CSC 370

Database Systems: Towards a Design Theory for Relational Models

Major topics

- **Functional dependencies**
  - Mathematical notation for describing relationships amongst relations
- **Decompositions**
  - Breaking larger relations into sets of smaller relations
- **Normal forms**
  - Using functional dependencies + decompositions in order to prevent data anomalies

Functional dependencies

- $X \to Y$
  - This is an assertion about some relation R
  - Whenever two tuples of R agree on all attributes of set X...
  - ... they must also agree on attributes in set Y.
  - (Note that there may be more attributes in R than in $X \cup Y$.)
- Pronounced: "$X \to Y$ holds in R"
- Our notational convention:
  - X, Y, Z: Sets of attributes
  - A, B, C: Single attributes
  - We write ABC rather than {A, B, C}

Simplification: Splitting FD right-hand sides

- $X \rightarrow A_1A_2...A_n$ holds for R exactly when all the following hold for R:
  - $X \rightarrow A_1$
  - $X \rightarrow A_2$
  - ...
  - $X \rightarrow A_n$
- Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$
- (There is no splitting rule for left-hand sides!)
- In general, we will express FDs with singleton right sides.

Example of an FD

- **Patrons(name, addr, beersLiked, manf, favBeer)**
- Some reasonable FDs we could assert
  - name $\rightarrow$ addr favBeer
  - beersLiked $\rightarrow$ manf
- Note: beersLiked in the relation schema suggests there may be multiple entries for a patron in the table

Example: Patron (with sample data)

name	addr	beersLiked	manf	favBeer
Norm	Dallas Road	Blue	Labatts	Bud Light
Norm	Dallas Road	Bud Light	Anheuser-Busch	Bud Light
Cliff	Myrtle Ave	Blue	Labatts	Blue

Because
name → addr

Because
beersLiked → manf

Because
name → favBeer

Keys of relations

- Assume K is a set of attributes from relation R.
- K is a **superkey** for relation R if K functionally determines all of R
  - (i.e., determines all attributes in R)
- K is a **key** for R if:
  - K is a superkey and...
  - There exists no proper subset of K that is also a superkey.
  - Put differently, if $\{A_1, A_2, \ldots A_n\}$ is K, then it is impossible for two distinct tuples in R to agree on all of $A_1, A_2, \ldots A_n$.

Example: superkey

- **Patrons(name, addr, beersLiked, manf, favBeer)**
- {name, addr, beersLiked}
  - Forms a superkey because together these attributes determine all other attributes
  - name $\rightarrow$ addr favBeer
  - beersLiked $\rightarrow$ manf

Example: key

- **Patrons(name, addr, beersLiked, manf, favBeer)**
- {name, beersLiked}
  - Forms a key because neither {name} or {beersLike} is a superkey
  - i.e., names doesn't imply manf
  - i.e., beersLiked doesn't imply addr
- There are no other keys, but there are lots of superkeys
  - Any set of R's attributes that includes {name, beersLiked}

"Mommy, where do keys come from?"

1. Sometimes we simply assert the fact of a key K on the data
   - The only FD becomes K $\rightarrow$ Z for all attributes Z in the relation R
2. Sometimes we assert the FDs and then try to deduce the keys by systematic exploration
3. Sometimes the facts of life intervene. Example:
   - "No two courses can meet in the same room at the same time" implies: hour room $\rightarrow$ course

- (Wrinkle: Relations can have more than one key...)

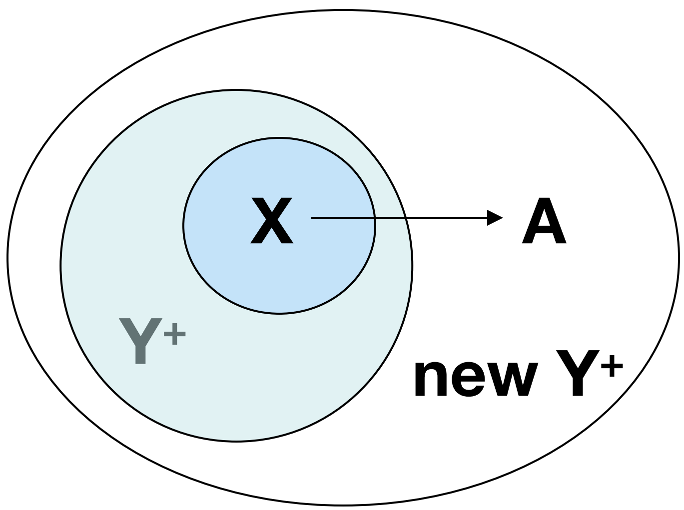
Inferring FDs

- There exist rules for reasoning about functional dependencies
- Sometimes we would like to determine whether or not a specific set of FDs infers another
- Example:
  - We are given FDs $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, ..., $X_n \rightarrow A_n$
  - We want to know whether or not $Y \rightarrow B$ must hold in any relation satisfying the given FDs
- Example:
  - If $A \rightarrow B$ and $B \rightarrow C$ holds, it must be the case $A \rightarrow C$ holds even if this last FD is not part of the FD set.
- We need rules of inference in order to design sets of good relation schemas...

Closure test

- This test is a straightforward way to check if an FD (that is not part of the existing set of FDs) is valid
  - "Valid" implies the FD is supported by the existing set of FDs
- Recall:
  - We have FDs $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, ..., $X_n \rightarrow A_n$
  - We want to know if $Y \rightarrow B$ also holds
- To do this, we **compute the closure of Y (denoted Y^+)**
- Technique:
  - Basis: $Y^+ = Y$
  - Induction: Look at a given FDs left side X that is a subset of the current Y^+. If the FD is $X \rightarrow A$, add A to Y^+
  - Repeat inductive step until there are no more rules $X \rightarrow A$ that change Y^+

Visualization: How Y^+ grows



Finding all implied FDs

- Motivation:
  - We eventually want to apply normalization to our schema.
  - (This is the process by which we break a relation schema into two or more schemas.)
- Example:
  - Schema: ABCD
  - FDs: $AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$
  - Possible decomposition: ABCD becomes ABC and AD.
  - Question: Is $C \rightarrow A$ an FD that would be valid on ABC?

Why? Here's one way to look at it...

ABCD $a_1b_1cd_1$ $a_2b_2cd_2$

comes from

ABC a_1b_1c a_2b_2c

$d_1 = d_2$ because

$\mathbf{C} \rightarrow \mathbf{D}$

$a_1 = a_2$ because

$\mathbf{D} \rightarrow \mathbf{A}$

Thus, tuples in the projection with equal Cs have equal As

$\mathbf{C} \rightarrow \mathbf{A}$

Basic idea

- Start with given FDs and find all non-trivial FDs that follow from the given FD
  - Non-trivial $\Rrightarrow$ right side not contained in the left
- Restrict ourselves to examining those FDs involving only attributes of the projected schema

Simple exponential algorithm

- This is for finding all implied FDs for a schema
  - (Sometimes also called "projecting FDs" when we project onto a schema with a subset of the original attributes.)

1. For each set of attributes X, compute X^+
2. Add $X \rightarrow A$ for all A in ($X^+ - X$)
3. However, drop $XY \rightarrow A$ whenever we find $X \rightarrow A$
   - Why? Because $XY \rightarrow A$ will follow from $X \rightarrow A$ in any projection (i.e., the Y attributes add no new information).
4. Finally, use only FDs involving projected attributes

Some corners we can cut

- No need to compute the closure of the empty set or of the set of all attributes
- If we find X^+ equals all attributes, so is the closure of any superset of X

Example: Projecting FDs

- Assume the following schema and FDs
  - Schema: ABC
  - FDs: $A \rightarrow B$, $B \rightarrow C$
- We want to project onto AC
- Computing closure:
  - $A^+ = ABC$; yields FDs $A \rightarrow B$, $A \rightarrow C$
  - We need not compute AB^+ or AC^+
  - $B^+ = BC$; this yields $B \rightarrow C$
  - $C^+ = C$; this yields nothing non-trivial
  - $BC^+ = BC$; this yields nothing non-trivial

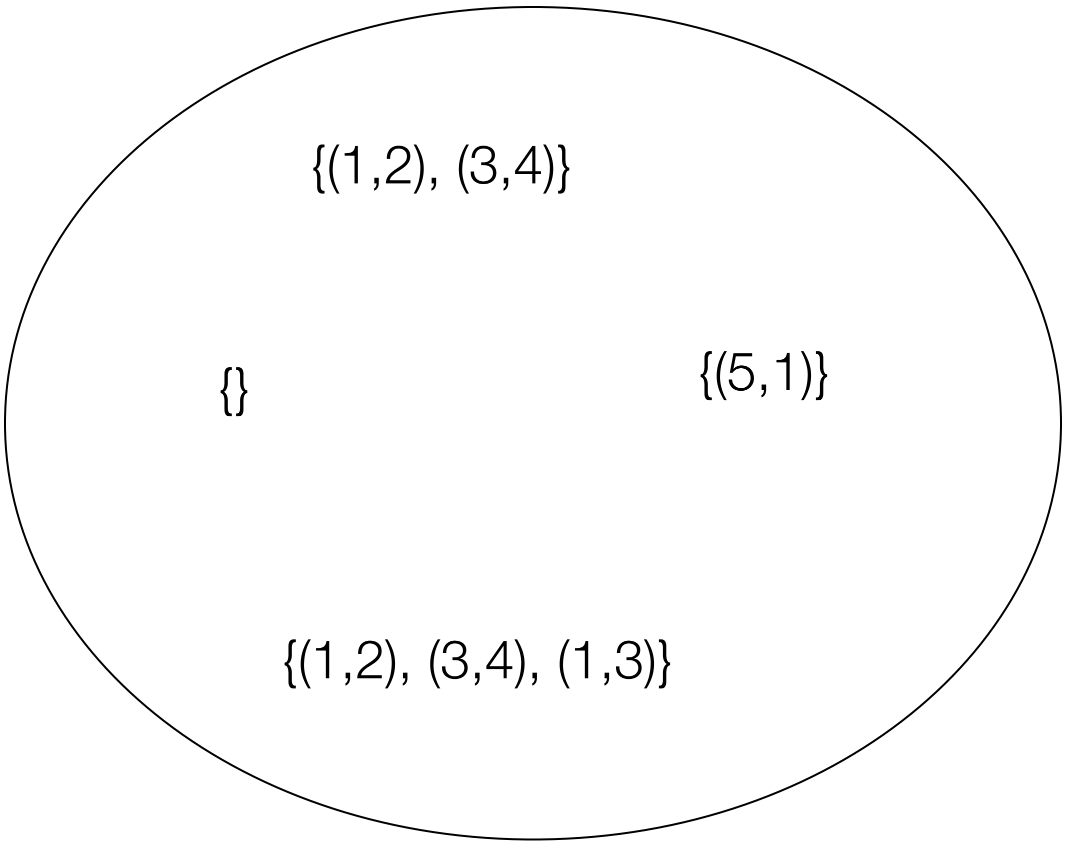
Example: Projecting FDs (continued)

- Resulting FDs
  - $A \rightarrow B$
  - $A \rightarrow C$
  - $B \rightarrow C$
- The FDs that project onto AC:
  - $A \rightarrow C$
  - That is, this single FD is the only one that involves a subset of attributes {A, C}

Spatial Metaphor for FDs

- Imagine the set of **all possible instances** of a particular relation
  - (Ignore for now the FDs proper to the relation)
- Put differently, we consider all finite sets of tuples having the proper number of components dictated by the schema
- Each instance is a point in some space

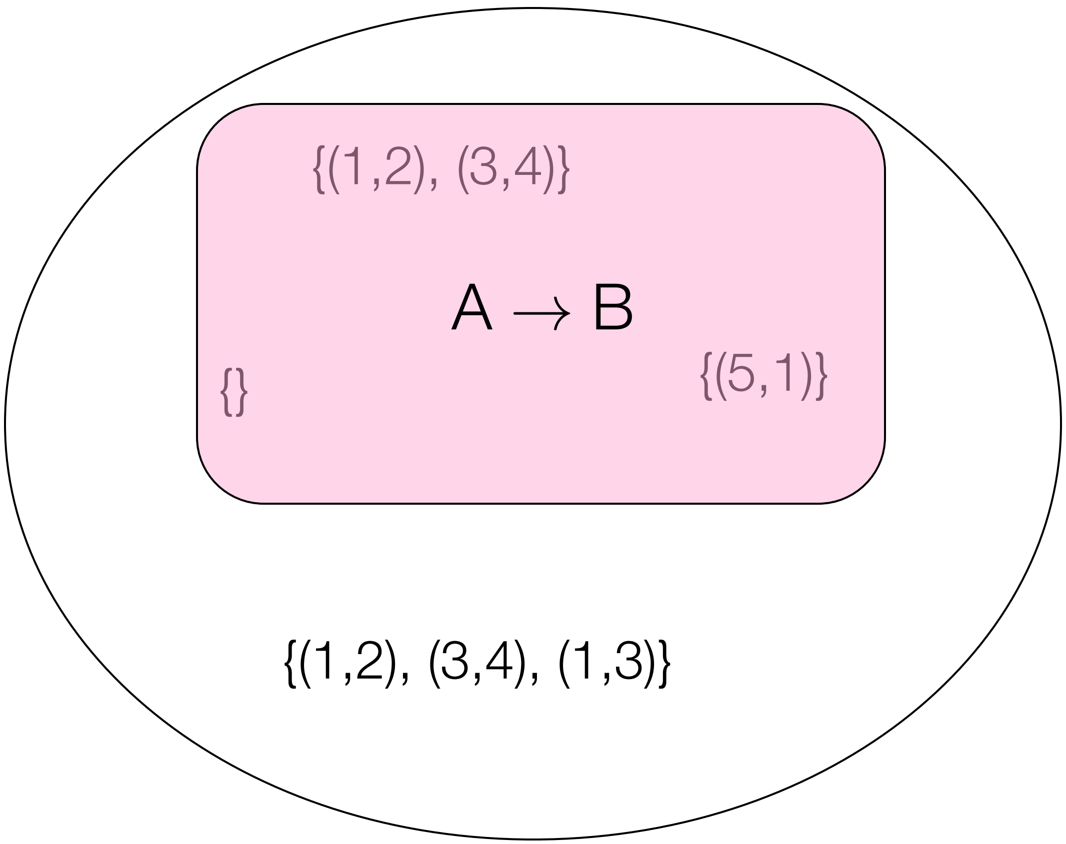
Example: R(A,B)



Spatial Metaphor for FDs (continued)

- A functional dependency is a **subset of the relation instances** in our space (i.e., a subset of points)
- For each FD $X \rightarrow A$ there is a subset of instances satisfying the FD
- We can therefore represent an FD as a **region in the space containing properly restricted relation instances**
- Trivial FD implies an FD represented by the entire space!

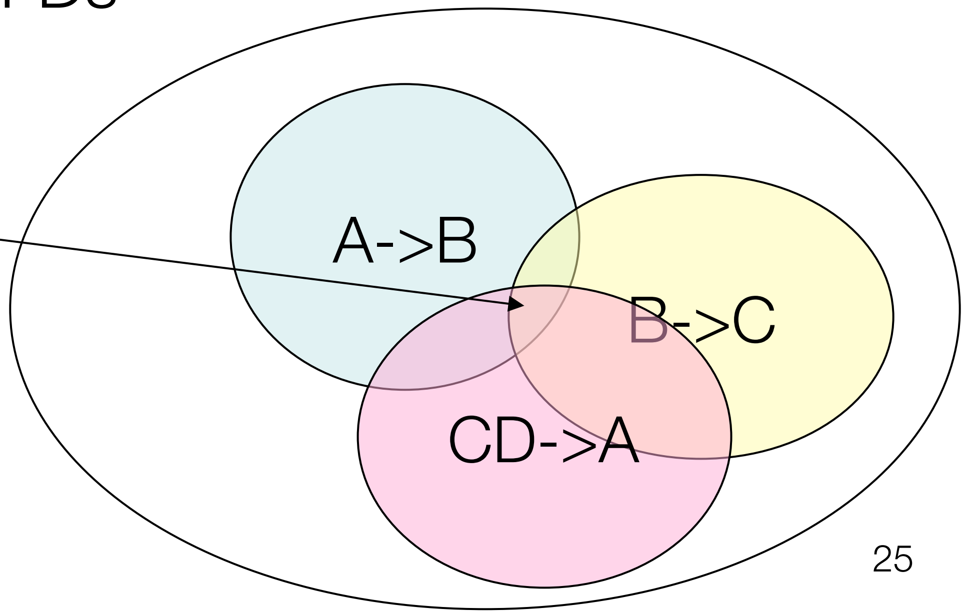
Example: $A \rightarrow B$ for $R(A,B)$



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**
- That is, the intersection is the set of all instances that satisfy all of the FDs

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



More implications of FDs

- Suppose $Y \rightarrow B$ follows from FDs $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, ... $X_n \rightarrow A_n$
  - Then the region in the space of instances for $Y \rightarrow B$ must include the intersection of the regions for all $X_i \rightarrow A_i$
  - That is, every instance satisfying all FDs $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$
  - However, **an instance could satisfy $Y \rightarrow B$ yet not be in the intersection**!

Relational Schema Design

- Goal of relational schema design is to **avoid redundancy** and **anomalies.**
- **Update anomaly**:
  - One occurrence of a fact is changed, but not all occurrences are changed.
- **Deletion anomaly**:
  - A valid fact is lost when a tuple is deleted.
- Let's look first at redundancy.

Example of bad design: **Patrons**

name	addr	beersLiked	manf	favBeer
Norm	Dallas Road	Blue	Labatts	Bud Light
Norm	???	Bud Light	Anheuser-Busch	???
Cliff	Myrtle Ave	Blue	???	Blue

- **Patrons(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)**
- Some data is **redundant**:
  - Each of the ???s can be figured out by using one of two FDs

  1. name $\rightarrow$ addr favBeer
  2. beersLiked $\rightarrow$ manf

Patrons also exhibits anomalies

name	addr	beersLiked	manf	favBeer
Norm	Dallas Road	Blue	Labatts	Bud Light
Norm	Dallas Road	Bud Light	Anheuser-Busch	Bud Light
Cliff	Myrtle Ave	Blue	Labatts	Blue

- Update anomaly
  - If Norm moves to Shelbourne, will we remember to change each of his tuples?
- Deletion anomaly:
  - If nobody likes Blue, we lose of the track of that fact that Labatt's manufactures Blue.

Normal Forms

- These are used to characterize schema decompositions
  - As we proceed through the normal forms, fewer and fewer kinds of anomalies are present
- Simple normal forms
  - First Normal Form (1NF): All components of all tuples are atomic (i.e., components do not have structure)
  - Second Normal Form (2NF): There exists no FD $X \rightarrow A$ such that (1) X is a subset of a key and (2) A is not in the key (i.e., A is non-prime)
- We will examine:
  - **Boyce-Codd Normal Form**
  - **Third Normal Form (3NF)**
  - **Fourth Normal Form (4NF)**

Boyce-Codd Normal Form (BCNF)

- **A relation R is in BCNF if**:
  - for each $X \rightarrow Y$ that is both a non-trivial FD and holds in R
  - ... it is the case that X is a superkey.
- Recall: nontrivial means the set of attributes Y is not contained in the set of attributes X.
- Also recall: A superkey is any superset of a key.

Example 1: BCNF

- **Patrons(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)**
- FDs:
  - **name** $\rightarrow$ **addr favBeer**
  - **beersLiked** $\rightarrow$ **manf**
- Only key is {name, beersLiked}
- Is the relation in BCNF?
  - Each FD has a left side that is **not a superkey**
  - Therefore each of the FDs demonstrates that Patrons **is not in BCNF**

Example 2: BCNF

- **Beers(<u>name</u>, brewer, addr)**
- FDs:
  - name $\rightarrow$ brewer
  - brewer $\rightarrow$ addr
- Only key is {name} (Why?)
- Is the relation in BCNF?
  - **name** $\rightarrow$ **brewer** does not violate BCNF
  - However, **brewer** $\rightarrow$ **addr** does violate BCNF
  - Therefore schema is not in BCNF

Idea: Transform schema in BCNF

- So far we have seen schemas that are not in BCNF (given the associated FDs)
- We want to decompose the schema such that the smaller ones are in BCNF
  - That is, we guide our decomposition such that all schemas (and FDs that apply to them) are in BCNF
- So now let us look closer at decompositions...

Decomposition into BCNF: Basic idea

- We are given:
  - Relation R
  - A set of FDs named F
- Look among given FDs for a BCNF violation involving $X \to Y$
  - (And if any FD that can be inferred from F violates BCNF, then there will be an FD in F itself that violates BCNF.)
- Compute X^+
  - We don't expect X^+ to consist of all attributes, otherwise we would have already had X as a superkey!

Basic idea continued...

- Replace R with two new schemas
  A. $R_1 = X^+$
  B. $R_2 = R - (X^+ - X)$
- Project each FD in the set F onto the two new relations
- Check to ensure that both of the resulting relations are in BCNF
  – If one is still not in BCNF, then decompose that one following a similar procedure.

Decomposition Visualization