Boyce-Codd Normal Form

- Potential problem with 3NF
  <ex> R=(Course#, Section #, Professor)
  F={C# S# → Prof, Prof → C#}, Key=(C# S#, Prof S#)
  - R is in 3NF although Prof S# → Prof → C#, since no non-prime attribute exists (no transitive dependency)
  - Prof# → C# could be duplicated in many tuples

- BCNF
  A relation schema R is in BCNF if for every set of attributes X in R and attribute A in R, X is a superkey whenever X → A.
  --- no attribute is transitively dependent on a key
  R₁=(Prof, S#) K₁=(Prof S#)
  R₂=(Prof, C#) K₂=(Prof)
  --- R₁ and R₂ are both in BCNF

- One problem: FD (C# S#) → Prof is not embedded in either relation alone, hence its violation is not easily detectable

Other Example

<ex> R=(City, Street, Zip) F={CS Z, Z C}
Is R in 3NF?
Is R in BCNF?
Decompose into
R₁=(S,Z) R₂=(C,Z)
(1) lossless join?
R₁ ∩ R₂ = Z, R₁ − R₂ = C, Z → C in F? Yes
(2) dependency preserving?
R₁: only trivial FD
R₂: Z → C and trivial FD
(ΠR₁(F)∪ΠR₂(F))⁺ ≠ F⁺
They do not imply CS → Z.
Hence the decomposition does not preserve dependency.

BCNF

In general, consider the schema S=(J,K,L) with F={JK→L, L→K}.
Clearly, S is not in BCNF, since L→K and L is not a superkey.
However, every BCNF decomposition of S will fail to preserve JK→L.
Probably, we will end up with R₁=(K,L) and R₂=(J,L), which preserves L→K but not JK→L.

- Not every set of FDs for relations can be represented by a BCNF schema with all FDs embedded in a single relation
- It is a NP-complete problem to decide if a given DB schema is in BCNF
- What is the relationship between 3NF and BCNF?
  3NF = BCNF
  3NF ⊆ BCNF
  3NF ⊇ BCNF

BCNF

- Every BCNF schema is also in 3NF
  - BCNF is a more restrictive constraint than 3NF
    --- BCNF cannot have any transitive dependency
  S=(J,K,L) with F={(K→L, L→K)}
  S is not in BCNF but in 3NF, because JK is a candidate key and JL is also a candidate key, and JK→L→K does not violate the requirement of 3NF
- Advantages and shortfalls of 3NF
  - always possible to obtain a 3NF schema without sacrificing lossless join or dependency preserving
  - to represent a certain meaningful relationship, it is necessary to use null values
    --- Since L→K, there’s a relationship between L and K.
    What should be the corresponding J value?
    --- it might be null
**BCNF and 3NF**

- BCNF and dependency preservation
  - without dependency test, violation of data integrity possible
  - no dependency preservation implies no efficient test, resulting in high performance penalty during operation
  - neither of them is attractive
- Dependency preservation with 3NF is preferred
  - limited amount of redundancy imposed by transitive dependencies allowed in 3NF is lesser evil
- Relational database design goal
  - 1st choice: BCNF + lossless join + dependency preservation
  - 2nd choice: if 1st choice not possible, change BCNF to 3NF instead of sacrificing the other two

---

**BCNF Decomposition**

Given R and F (R is not in BCNF),
1. Let X→Y be a non-trivial FD that holds on R such that X→R is not in F+
   --- X is not a superkey of R and Y ∩ X = ∅
2. Let R_1 = R → Y, R_2 = XY (X is a key for R_2)
3. Continue until no BCNF schema is resulted

<ex> Lending (B-name, Assets, B-city, Loan#, C-name, Amount)
   Key = (Loan#, C-name)
   F = { B-name → Assets B-city, Loan# → B-name Amount }

1. B-name → Assets, but b-name is not a key
   Decompose into R_1 = (B-name, Assets) \( \sqrt{ } \)
   R_2 = (B-name, B-city, Loan#, C-name, Amount)
2. B-name → B-city: R₃ = (B-name, B-city) \( \sqrt{ } \)
   R₄ = (B-name, Loan#, C-name, Amount)
3. Loan# → Amount: R₅ = (Loan#, Amount) \( \sqrt{ } \)
   R₆ = (B-name, Loan#, C-name)
4. Loan# → B-name: R₇ = (Loan#, B-name) \( \sqrt{ } \)
   R₈ = (Loan#, C-name) \( \sqrt{ } \)