving decomposition of `Inventory` into relations that are in 3NF. Are all the relations in BCNF?

    Solution: Do so even though `Inventory` is in 3NF, just to realise that even the 3NF synthesis algorithm cannot guarantee BCNF.

Problem 2 (From the 2006 midterm examination) Excited by what you are learning in CS 4604, you decide to create a database to track the songs your favourite band plays in its live concerts. Since you decide that E/R diagrams are for kids, you decide to create a relation schema directly for your database. After much consideration, you believe that a single schema will serve:

`Concerts(City, Venue, Year, Month, Date, Song, Album).`

In this relation, `City` (e.g., "Blacksburg") and `Venue` (e.g., "Cassell Colisseum") record where the concert took place and `Year`, `Month`, and `Date` keep track of when the concert took place. The idea is that these five attributes uniquely specify a concert. The attribute `Song` records the name of a song performed at a concert. You add the attribute `Album` to record which album the song belongs to. Perfect!

However, after using the database for a few months, you realise that your band (and the real world) have some characteristics that you should model in your database. Convert each of the next four sentences about `Concerts` into a functional or a multi-valued dependency. You can use the first letter of each attribute as an abbreviation for the attribute. *Consider each of these four sentences independently.* If you cannot write down a functional or a multi-valued dependency, say so, and explain why you cannot, if possible. Do not assume any other constraints, even if they seem reasonable to you.

1. Each song appears in at most one album. In other words, the band does not repeat the same song in different albums.

   Solution: Song → Album. This FD states that if two tuples have the same value for `Song`, they must have the same value for `Album`.

2. A city does not have two venues with the same name. In other words, `City` and `Venue` serve to identify the location of a concert uniquely.

   Solution: It is not possible to state any dependency reflecting this constraint.

3. In an effort to please its fans, the band plays at most one song from any album in a given concert.

   Solution: City Venue Year Month Date Album → Song. This FD states that if you fix the concert (using `City`, `Venue`, `Year`, `Month`, `Date`) and fix the `Album`, you can uniquely determine the `Song`.

4. The manager books the band in any city at most once every year.

   Solution: City Year → Venue Month Date. This FD states that if `Concerts` has two tuples with the same value for `City` and `Year`, they must have the same value for `Venue`, `Month`, and `Date`.

For the next two parts of this question, assume that all the functional and/or multi-valued dependencies you specified in the previous parts hold in `Concerts`, as do any dependencies that follow from them. However, no other dependencies hold in `Concerts`.

5. Use (i) the chase and (ii) Armstrong's axioms to derive the FD City Year Album → Song.

   Solution:

   (i) Chase:

   (ii) Armstrong's axioms: We know City Year → Venue Month Date. The completely trivial FD City Year → City Year is also true. Therefore, by using the combining rule, we obtain City Year → City Year Venue Month Date. Now, we can apply the augmentation rule to add the `Album` attribute to both sides to get City Year Album → City Year Album Venue Month Date. Now combining with City Venue Year Month Date Album → Song using the transitive rule, we obtain City Year Album → Song.

6. What are the keys for `Concerts`?

   Solution: The two keys are {City, Year, Song} and {City, Year, Album}. Here is the reasoning: Since City Year → Venue Month Date, `City` and `Year` can appear in the key. We have Song → Album as well as City Year Album → Song, so adding `Song` or `Album` to `City` and `Year` yields a key.

   It appears that `Venue`. `Month`, and `Date` could also appear in the key; however, they are already determined by `City` and `Year` together.

7. What normal forms does `Concerts` satisfy? **Solution:**

8. You realise that you must decompose `Concerts` into multiple relations. Here is a candidate decomposition into two relations:

   `Concerts1(City, Venue, Year, Month, Date)`

   `Concerts2(City, Year, Song, Album)`

   (i) For each relation `Concerts1` and `Concerts2`, state what normal forms it satisfies.
   Solution:

   Concerts1 This relation is in 3NF, BCNF, and 4NF.

   Concerts2 This relation is in 3NF. It is not in BCNF since the FDs Song $\rightarrow$ Album and City Year Album $\rightarrow$ Song both hold in it and the relation has the same keys as `Concerts`.

   (ii) Use the chase to determine whether the decomposition of `Concerts` into `Concerts1` and `Concerts2` is lossless-join.
   Solution:

   (iii) For each of the four FDs from the previous page, specify which relation (`Concerts1`, `Concerts2`, both, or neither) you can use to verify the FD.

   FD 1 **Solution:** `Concerts2` can verify Song $\rightarrow$ Album.

   FD 2 **Solution:** Not applicable, since there is no FD or MD.

   FD 3 **Solution:** This one was a little tricky. The FDs City Year $\rightarrow$ Venue Month Date and City Year Album $\rightarrow$ Song together imply the FD City Venue Year Month Date Album $\rightarrow$ Song in this question. We can verify City Year $\rightarrow$ Venue Month Date using `Concerts1` and City Year Album $\rightarrow$ Song using `Concerts2`

   FD 4 **Solution:** `Concerts1` can verify City Year $\rightarrow$ Venue Month Date.

Problem continues on the next page.

9. After a few years of using relations `Concerts1` and `Concerts2` and faithfully recording the band's performances in it, your realise that your beloved band plays a mean trick on its audience. In each city, all its concerts (spread over different years, of course) feature exactly the same songs! For example, the songs the band played in Blacksburg in 1996 are identical to the songs the band played in Blacksburg in 2000 and in 2004. Write down the dependencies that model this discovery and the names of the relations (`Concerts1` and/or `Concerts2`) in which these dependencies hold.

   Solution: Many students did not realise that the sentence implied an MD. If we fix the city to be `Blacksburg`, then all `Song/Album-Year` combinations must appear in the affected relation. The only relation involving these attributes is `Concerts2`. The *pair* of MDs are City $\twoheadrightarrow$ Song Album and City $\twoheadrightarrow$ Year. **ANSWER THE QUESTION: Why cannot we just say** City $\twoheadrightarrow$ Song **and argue that the** Album **attribute will appear on the right-hand side of the MD because of the FD** Song $\rightarrow$ Album**?**

10. Decompose the relations affected in the previous part into 4NF. Just apply one step of the 4NF decomposition algorithm.

    Solution: Decomposing `Concerts` based on the MDs in the previous part yields the relations

    ```
    Concerts3(City, Song, Album)
    Concerts4(City, Year)
    ```

11. Use the 3NF synthesis algorithm to decompose `Concerts` into a set of dependency-preserving relations each of which is in 3NF. Is each relation also in BCNF? If not, is there any way of decomposing the BCNF-violating relations so that the new relations are in BCNF, yet the overall decomposition is dependency-preserving?