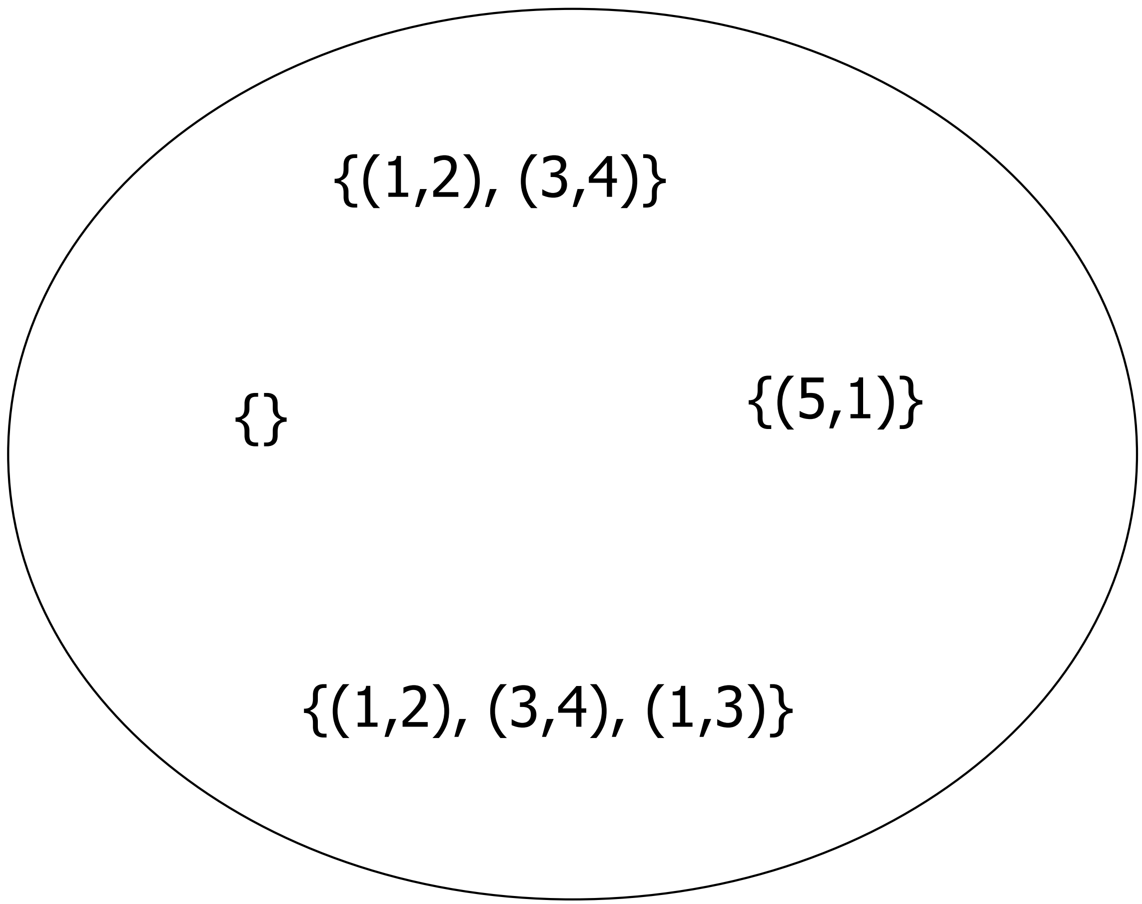
A Geometric View of FDs

- Imagine the set of all *instances* of a particular relation.
- That is, all finite sets of tuples that have the proper number of components.
- Each instance is a point in this space.

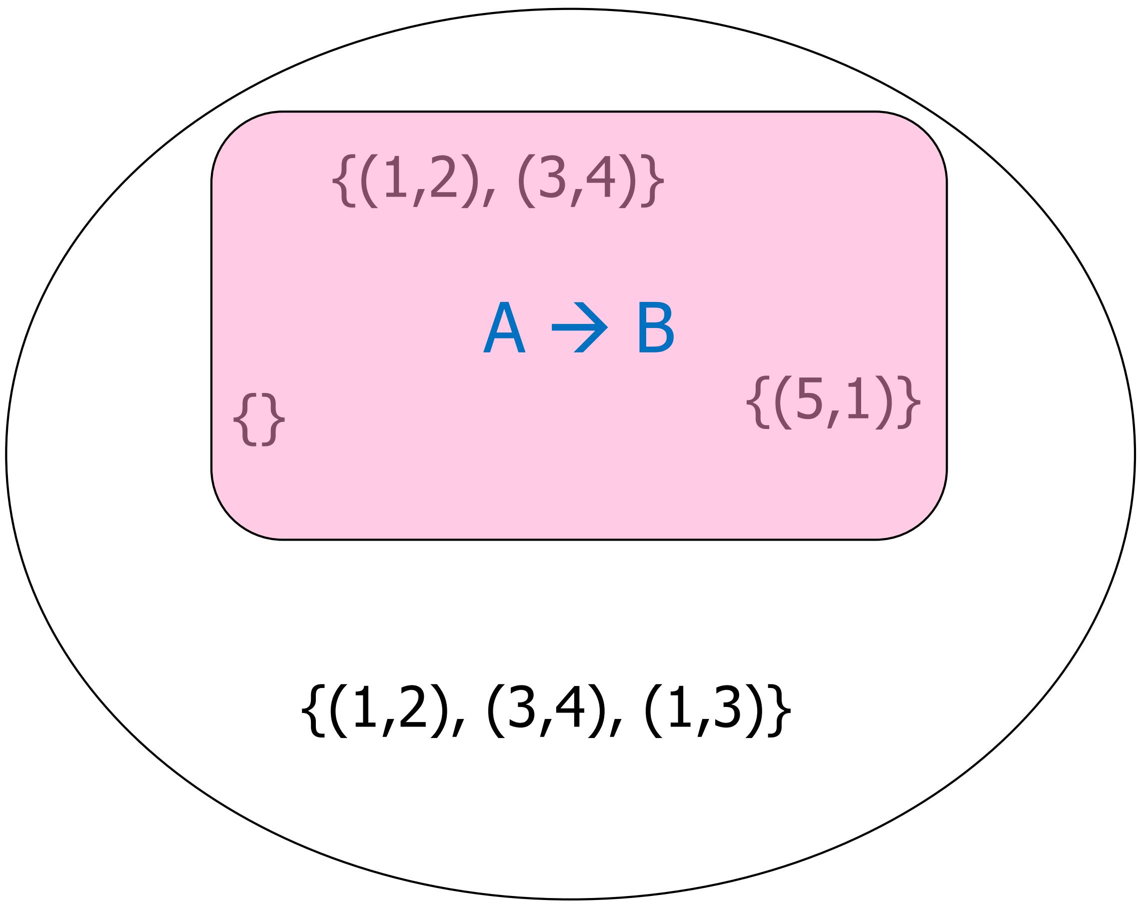
Example: R(A,B)



An FD is a Subset of Instances

- For each FD $X \rightarrow A$ there is a subset of all instances that satisfy the FD.
- We can represent an FD by a region in the space.

Example: A → B for R(A,B)

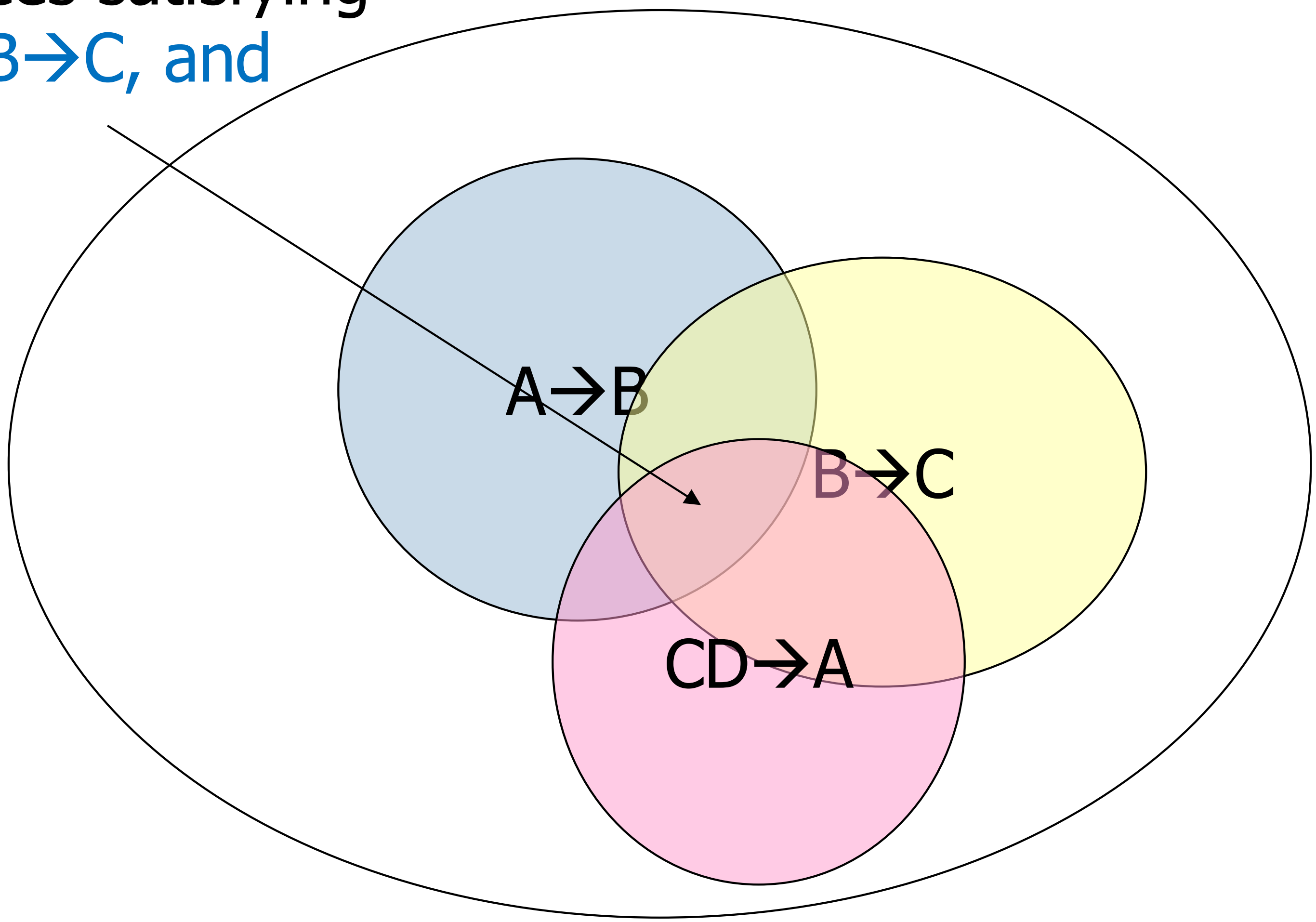


Representing *Sets* of FDs

- If each FD is a set of relation instances, then a collection of FDs corresponds to the intersection of those sets.
  - Intersection = all instances that satisfy all of the FDs.

Example

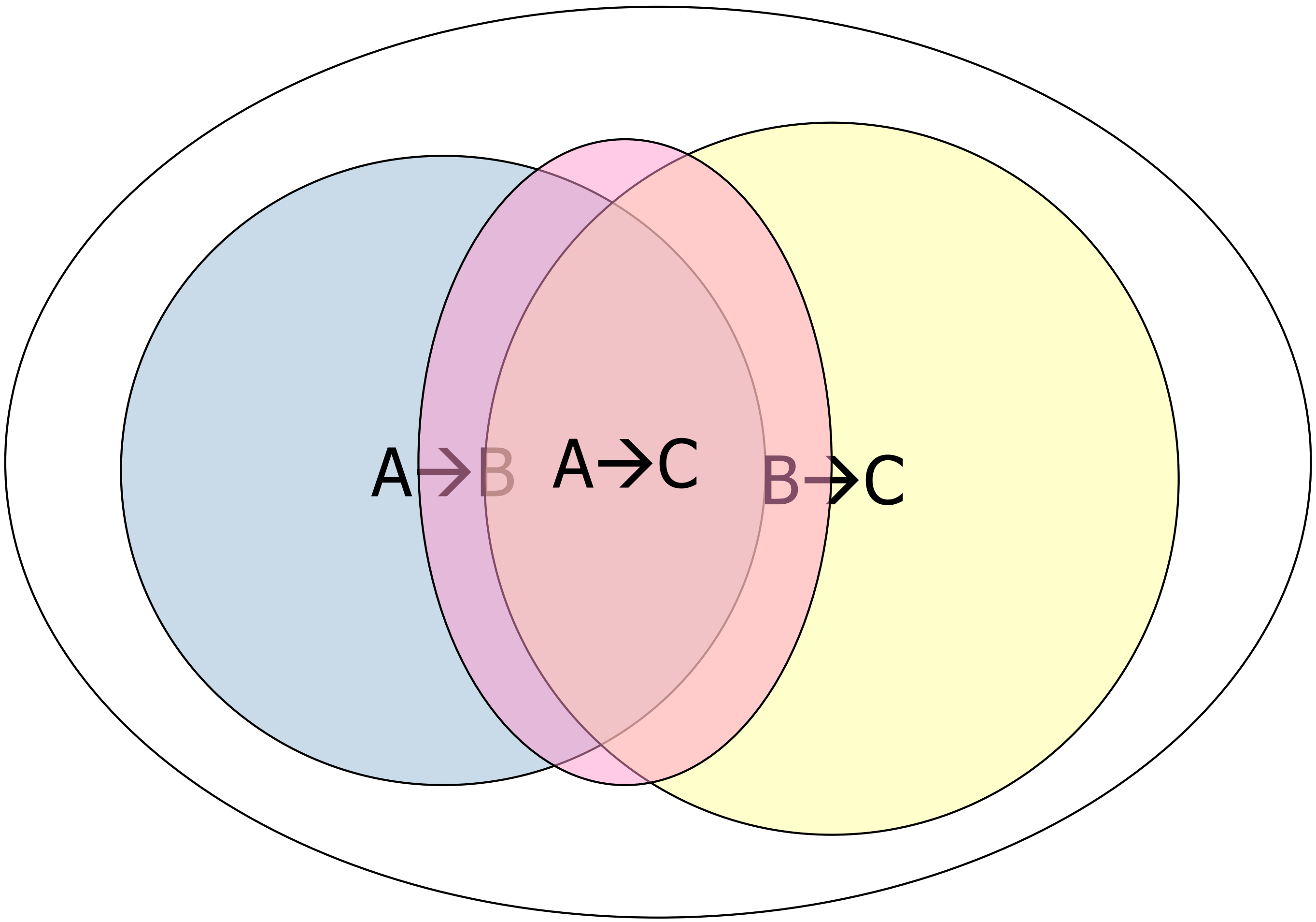
Instances satisfying
A→B, B→C, and
CD→A



Implication of FDs

- If an FD $Y \rightarrow B$ follows from FDs $X_1 \rightarrow A_1, \ldots, X_n \rightarrow A_n$, then the region in the space of instances for $Y \rightarrow B$ must include the intersection of the regions for the FDs $X_i \rightarrow A_i$.
  - That is, every instance satisfying all the FDs $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$.
  - But an instance could satisfy $Y \rightarrow B$, yet not be in this intersection.
- For a set of FDs ***F***, ***F***$^+$ (the closure of ***F***) is the set of all FDs implied by ***F***

Example



Closure of F

- For a set of FDs ***F***, ***F***$^+$ (the closure of ***F***) is the set of all FDs that can be derived (implied) from ***F***
  - Do not confuse closure of F with closure of an attribute set

Closure of F

- Example: Assume R(A, B, C, D), with F = {A→ B, B→C). Then F^+ includes the following FDs:

$$A \to A,\ A \to B,\ A \to C,\ B \to B,\ B \to C,\ C \to C,\ D \to D,$$
$$AB \to A,\ AB \to B,\ AB \to C,\ AC \to A,\ AC \to B,\ AC \to C,$$
$$AD \to A,\ AD \to B,\ AD \to C,\ AD \to D,\ BC \to B,\ BC \to C,$$
$$BD \to B,\ BD \to C,\ BD \to D,\ CD \to C,\ CD \to D,$$
$$ABC \to A,\ ABC \to B,\ ABC \to C,\ ABD \to A,\ ABD \to B,$$
$$ABD \to C,\ ABD \to D,\ BCD \to B,\ BCD \to C,\ BCD \to D,$$
$$ABCD \to A,\ ABCD \to B,\ ABCD \to C,\ ABCD \to D.$$

Part II:
Schema Decomposition

Relational Schema Design

- Goal of relational schema design is to avoid redundancy, and the anomalies it enables.
  - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences have been updated.
  - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

Result of bad design: Anomalies

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete’s	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Example of Bad Design

Suppose we have FDs name -> addr, favBeer and beersLiked -> manf. This design is bad:

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the FDs.

Goal of Decomposition

- Eliminate redundancy by decomposing a relation into several relations

- Check that a decomposition does not lead to bad design

FDs and redundancy

Given relation R and FDs F

- R often exhibits anomalies due to redundancy
- F identifies many (not all) of the underlying problems

Idea

- Use F to identify "good" ways to split relations
- Split R into 2+ smaller relations having less redundancy
- Split F into subsets which apply to the new relations (compute the projection of functional dependencies)

Schema decomposition

- Given relation R and FDs F
  - Split R into R_i s.t. for all i $R_i \subset R$ (no new attributes)
  - Split F into F_i s.t. for all i, F entails F_i (no new FDs)
  - F_i involves only attributes in R_i
- Caveat: entirely possible to lose information
  - F^+ may entail FD f which is not in $(\cup_i F_i)^+$

  => Decomposition lost some FDs
  - Possible to have $R \subset \bowtie_i R_i$

  => Decomposition lost some relationships
- Goal: minimize anomalies without losing info

Good Properties of Decomposition

1) Lossless Join Decomposition
   - When we join decomposed relations we should get ***exactly*** what we started with
2) Avoid anomalies
   - Avoid redundant data
3) Dependency Preservation
   - $(F_1 \cup \ldots \cup F_n)^+ = F^+$

Problem with Decomposition

Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss

Example: Splitting Relations

Student_Name	Student_Email	Course	Instructor
Alice	alice@gmail	SE 3DB3	Chiang
Alice	alice@gmail	CS 3SH3	Zheng
Bob	bob@mcmaster	SE 3RA3	Janicki
Laura	laura@gmail	SE 3DB3	Jones

Students (email, name)

Courses (code, instructor)

Taking (email, courseCode)

Students $\bowtie$ Taking $\bowtie$ Courses has additional tuples!

- (Alice, alice@gmail, SE3DB3, Jones), but Alice is not in Jones' section of SE 3DB3
- (Laura, laura@gmail, SE3DB3, Chiang), but Laura is not in Chiang's section of SE 3DB3

Why did this happen? How to prevent it?

Information loss with decomposition

- Decompose R into S and T
  - Consider FD A->B, with A only in S and B only in T
- FD loss
  - Attributes A and B no longer in same relation

  => Must join T and S to enforce A->B (expensive)
- Join loss
  - Neither $(S \cap T) \rightarrow S$ nor $(S \cap T) \rightarrow T$ in F^+

  => Joining T and S produces extraneous tuples

Lossless Join Decomposition

- A decomposition should not lose information
- A decomposition ($\mathbf{R}_1,\ldots,\mathbf{R}_n$) of a schema, **R**, is **lossless** if every valid instance, r, of **R** can be reconstructed from its components:
  - $r = r_1 \bowtie \ \ldots \ \bowtie \ r_n$ where $r_i = \Pi_{R_i}(r)$

Lossy Decomposition

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

$r_1 \bowtie r_2$

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

What is lost?

- Lossy decomposition
  - Loses the fact that (12, Jen) lives at 2 Pine (not 3 Oak)
  - Loses the fact that (13, Jen) lives at 3 Oak
- Remember: lossy decompositions yield ***more*** tuples than they should when relations are joined together

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

Example 2

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

R1

Model Name	Category
a11	Canon
s20	Nikon
a70	Canon

R2

Price	Category
100	Canon
200	Nikon
150	Canon

Ack: S.M. Lee

Example 2 (cont'd)

R1 ⋈ R2

Model Name	Price	Category
a11	100	Canon
a11	150	Canon
s20	200	Nikon
a70	100	Canon
a70	150	Canon

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

Lossy decomposition

- Additional tuples are obtained along with original tuples
- Although there are more tuples, this leads to less information
- Due to the loss of information, the decomposition for the previous example is called lossy decomposition or lossy-join decomposition

Testing for Losslessness

- A (binary) decomposition of **R** = (R, **F**) into **R**1= (R1, **F**1) and **R**2 = (R2, **F**2) is lossless if and only if:
  - either the FD (R1 $\cap$ R2) → R1 is in **F**+
  - or the FD (R1 $\cap$ R2) → R2 is in **F**+

  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R1 OR
  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R2

Decomposition Property

- In our example
  - *Name* $\not\to$ ID,*Name*
  - *Name* $\not\to$ *Name, Addr*
- *A lossless decomposition*
  - *[ID,Name] and [ID,Addr]*
- Example 2:
  - *Category* $\not\to$ ModelName, Category
  - *Category* $\not\to$ Price, Category
  - Better to use [MN, Category] and [MN, Price]
- In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless decomposition

Lossless Decomposition

A decomposition is lossless if we can recover:

R(A, B, C)

Decompose

R1(A, B) R2(A, C)

Recover

R'(A, B, C)

Thus, R' = R

Example : Lossless Decomposition

Given:

Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)

FDs:

branch-name $\longrightarrow$ *branch-city, assets*

loan-number $\longrightarrow$ *amount, branch-name*

Decompose Lending-schema into two schemas:

Branch-schema = (*branch-name, branch-city, assets*)

Loan-info-schema = (*branch-name, customer-name, loan-number, amount*)

Example : Lossless Decomposition

Show that the decomposition is a Lossless Decomposition

Branch-schema = (*branch-name*, *branch-city*, *assets*)

Loan-info-schema = (*branch-name*, *customer-name*, *loan-number*, *amount*)

- Since *Branch-schema* ∩ *Loan-info-schema* = {*branch-name*}
- We are given: *branch-name* $\longrightarrow$ *branch-city*, *assets*

Thus, this decomposition is lossless.

Projecting FDs

- Once we've split a relation, we have to re-factor our FDs to match
  - Each FDs must only mention attributes from one relation
- Similar to geometric projection
  - Many possible projections (depends on how we slice it)
  - Keep only the ones we need (minimal basis)