Example 1: BCNF

- **Patrons(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)**
- FDs:
  - **name** $\rightarrow$ **addr**
  - **name** $\rightarrow$ **favBeer**
  - **beersLiked** $\rightarrow$ **manf**
- We know the schema is not in BCNF
- An FD in violation: name $\rightarrow$ addr
- $\{name\}^+ = \{name, addr, favBeer\}$

Example 1: BCNF

- **Patrons(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)**
- {name}$^+$ = {name, addr, favBeer}
- Decomposed relations:
  - Patrons_1(name, addr, favBeer}
  - Patrons_2(name, beersLiked, manf}
- But are Patrons_1 and Patrons_2 in BCNF?
  - To check, we must project FDs
  - In this example, relatively easy

Example 1: BCNF

- Decomposed relations:
  - **Patrons_1(name, addr, favBeer)**
  - Patrons_2(name, beersLiked, manf)
- For Patrons_1 relevant FDs are:
  - name $\rightarrow$ addr
  - name $\rightarrow$ favBeer
- Both FDs have a key/superkey on the left-hand side
- Therefore Patron_1 is in BCNF

Example 1: BCNF

- Decomposed relations:
  - $Patrons_1$(name, addr, favBeer}
  - **$Patrons_2$(name, beersLiked, manf)**
- For $Patrons_2$ relevant FDs are:
  - beersLiked $\rightarrow$ manf
  - (The two original FDs involving name had RHS attributes that are not in $Patrons_2$.)
  - BCNF violation (beersLiked is not a key)
- {beersLiked}+ = {beersLiked, manf}
  - $Patrons_3$(beersLiked, manf)
  - $Patrons_4$(name, beersLiked)

Example 1: The saga concludes...

- The resulting decomposition of Patrons is now:
  - $Patrons_1$(name, addr, favBeer)
  - $Patrons_3$(beersLiked, manf)
  - $Patrons_4$(name, beersLiked)
  - ($Patrons_2$ disappears)
- Notice that:
  - $Patrons_1$ tells use about patrons
  - $Patrons_3$ tells us about beers
  - $Patrons_4$ tells us the relationship between patrons and the beers they like

Another possible problem...

- There is a particular configuration of FDs that cause trouble when we decompose via BCNF
- Example:
  - Schema: R(A, B, C)
  - AB $\rightarrow$ C
  - C $\rightarrow$ B
  - Two keys: AB and AC
- Intuition:
  - A is a street address
  - B is a city
  - C is a postal code

Another possible problem...

- Example:
  - Schema: R(A, B, C)
  - AB $\rightarrow$ C (e.g., street city $\rightarrow$ postalCode)
  - C $\rightarrow$ B (e.g., postalCode $\rightarrow$ city)
  - Two keys: AB and AC
- C $\rightarrow$ B is a BCNF violation
  - This suggests decomposing ABC into AC and BC
- The big problem:
  - With this decomposition, we cannot possibly enforce the first FD (AB $\rightarrow$ C)

An Unenforceable Functional Dependency

street	postalCode
1638 Myrtle Ave.	V8R 4J8
1638 Myrtle Ave.	V8R 4J9

city	postalCode
Victoria	V8R 4J8
Victoria	V8R 4J9

Join tuples with equal postal codes

street	city	postalCode
1638 Myrtle Ave.	Victoria	V8R 4J8
1638 Myrtle Ave.	Victoria	V8R 4J9

Although no FDs were violated in the decomposed relations, the FD **street city** $\rightarrow$ **postalCode** is violated by the database as a whole.

Third Normal Form (3NF)

- Using 3NF we can avoid the problem we have just described.
- 3NF modifies the BCNF condition such that we do not decompose when given the situation
- **Prime attribute**: An attribute which is a member of any key
- A functional dependency $\mathbf{X} \rightarrow \mathbf{A}$ violates 3NF iff:
  - X is not a superkey, and
  - A is not prime.

Example: 3NF

- Recall our suggested problem scenario:
  - Keys were AB and AC
  - FDs were AB $\rightarrow$ C and C $\rightarrow$ B
- From this we see that:
  - A is prime
  - B is prime
  - C is prime
- So although C $\rightarrow$ B violates BCNF...
  - ... the schema does not violate 3NF.
  - (Which should suggest to you that a schema in BCNF is also in 3NF!)

What we must have from 3NF and BCNF

- Whenever we decompose a schema, there are **two important properties** we want to exist

1. **Lossless join**:
   - It should be possible to project the original relations onto the decomposed schema...
   - ... and then reconstruct the original schema.
2. **Dependency preservation**:
   - It should be possible to check the projected relations for whether or not the given FDs are satisfied.

What we must have by 3NF and BCNF

- With BCNF:
  - We can always get lossless join, but not necessarily dependency preservation.
  - We saw this with the "street city postalCode" example.
- With 3NF:
  - We can get both lossless join and dependency preservation.

Testing for a Lossless Join

- Suppose we project R onto R_1, R_2, ... R_n.
- Can we recover an original instance of R from rejoining the R_is?
- Note there is a subtlety here:
  - Any tuple in R can be recovered from its projected fragments – no problem here.
  - However, when we rejoin, **do we find new tuples that didn't exist in the original relation instance**?
- The challenge: **We want to convince ourselves that lossless joins will occur given any valid original relation instance R and decomposition R_1, R_2, ... R_n**

Chase test (1)

- We accomplish this via **the chase test**
- Intuition:
  - Suppose some tuple **t** results from a join of the decomposed schemas
  - That means **t** is the join of projections of some tuples in R (one for each R_i in the decomposition).
  - We use the given FDs to show that one of these tuples must be **t**.
- Put differently, we want to prove that the following both hold hold for **t**:
  - **t** $\in \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rn}(R)$
  - **t** $\in$ R

Chase test (2)

- We start by assuming **t** = abc... (i.e., letters indicate possible component values)
- For each relation R_i:
  - There is a tuple $\mathbf{t_i}$ of R that has a, b, c, ... in the attributes of R_i
- $\mathbf{t_i}$ can have any values in other attributes (i.e., those attributes not in R_i)
  - For these attributes, we will use the same letter as in **t** but with a subscript.
- Intuition:
  - Want to build a set of tuples $\mathbf{t_i}$...
  - ... and then use the functional dependencies to prove (?) equivalences amongst attributes between different $\mathbf{t_i}$ tuples.