University of California, Los Angeles
Department of Computer Science

Computer Science 143 **Prof. Ryan Rosario**

Homework 4

Solutions

Please remember the following:

1. Homework is mostly graded on completion. We may grade a few parts, but it will never be the majority of the grade on the assignment. So try your best, and focus on solving the problems. Consider homework (and studying the solutions) as practice for the final exam.
2. Homework must be submitted digitally, on CCLE. We will not do any paper grading. You can use a text file, but if you use Word, a PDF is preferred rather than a DOC file.
3. If there are any exercises that are difficult to do digitally (such as diagrams or math), consider scanning your drawing or math, or using a graphics program (even a readable MS Paint is fine) or Equation Editor.
4. **For the sanity of the grader** we will ask you to run the queries and submit the result. You may lose points if you only provide a query.
5. Solutions will be posted.

Special thanks to TA Jiaqi for contributing some of the solutions for Part 1.

Part 1: There's Nothing Wrong with Being Abnormal Unless you are a Relation

1. Suppose that we decompose the schema $R(A, B, C, D, E, F)$ into $R_1 = (A, B, C, F)$ and $R_2 = (A, D, E)$. Given the following functional dependencies hold, is the decomposition lossless? Explain your answer.

$$\mathcal{F} = \{A \to BC, CD \to E, B \to D, E \to A\}$$

Yes, the decomposition is lossless because the decomposed relations can be joined to form the original table *exactly*. It is lossy if *information* about the original table is lost, such as extra rows being created. More generally, this can be detected by checking if the intersection of two decomposed relations is a key for *at least one* of the two relations (key for both is OK too).

There are a variety of ways to determine if a decomposition is lossless. Based on lecture and the book, one way is to compute

$$R_1 \bowtie R_2$$

to see if it matches R. Perhaps a more precise way is also described in the book. If R is decomposed into R_1 and R_2 and F is the set of functional dependencies then R_1 and R_2 form a lossless decomposition of R is at least one of the following functional dependencies is in F^+:

(a) $R_1 \cap R_2 \to R_1$

(b) $R_1 \cap R_2 \to R_2$

These conditions mean that $R_1 \cap R_2$ form a superkey for either R_1 or R_2.

$$R_1 \cap R_2 = \{A, B, C, F\} \cap \{A, D, E\} = A$$

We immediately have a match since $E \to A$. Fortunately we do not even need to look in F^+. **The decomposition is lossless.** We can use *attribute* closure to find superkeys, if it is needed.

There is another method called the *chase algorithm* that also works, but seems to be more trouble than it is worth. You can read about the chase algorithm at `https://en.wikipedia.org/wiki/Chase_(algorithm)`.

2. List non-trivial functional dependencies satisfied by the following relation. You do not need to find *all* dependencies. In other words, please F, but there is no need to find F^+. It is enough to identify a set F of functional dependencies that imply all functional dependencies satisfied by this relation.

A	B	C
a_1	b_1	c_2
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Note that every $a_i \in A$ maps to exactly one $\beta_j in B$, **so one functional dependency is $A \to B$.** $A \to C$ does not hold because a_2 maps to both c_1 and c_3. $B \to C$ does not hold because b_1 maps to c_1 and c_3. $C \to A$ also holds.

The only nontrivial dependencies are $A \to B$ and $C \to A$. The original author used the phrase "nontrivial" differently than the technical definition. The original author also considers any function dependency that can be *derived from* these nontrivial dependencies to be trivial.

$BC \to A$ also holds, but it is not a trivial dependency because we can derive it from $C \to A$ first via augmentation to get $BC \to AC$ and then to the set $BC \to A$ and $BC \to C$ via decomposition, where $BC \to C$ is technically a trivial dependency since $C \in BC$.

In other words, you are asked to find F, but not F^+.

3. Assume the following set of functional dependencies hold for the relation $R(A, B, C, D, E)$:

$$F = \{A \to BC, CD \to E, B \to D, E \to A\}$$

Assume that key means *candidate key* if we do not specify otherwise.

(a) Is E a key for R? Explain your answer.

To determine if E is a key for R, we compute the closure of E, denoted E^+.

We first have $E \to A$, so $A \in E^+$. Next we have $A \to BC$ so $BC \in E^+$. So far, $E^+ = \{ABC\}$. But, $B \to D$, so $D \in E^+$. Finally, $CD \to E$ and $CD \in E^+$, so $E \in E^+$. Thus $E^+ = \{ABCDE\} = R$. Thus E is a key.

(b) Is BC a key for R? Explain your answer.

We compute the closure $(BC)^+$. By the augmentation rule, we have $BC \to DC$, so we have $CD \in (BC)^+$. By $CD \to E$, we have $E \in (BC)^+$. Then by $E \to A$, we have $A \in (BC)^+$. So our $(BC)^+ = \{ACDE\}$.

We decompose $A \to BC$ to $A \to B$ and $A \to C$. Since $A \in (BC)^+$, and $A \to B$, $B \in (BC)^+$.

Thus $(BC)^+ = \{ABCDE\} = R$.

BC is a key.

4. Assume the following set of functional dependencies hold for the relation $R(A, B, C, D, E, F) : \mathcal{F} = \{A \to BC, C \to E, B \to D\}$

Is R in Boyce-Codd Normal Form (BCNF)? Explain your answer. If it is not, normalize it into a set of relations in BCNF.

We determine if a R is in BCNF by considering all function dependencies in $\mathcal{F}$ and verifying that **each** of them is either

(a) trivial ($\beta \subseteq \alpha$, $\alpha \cup \beta = R$); or

(b) α is a superkey.

First we must determine the key. A looks like a good choice so let's compute A^+. By $A \to BC$ we get $BC \in A^+$. By $C \to E$ and $B \to D$ we get that $DE \in A^+$, so $A^+ = \{ABCDE\}$.

But wait a minute!

A can't be a key because it doesn't determine F! So what if we take AF to be the key? That's what we do. AF is a superkey because $(AF)^+ = R$.

Now, α, the lefthand side of each functional dependency, must be a superkey for R to be in BCNF. Right off the bat, A from $A \to BC$ is not a superkey, so $\boxed{R \text{ is not in BCNF.}}$

The order in which we decompose matters, in the sense that we may not get a dependency preserving decomposition. The problem only asks to decompose to BCNF though. This happens if we start with $A \to BC$. Consider $C \to E$ instead.

We decompose into

$$R_1 = (C, E), R_2 = (A, B, C, D, F)$$

Now we are done with $C \to E$. Then we use $B \to D$ to get

$$R_1 = (C, E), R_2 = (B, D), R_3 = (A, B, C, F)$$

Now we are done with $B \to D$. Now we deal with $A \to BC$ to get

$$\boxed{R_1 = (C, E), R_2 = (B, D), R_3 = (A, B, C), R_4 = (A, F)}$$

Note that if we continue by using the decomposition rule on $A \to BC$ to get $A \to B$ and $A \to C$, we over-normalize, and end up losing the functional dependency $A \to BC$.

5. Suppose we have a relation $R(A, B, C, D)$ with a multivalued dependency (MVD) $A \twoheadrightarrow BC$. If we know that the tuples $(a, b_1, c_1, d_1), (a, b_2, c_2, d_2), (a, b_3, c_3, d_3)$ are in the current instance of R, what other tuples do we know must also be in R?

   Since we have $A \twoheadrightarrow BC$, this means that we can swap BC in tuples with any other tuple, given that they agree on A, and create a new tuple that is in R. Also, by definition, BC is independent of D. So the (b_1, c_1) must be seen with all d_i, (b_2, c_2) must been with all d_i and (b_3, c_3) must be seen with all d_i. So the other tuples that must be in R are

   (a) (a, b_1, c_1, d_2)

   (b) (a, b_1, c_1, d_3)

   (c) (a, b_2, c_2, d_1)

   (d) (a, b_2, c_2, d_3)

   (e) (a, b_3, c_3, d_1)

   (f) (a, b_3, c_3, d_2)

6. For relation $R(A, B, C, D, E, F)$, suppose a functional dependency $AB \to E$ and two multivalued dependencies $AB \twoheadrightarrow C$ and $A \twoheadrightarrow B$ hold. Is R in 4NF? Explain your answer. If not, normalize it into 4NF.

   R is definitely not in 4NF because we have redundancy from the MVDs to deal with. Remember that we deal with the functional dependencies first, and normalize based on them.

   Step 1: Using $AB \to E$ decompose R into $R_1(A, B, E)$ and $R_2(A, B, C, D, F)$.

   Step 2: Using $A \twoheadrightarrow B$ decompose R_1 to $R_3(A, B)$ and $R_4(A, E)$.

   Step 3: Using $AB \twoheadrightarrow C$ decompose R_2 into $R_5(A, B, C)$ and $R_6(A, B, D, F)$.

Step 4: Using $A \twoheadrightarrow B$ decompose R_5 into $R_3(A, B)$ (duplicate) and $R_7(A, C)$.

Step 5: Using $A \twoheadrightarrow B$ decompose R_6 into $R_3(A, B)$ and $R_8(A, D, F)$.

Note that we must use the same MVD ($A \twoheadrightarrow B$) more than once because in the decomposed relations (using functional dependency of other MVD) we still find the 4NF violation.

In the end, we have as our final decomposition, $\boxed{R_3(A, B), R_4(A, E), R_7(A, C), R_8(A, D, F)}$

Part 2: Entity-Relationship Status – It's Complicated

1. You are to design a database that maintains information for producing a weekly television guide for a given region (such as the Greater Los Angeles region). The data should include information about television shows, television networks, cities, channels, show times, etc. For starters, you may make the following assumptions:

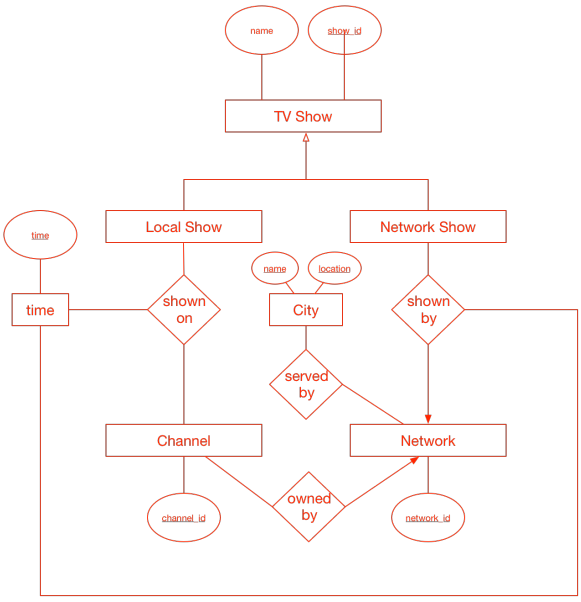
   - A given channel in a given city is associated with one network.
   - A given show is either owned by a network (and shown on a channel associated with that network) or is a local show and may be shown on any channel.
   - Not all shows are shown in all cities, and the days and times for a given show may differ from city to city.
   - You may ignore cable channels, which generally are not city-dependent.

   Please feel free to make additional assumptions about the real world in your design, as long as the assumptions are reasonably realistic and are stated clearly as part of your solution.

   Specify an entity-relationship diagram for your database. Dont forget to underline key attributes and include arrowheads and double lines.

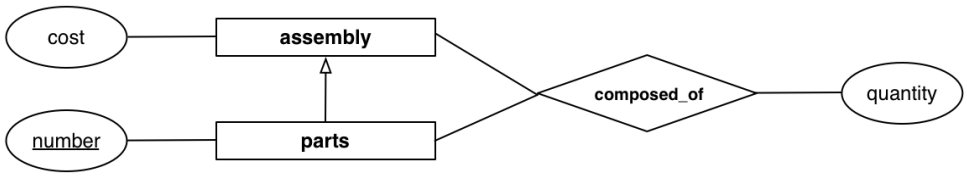
   Note that this question is fairly open-ended and there is no single right answer, but some designs are better than others.

   There are a ton of different ways to do this problem, and the main goal is for your assumptions to make sense.



2. This problem is based on an E/R design for a database used in a manufacturing company shown in Figure 1. This database stores information about parts. Each part has a part number, which uniquely identifies the part. A part may in fact be an assembly, which consists of some number of one or more subparts. For example, a bicycle might be described as an assembly consisting of one frame and two wheels; a frame is just a basic part; a wheel is an assembly consisting of one tire, one rim, and 48 spokes. Each assembly is also associated with the cost of assembling its subparts.

   Convert the E/R diagram to relations. For the translation of subclasses, assume that we generate multiple tables for specialization and that a subclass does not inherit non-key attributes from its superclass. The text does not properly discuss the circle: it just means an "attribute of an entity set."

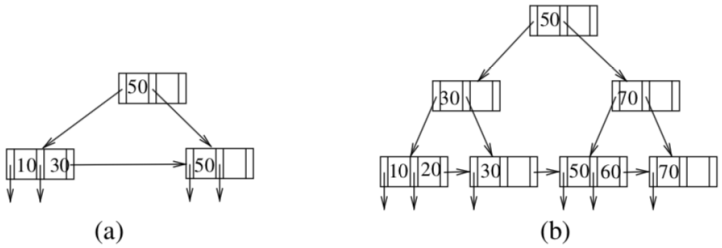


The only tricky thing in this conversion is the inheritance relationship (sometimes marked by ISA but is presented as an "other" representation). See section 7.8.6.1 in the text for more information. We get

```
Parts(number)
Assembly(number, cost)
ComposedOf(assemblynumber, partnumber, quantity)
```

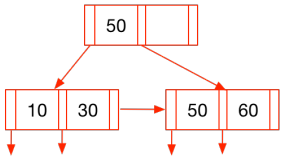
Part 3: Seeing the Forest for the Trees

Consider the following two B+trees for this problem. You may need to review chapter 11, or the lecture slides.

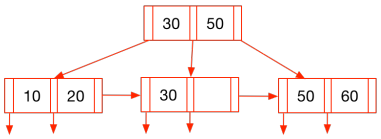


1. Show the final B+tree structure after we insert 60, 20, and 80 into Figure (a) in the given order.

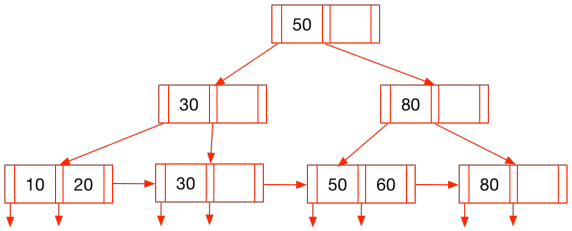
   (a) Inserting 60 is easy, we just insert it in the proper spot. No overflow occurs.



   (b) Inserting 20 causes an overflow in the bottom left leaf, but we know that 20 must go into this leaf, so we evict 30 and move it up a level. Unfortunately, we have lost 30 as a key which is illegal, so we must put it somewhere, but where? We cannot move it to the right subtree because 30 is less than 50, so we must create a new node under the root, only containing 30.



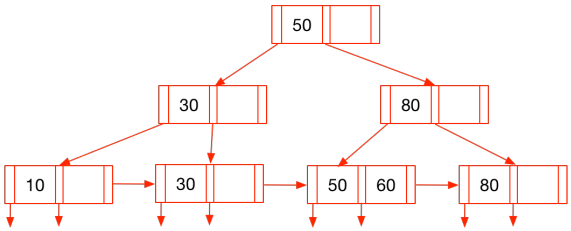
   (c) It gets worse. Now we need to add 80. It should go to the extreme far right of the bottom right leaf. But we have an overflow if we insert it there. We create a new node under the root with a router 80, which points to 50 and 60, and we create a new far right leaf containing only 80.



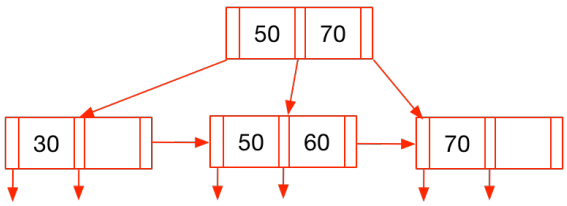
   Oh it gets worse...

2. Show the final B+tree structure after we delete 20, 10, and 70 from Figure (b) in the given order.

(a) Deleting 20 is easy because there must be at least $\lceil \frac{n}{2} \rceil$ values in a node. There will be one left when we delete 20, so easy peasy.



(b) Deleting 10 is where the trouble starts. We will have an empty (an underflow) node if we delete 10 as is. We end up losing a level in the left subtree as a result, and when that happens, we have to lose a level in the right subtree as well, to keep the tree balanced. To balance the tree, we end up cascading up 70 into the root.



(c) Finally we delete 70.