CECS 323 LAB BCNF NORMALIZATION

OBJECTIVE:

Get some practice performing BCNF normalization and seeing what we mean when we talk about "losing" a functional dependency during the normalization.

INTRODUCTION:

In this exercise we are going to use a publishing scenario. The business rules are:

- Instead of individual authors, books are written and published by **groups** of authors, called writing groups.
- Each writing group has a unique name, and a genre (like comedy, science fiction, horror, …) that characterizes that writing group.
- The genre name is unique. That is, not two genres have the same name.
- Each genre has a description.
- A given writing group only has one genre, but a given genre could apply to multiple writing groups.
- Each book is published by just one publisher.
- The publisher is located by their zip code.
- Each writing group makes sure that they **never** publish two books by the same title.
- Each publisher makes sure that no two books that the publisher publishes have the same title.

We will start with a single table with the following attributes: *R* = {ISBN, group_name, group_genre, group_genre_description, book_title, book_publication_date, book_publisher_name, book_publisher_city, book_publisher_state, book_publisher_zip_code}

The following minimal functional dependencies (F) are given:

F = {

1. {ISBN} → {group_name, book_title}
2. {book_publisher_name} → {book_publisher_zip_code}
3. {book_publisher_zip_code} → {book_publisher_city, book_publisher_state}
4. {group_name} → {group_genre}
5. {group_genre} → {group_genre_description}
6. {book_title, book_publisher_name} → {group_name, book_publication_date, **ISBN**}
7. {book_title, group_name} → {book_publisher_name, book_publication_date, **ISBN**}

}

Formally, we can now say that **R** = (*R*; *F*)

R denotes the complete relation: both the set of attributes as well as the FDs. That set of attributes is denoted by *R*, and the set of FDs is *F*. If we need to split **R**, we will create $\mathbf{S_1}$ and $\mathbf{S_2}$. If we split $\mathbf{S_1}$ for instance, that will create $\mathbf{S_{1.1}}$ and $\mathbf{S_{1.2}}$ and so forth as we break the original relation **R** down in the normalization process.

To begin the normalization algorithm, determine whether we have any subkeys in **R**. You will hopefully remember several of them from the previous lab. Remember that the definition of a subkey is for any functional dependency X → Y in *F*, if X^+ (the closure of X) is a proper **subset** of *R*, then X is a subkey.

Remember that we will only concern ourselves with minimal subkeys for the purposes of normalization. Also know (you can prove this to yourself) that each of the above functional dependencies (FDs) are minimal. That is, we cannot remove any attributes on the left, and still have a functional dependency that determines all the attributes on the right.

To make the description below a little easier, I am going to talk about the determinant and the dependent of a FD. The determinant is the left-hand side, and the dependent is the right-hand side.

1. With that out of the way, let us pick a subkey and use it to start our normalization algorithm. The determinant of: {book_publisher_name} → {book_publisher_zip_code} is one such subkey (there are several others).
    a. The first thing that we need to do is find the closure of {book_publisher_name} so that we bring all the functionally determined attributes that it has along into the new relation that we form. In this case, we start with {book_publisher_name, book_publisher_zip_code} and add {book_publisher_city, book_publisher_state} by using FD_3. There are no other FDs that we can use to expand the closure, so we finish with {book_publisher_name, book_publisher_zip_code, book_publisher_city, book_publisher_state} as the closure of {book_publisher_name}.
    b. The functional dependencies in our new relation are F_2, and F_3 from the original FDs in **R**.
    c. **S_1**
        i. S_1 = {book_publisher_name, book_publisher_zip_code, book_publisher_city, book_publisher_state}
        ii. F_1 = {{book_publisher_name} → {book_publisher_zip_code}, {book_publisher_zip_code} → {book_publisher_city, book_publisher_state}}.
    d. **S_2**
        i. S_2 = {ISBN, group_name, group_genre, group_genre_description, book_title, book_publication_date, book_publisher_name}
            1. The FD: {book_publisher_name} → {book_publisher_zip_code, book_publisher_city, book_publisher_state} we can denote symbolically as X → Y. This set of attributes (S_2) is what is left over when we subtract Y – X off from R.
        ii. F_2 = {ISBN} → {group_name, book_title}, {group_name} → {group_genre}, {group_genre} → {group_genre_description}, {book_title, book_publisher_name} → {group_name, book_publication_date, ISBN}, {book_title, group_name} → {book_publisher_name, book_publication_date, ISBN}}.
            1. We removed F_2 and F_3 from ***F*** because both of those original FDs involved attributes that are now part of S_1.
2. Looking at **S_1**, we see that the determinant of the second functional dependency in F_1 is a subkey itself. The closure of {book_publisher_zip_code} is {book_publisher_zip_code, book_publisher_city, book_publisher_state}. We need to break **S_1** into **$S_{1.1}$** and **$S_{1.2}$** such that:
    a. **$S_{1.1}$**
        i. $S_{1.1}$ = {{book_publisher_zip_code, book_publisher_city, book_publisher_state};
        ii. $F_{1.1}$ = {{book_publisher_zip_code} → {book_publisher_city, book_publisher_state}}

   b. **$S_{1.2}$**
      i. *$S_{1.2}$* = {book_publisher_name, book_publisher_zip_code} which is just S_1 – (Y – X) where X → Y is just a symbol for {book_publisher_zip_code} → {book_publisher_city, book_pubslisher_state}.
      ii. *$F_{1.2}$* = {{book_publisher_name} → {book_publisher_zip_code}}
3. **S_2** has **two** subkeys in it, so I will start[1] with the determinant of {group_genre} → {group_genre_description} and split that out into **$S_{2.1}$** and **$S_{2.2}$** such that:
   a. **$S_{2.1}$** thankfully is short:
      i. *$S_{2.1}$* = {group_genre, group_genre_description}
      ii. *$F_{2.1}$* = {{group_genre} → {group_genre_description}}
   b. **$S_{2.2}$**
      i. *$S_{2.2}$* = {ISBN, group_name, group_genre, book_title, book_publication_date, book_publisher_name}
         1. We got this list by subtracting Y – X from S as usual.
      ii. *$F_{2.2}$* = {{ISBN} → {group_name, book_title}, {group_name} → {group_genre}, {book_title, book_publisher_name} → {group_name, book_publication_date, ISBN}, {book_title, group_name} → {book_publisher_name, book_publication_date, ISBN}}
         1. This *F* is what we get by removing {group_genre} → {group_genre_description} from *S_2*.
4. **$S_{2.2}$** has a subkey in it, which is the determinant of the FD: {group_name} → {group_genre}. So, we break this relation down into **$S_{2.1.1}$** and **$S_{2.1.2}$**.
   a. **$S_{2.2.1}$**
      i. *$S_{2.1.1}$* = {group_name, group_genre}
      ii. *$F_{2.1.1}$* = {group_name} → {group_genre}
   b. **$S_{2.2.2}$**
      i. *$S_{2.1.2}$* = {ISBN, group_name, book_title, book_publication_date, book_publisher_name}
      ii. *$F_{2.1.2}$* = {ISBN} → {group_name, book_title}, {book_title, book_publisher_name} → {group_name, book_publication_date, ISBN}, {book_title, group_name} → {book_publisher_name, book_publication_date, ISBN}

The result of decomposing the original relation **R** is: **$S_{2.2.1}$, $S_{2.2.2}$, $S_{2.1}$, $S_{1.1}$, $S_{1.2}$**. These relations are all related to each other. Each time that we split off one relation from another (for instance breaking **$S_{2.1.1}$** off from **$S_{2.2}$**) we left the determinant (the primary key if you will) from the relation that we split off as a migrating foreign key in the other relation. In this example, **$S_{2.1.2}$** has the group_name left behind, which allows that relation to reference the appropriate tuple in **$S_{2.1.1}$**. Which means that you should be able to draw a UML diagram directly from **$S_{2.1.1}$, $S_{2.1.2}$, $S_{1.1}$, $S_{1.2}$, $S_{2.2}$.**

[1] Remember that the algorithm works (that is the result is a BCNF set of relations) regardless of what order you tackle the subkeys. However, the order in which you address the subkeys may change the structure of the result.

PROCEDURE:

1. The above algorithm is not deterministic. That is, the same algorithm **could**[2] yield a different set of relations, depending on the order in which you address the subkeys. Given that, start with the same ***R*** that we did, but split off the determinant for {group_name} → {group_genre} first, and then keep following the same process, splitting the relations not in BCNF as you go.
2. Build the UML diagram that results from your decomposed relations.
3. Build the relation scheme diagram from the UML diagram.
    a. Be careful to show any candidate keys that the functional dependencies call for.

WHAT TO TURN IN:

- Your decomposition of **R** (the original relation) down into its BCNF relations, given the different starting point.
- Your new and improved UML diagram.
- The relation scheme diagram.
- Your team's filled out collaboration document. You can find a template for that here.

[2] There is no guarantee that a different order of subkeys will give you a different outcome, just that it is possible.