BCNF and 3NF

Given a relation schema R with a set F of FDs that hold over R, let's define X to be a subset of the attributes of R, and A is an attribute of R. R is in BCNF (Bryce-Codd Normal Form) if for every FD X -> A in F, one of the following is true:

- $A \in X$ (reads A is an element of X) – i.e. it is a trivial FD or
- X is a superkey

Basically – this says that the only non-trivial dependencies are those in which a key determines some attribute. If this is the case, each tuple can be thought of as a separate entity (or relationship), with a key and a set of describing attributes.

If a relation is in BCNF form, it is given that no redundancy can be detected using the FDs alone – making it the most desirable form (from the point of redundancy). Every field of every tuple records a piece of info that cannot be inferred from the values in the other fields (from a FD point of view).

So let's look at a couple of examples:

If you have a relation schema R that is ABCD, and a set F of FDs that is:

F = {A -> BCD} then clearly R is in BCNF because the only FDs are : A-> B, A -> C, and A -> D and in each case, A is a superkey of R.

How about this:

R = ABCDE, F = {A -> B, C -> DE}

In this case, the candidate key of ABCDE is AC. Is R in BCNF? No – because in the FD A->B, A is not a superkey (it is only part of a key). What do we have to do to decompose it into BCNF?

We look at the first FD A -> B. Since this FD causes a violation of BCNF, we create a decomposition of:

(AB)(ACDE)

where we created 2 new schemas R_1 and R_2. Let's look at R_1. Is it in BCNF? In the set F, the FD A->B will hold over R_1, and the FD C -> DE will not – because the attributes in this sub-relation do not include the attributes in the FD. We end with:

R_1 = (AB), F_1 = {A -> B}

Which is clearly in BCNF, because A is the candidate key of R_1 and the only FD has A as a superkey in it. Now let's look at R_2.

R_2 = (ACDE), F_2={C -> DE} (A -> B is not included, because B is not in this relation). The candidate key for R_2 is AC. Is R_2 in BCNF? No, because of the FD C -> D, where C is not a superkey. So we recursively apply the same technique as before. R_2 is broken into:

(AC)(CDE)

Where 2 more schemas have been created. Looking at the first, we see that the candidate key is AC and that none of the FDs hold over it, so it is in BCNF. The second has C as a candidate key because the only FD that holds over it is C -> DE. This is also in BCNF. Our final decomposition looks like:

(AB)(AC)(CDE)

Third Normal Form

While BCNF is ideal for redundancy, it has a side effect sometimes of not preserving dependencies. As as example, let's look at:

R = (ABC), F = {AB -> C, C-> A). There are 2 candidate keys, AB, and BC. When we inspect R, we see the first FD (AB -> C) does not cause a problem, but that C -> A does. We pull it out into a separate relation:

(BC)(AC) – which is in BCNF because BC is the candidate key of the first with no FDs holding over it, and C is the candidate key of the second with C –> A holding over it. When we decomposed however, we lost the FD AB -> C.

Third normal form provides a way to address this. In third normal form, an additional condition is possible:

- $A \in X$ (reads A is an element of X) – i.e. it is a trivial FD or
- X is a superkey
- A is a part of some key for R

Now let's look at R. R = (ABC), F = {AB->C, C->A}. The candidate keys are AB and BC (as before). When we inspect the first FD, we see that AB is a candidate key, so the FD does not cause a violation. When we inspect the second, we see that A is part of a candidate key, so it passes the test as well. This relation is in 3NF form. It is always possible to decompose a relation into 3NF that is both lossless join and dependency preserving.