**CPSC 304: Introduction to Relational Databases**
**Tutorial #3 Solution: Functional Dependencies and Normalization – SuperRent**

---

1. Give an instance of **SuperRentInfo** (i.e., a relation) that illustrates these three anomalies: insertion, deletion, and update.

| I | N | E | S | ED | C | CN | B | T | P |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Tom | a@gmail.com | 2020-01-01 | 2020-02-02 | Toronto | 2 | Markham | SUV | 1 |
| 2 | Jen | j@gmail.com | 2020-03-01 | 2020-04-01 | Vancouver | 1 | Richmond | Sedan | 2 |
| 3 | Nana | n@gmail.com | 2020-05-01 | 2020-08-01 | Toronto | 5 | Markham | Van | 3 |

- **Insertion anomaly:** We cannot represent the fact, for example, that Peter is a customer with an email address, unless we also know what vehicle type is reserved by Peter (that is, Peter MUST reserve a vehicle and cannot just exist in the database, and this is bad design). The only way out is to introduce NULL values. However, NULLs are problematic; therefore, it is desirable to minimize their use.

- **Deletion anomaly**: Suppose we delete's Jen customer information. Deleting information related to Jen forces us to accidentally lose information about our SuperRent branches and their location, as well as any reservations Jen made.

- **Update anomaly:** Suppose the branch name for the branch in Toronto is updated from Markham to another name like York. This means every record that contains that info in the above table must be updated; thus, a chain of updating redundancy needs to be handled.

2. Consider the decomposition of the relation **SuperRentInfo** into:

   SI1(I, N, E, S, C, B) and
   SI2(C, P, T, ED, CN, B)

   Is this a lossy, or a lossless-join, decomposition? Justify your answer.

   Consider the common attributes between SI1 and SI2 (i.e., the intersection):

   SI1 ∩ SI2 = { C, B }

   We now check if that closure corresponds to SI1 or SI2, that is:

   SI1 ∩ SI2 → SI1 **OR** SI1 ∩ SI2 → SI2

   Let's check the closure by using the given FDs :

   (C,B)+ = {C, B} // *we can't use any FDs, we're stuck!*

   {C, B} does not include either SI1 or SI2; so, we conclude that this decomposition is **lossy.**

3. Repeat (2) for the decomposition:

   SI1(I, N, E, S, C, CN, B) and
   SI2(C, P, T, ED, CN, B)

   Consider common attributes between SI1 and SI2 (i.e., the intersection):

   SI1 ∩ SI2 = { C, CN, B }

   We now check if that closure corresponds to SI1 or SI2, that is:

   SI1 ∩ SI2 → SI1 **OR** SI1 ∩ SI2 → SI2

   Let's check the closure by using the given FDs:

   {C, B, CN}+ = {C, B, CN, I, N, E, T, S, ED}
   {C, B, CN}+ does include **SI1**; so, we conclude this decomposition is **lossless.**

4. Find all keys for **SuperRentInfo**.

   We use the LHS-Both-RHS table to investigate:

| Attributes | LHS | Both | RHS | Conclusion |
|---|---|---|---|---|
| I | | Yes | | Must check |
| N | | | Yes | No need to check |
| E | | | Yes | No need to check |
| S | | | Yes | No need to check |
| ED | | | Yes | No need to check |
| C | | | Yes | No need to check |
| CN | | Yes | | Must check |
| B | Yes | | | Must be in a key |
| T | | Yes | | Must check |
| P | Yes | | | Must be in a key |

   Attributes B and P do not appear on the RHS and must be part of every key.

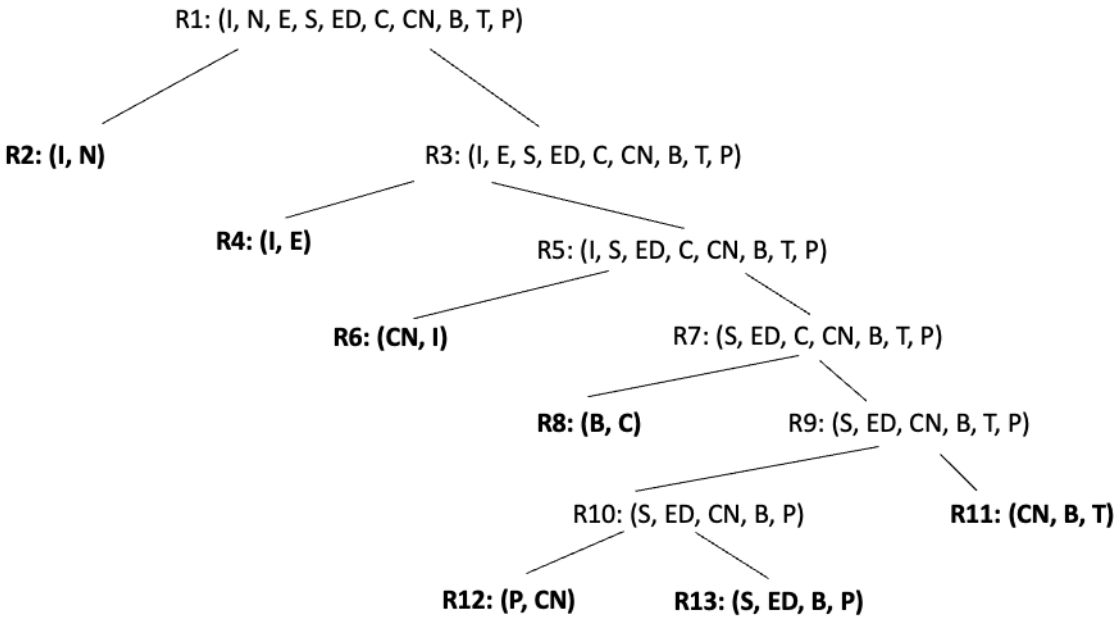
   Check the closure of { B, P } and see if you can get every attribute from it, using the FDs:

   {B,P}+ = { B, P, C, I, CN, T, S, ED, N, E } *// It works! Note: if this does not get you all the attributes, then you would iteratively investigate LHS + "Both" – one by one – (i.e. { B, P, I}+ and {B, P, CN}+ and so on. )*

   Based on this, we claim that BP is the only key.

5. Obtain a lossless-join, BCNF decomposition of **SuperRentInfo**.

   We have **SuperRentInfo** (I, N, E, S, ED, C, CN, B, T, P) with the above FDs. Currently, all FDs violate the BCNF condition. Pick any one of them, say FD 1: I → N, E (use one attribute on the right-hand side each time). Below, each node is split into two children, recursively, whenever the relational schema at the node has an applicable FD that violates BCNF. We terminate when each leaf satisfies BCNF with respect to its applicable set of FDs. You can sketch an appropriate tree. Reminder, **BCNF MAY NOT BE DEPENDENCY PRESERVING.**




The bold relations are your resultant relations after BCNF decomposition. R2 and R4 decompose based on FD 1; R6 decomposes based on FD 2; R8 decomposes based on FD 3; R11 decomposes on FD 4. R12 decomposes on FD 6 because it can be broken down to P -> CN and P -> I, so we preserve P -> CN but not P -> I . In summary, FD5 and partial of FD6 is **not preserved** but the above is still a lossless join BCNF decomposition.

6. Find a minimal cover for this set of FDs.

   **Step 1**: Split RHS attributes to obtain simpler FDs

   1. I → N
   2. I → E
   3. CN → I
   4. B → C
   5. CN, B → T
   6. CN, B, T → S
   7. CN, B, T → ED
   8. P → I
   9. P → CN

**Step 2**: Can we delete any attributes from the LHS?

Only FDs 5, 6 and 7 are candidates for this because all other FDs have a single attribute on their LHS.

FD 5: Note that none of the attributes can be removed from the LHS because the closure fails to contain T when we drop an attribute. For example, if we drop CN, T can't be obtained in any alternative way, that is, T is in no other FD's RHS. The same can be said if we drop B.

FD 6: T can be dropped because FD 5 will compensate to generate T. Result: **CN, B -> S**

FD 7: Same idea as above. Result: **CN, B -> ED**

**Step 3**: Is any FD redundant?

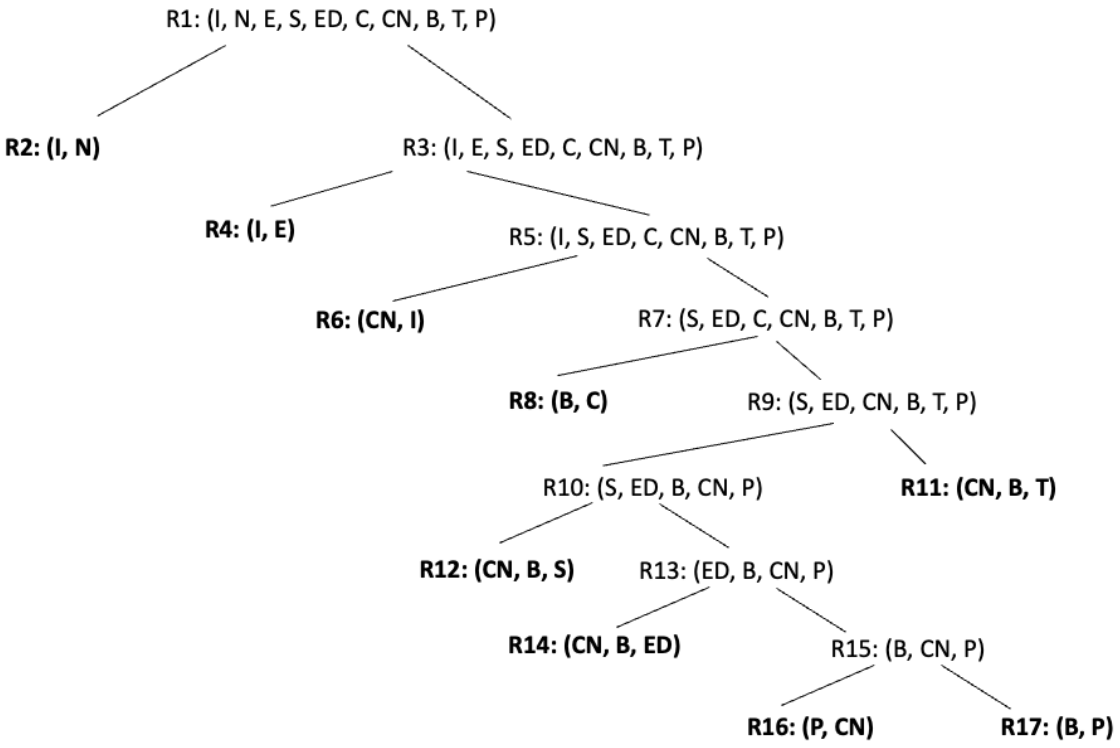
Check one-by-one that no FD in the above set is redundant. For example, the closure of I, computed without using FD 1, does not contain N; therefore, FD 1 is not redundant.

The only redundant FD is found to be P → I because we can use FD 9 and FD 3 to mimic it. We then eliminate that FD and come to our final result of a minimal cover:

I → N
I → E
CN → I
B → C
CN, B → T
CN, B → S
CN, B → ED
P → CN

7. Obtain a lossless-join, dependency-preserving, 3NF decomposition of **SuperRentInfo** by using the **decomposition** method.

   All FDs violate 3NF, so we use the calculated minimal cover to decompose accordingly.




   This is an example of a "perfect" scenario (rare case) where there is no need to add any relations to maintain this to be dependency preserving. If any of the minimal cover FDs are not preserved via decomposition, you have to add it at the very end to get your final set of 3NF abiding relations.

8. Obtain a lossless-join, dependency-preserving, 3NF decomposition of **SuperRentInfo** via the **synthesis** method.

   Using the synthesis approach, we first create a relational schema corresponding to each FD in the minimal cover. This gives the database schema {INE, CN+I, BC, CN+BTS+ED, P+CN}—with each relational schema having the obvious FD and key. This set of relational schemas preserves all FDs, by construction/definition.

   If no relation in the set is a superkey of the original relation, then add a relation containing the key.

   We see that B,P is not in any of the above set of relations even though it's the key. No worries! Just add it in!

   Final Result:

   R1 = (I, N)
   R2 = (I, E)
   R3 = (CN, I)
   R4 = (B, C)
   R5 = (CN, B, T)
   R6 = (CN, B, S)
   R7 = (CN, B, ED)
   R8 = (P, CN)
   R9 = (B, P)

   Compare this to your answer for Q7. Are there any differences?