INFO2120 – INFO2820 – COMP5138 Database Systems

Week 6: Schema Normalization

(Kifer/Bernstein/Lewis – Chapter 6; Ramakrishnan/Gehrke – Chapter 19; Ullman/Widom – Chapter 3)

Dr. Uwe Röhm
School of Information Technologies

Outline

- **Motivation**
- **Functional Dependencies and Normal Forms**
  - 1st and 2nd normal form
  - 3rd normal form
  - BCNF
- **Table Decompositions**
  - Lossless-join and dependency preserving
- **Making it precise**

Based on slides from Kifer/Bernstein/Lewis (2006) "Database Systems" and from Ramakrishnan/Gehrke (2003) "Database Management Systems", and also including material from Röhm.

Schema Design Process

- The relational schema is best obtained by starting with a conceptual design (eg. an E-R model)
  - This can be converted to relational schema
- However, the relational schema may arise in other ways
  - eg. start from data in a spreadsheet
    - Typically gives one wide table!
  - eg. choose tables by raw intuition
- We should evaluate the schema, and improve it if necessary

Motivating Example

- Example: *Assume a direct data import from an Excel worksheet*

Mining Data Collection					
mine	**state**	**commodity**	**abbrv**	**company**	**homepage**
Olympic Dam	SA	Uranium	U	BHP Billiton	www.bhpbilliton.com
Blair Athol	QLD	Coal	Cbl	Rio Tinto	www.riotinto.com
Hunter Valley	NSW	Coal	Cbl	Rio Tinto	www.riotinto.com/index.asp
Hunter Valley	NSW	Coal	Cbl	Coal and Allied	www.coalandallied.com.au
Mt Pleasant	WA	Gold	Au	NULL	NULL

Inconsistent Information

Redundant Information

Incomplete Information

- There are "better" and "worse" relational schemas; How can we judge the quality of relational schemas?

Evaluation of a DB Design

- The most important requirement is **adequacy**: that the design should allow representing all the important facts about the design
  - Make sure every important process can be done using the data in the database, by joining tables as needed
  - eg. “can we find out which driver made a particular delivery?”
- If a design is adequate, then we seek to **avoid redundancy** in the data
  - and at a side effect, being able to insert/update/delete information without the need for (extensive) use of null values

  (Redundant data is where the same information is repeated in several places in the db)

Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage
  - **Insertion Anomaly:**
    Adding new rows forces user to create duplicate data or to use *null* values.
  - **Deletion Anomaly:**
    Deleting rows may cause a loss of data that would be needed for other future rows!
  - **Update Anomaly:**
    Changing data in a row forces changes to other rows because of duplication.
- Note: It is the anomalies with modifications that are the serious concern, not the extra space used in storage

Anomalies Example

Mining Data Collection						
mine	**state**	**commodity**	**abbrv**	**capacity**	**company**	**homepage**
Olympic Dam	SA	Uranium	U	100	BHP Billiton	www.bhpbilliton.com
Blair Athol	QLD	Coal	Cbl	920	Rio Tinto	www.riotinto.com
Hunter Valley	NSW	Coal	Cbl	1430	Rio Tinto	www.riotinto.com/index.asp
Hunter Valley	NSW	Coal	Cbl	1430	Coal & Allied	www.coalandallied.com.au
Mt Pleasant	WA	Gold	Au	76	NULL	NULL

Question – Is this a relation?

Answer – Yes: unique rows and no multivalued attributes

Question – What's the primary key?

Answer – Composite: (Mine,Company) (but then no NULL values allowed!)

Question – What happens with data modifications?

Anomalies in Previous Example

- Insertion Anomaly:
  - If another company buys a stake into an existing mine, we have to re-enter the 'mine' information, causing duplication.
  - What if we want to insert a mine which has no owner so far?
We either cannot do it at all (PK!) or we get many *NULL* values.
- Deletion Anomaly:
  - If we delete all Gold mines, we loose the information that 'Au' is the chemical identifier for the commodity 'Gold'!
  - Or if composite PK, we cannot delete the last company for a mine!
- Update Anomaly:
  - For changing, e.g., the *homepage* of a company, we have to update multiple tuples.

Why do these anomalies exist here?

Because there are two themes (entity types) placed into one relation.
This results in duplication and an unnecessary dependencies

Today's Agenda

- Motivation
- **Functional Dependencies and Normal Forms**
  - 1st and 2nd normal form
  - 3rd normal form
  - BCNF
- **Table Decompositions**
  - Lossless-join and dependency preserving
- Making it precise

Textbook, Chapter 19

Functional Dependency (FD)

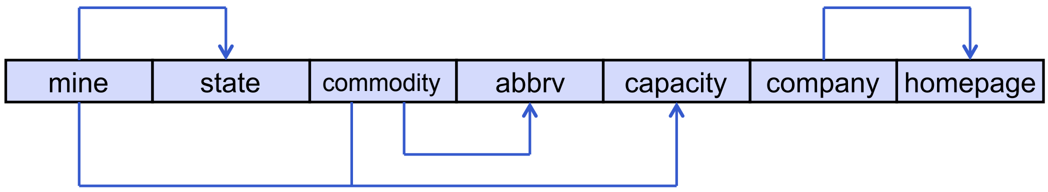
- Domain constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
- **Functional Dependency**: The value of one attribute (the determinant) determines the value of another attribute
  - Intuitively: "If two tuples of a relation R agree on values in X, then they must also agree on the Y values."
- We write $X \rightarrow Y$
  - "X (functionally) determines Y"
  - "Y is functionally dependent on X"

FD Example

- FDs in motivating example:
  - *mine* → *state*
  - *commodity* → *abbrv*
  - *mine commodity* → *capacity*
  - *company* → *homepage*

in general, a mine can produce several commodities.

- Graphical notation:



We draw a line with arrowheads
going to the dependent column(s)
from the column(s) that are depended on

- Q: Which FD do not hold in this example?

Some Remarks

- A FD is an assertion about the schema of a relation not about a particular instance.
  - It must be fulfilled by all allowable relations.
- If we look at an instance, we cannot tell for sure that a FD holds
  - The most insight we can gain by looking at a "typical" instance are "hints"…
  - We can however check if the instance violates some FD
- FDs must be identified based on the semantics of an application.

Keys and Functional Dependencies

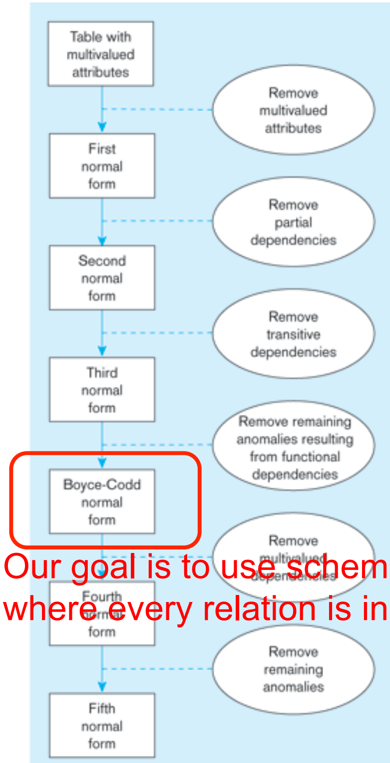
- If you know the functional dependencies, then you can check whether a column (or set of columns) is a ***key*** for the relation
  - Does the column/set determine every column?
  - Can we have rows which are the same in the column/set, but different somewhere?
- There may be several different ways to choose a column/set of columns as key for a relation
  - A column/set is called a ***candidate key*** if it's values are necessarily different among the rows
- Choose one *candidate key* as the ***primary key***
  - Used as identifier to capture relationships, and stored in other tables as foreign key
- A "*superkey*" is a column or set of columns that includes a *candidate key*
  - A *candidate key*, plus perhaps extra columns

Schema Normalization

- FDs can be used to identify schemas with problems and to suggest refinements.
- Main Idea: Only allow FDs of form of key constraints.
  - Each non-key field is functionally dependent on every candidate key
- **Schema Normalization:** The process of validating and improving a logical design so that it satisfies certain constraints (*Normal Forms*) that avoid unnecessary duplication of data
  - Idea: decompose relations with anomalies to produce smaller, *well-structured* relations
- Note: Using the Mapping Rules from week 3, we already get very close to a fully normalised schema.
  - But to be sure we have to check…

Normal Forms

- Database theory identifies several normal forms for relational schemas.
- Each normal form is characterized by a set of restrictions.
- For a relation to be in a normal form it must satisfy the restrictions associated with that form.
  - several NFs are used*; we focus on* ***BCNF***



Our goal is to use schemas where every relation is in BCNF

[Hoffer/Prescott/McFadden, 2005]

First and Second Normal Form

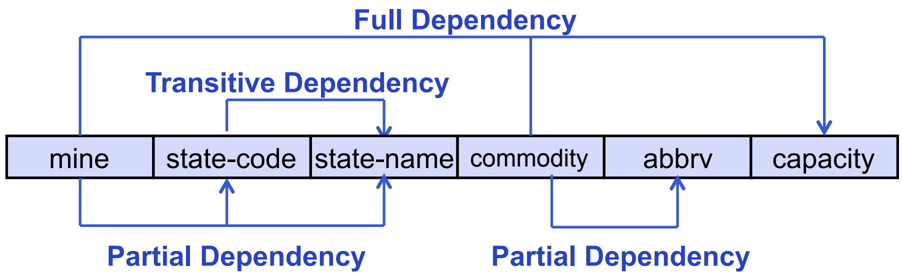
- Recall from third week:
  A relation *R* is in **first normal form (1NF)** if the domains of all attributes of *R* are *atomic*.
- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - multivalued attributes, composite attributes
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data
- **Second Normal Form (2NF)** more of history value…
  - 1NF + every non-key attribute is fully functionally dependent on the primary key
  - This means: **No partial dependencies**
    (no FD $X \rightarrow Y$ where X is a strict subset of some key)

Third Normal Form (3NF)

- 2NF + **no transitive dependencies** (functional dependencies between non-primary-key attributes)
  - No FD of the form $X \rightarrow Y$, where X is not a key and Y is not at least a subsets of a key
- Note: this is called transitive, because the primary key is a determinant for another attribute, which in turn is a determinant for a third *non-key* attribute
- Solution: non-key determinant with transitive dependencies go into a new table; non-key determinant becomes primary key in the new table and stays as foreign key in the old table

Normalisation Example

Mining relation (simplified):



- What the Key?
- In which Normal Form is this relation?
- Which anomalies could occur?

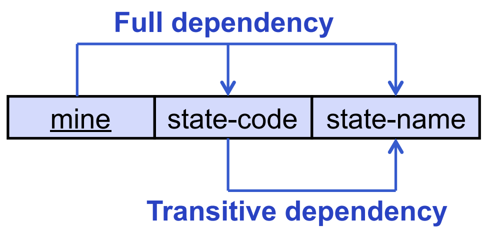
Table Decomposition

- Suppose that relation R contains attributes $A_1 \ldots A_n$.
  A *decomposition* of *R* consists of replacing *R* by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of *R* (and no attributes that do not appear in *R*), and
  - Every attribute of *R* appears as an attribute of one of the new relations.
  - All new relations differ.
- **Central Idea of Normalisation:**
  Decompose along a functional dependency.
  - **If *X -> Y* violates a normal form, decompose *R* into *R-Y* and *XY*.**
- Example:
  R (*A, B, C, D*) with FDs: {*A -> B D* and *B -> C*}

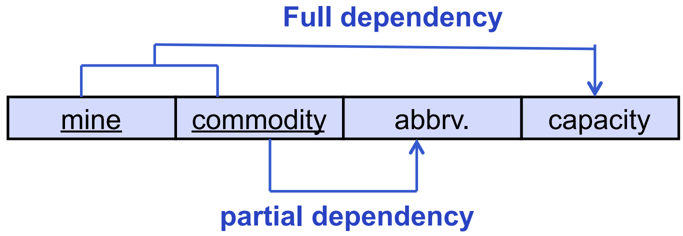
Example Decomposition 1

Remove 1. partial dependency (*mine → state-code state-name*)

Mine



MineProduction



There is still a partial dependency in **MineProduction**

Example Decomposition 2

Remove 2. partial dependency (*commodity* → *abbrv*):

Mine

full dependency

mine	state-code	state-name

transitive dependency

Commodity

full dependency

commodity	abbrv.

MineProduction

full dependency

mine	commodity	capacity

There is still a transitive dependency in **Mine**

Example: Final Decomposition

Remove Transitive Dependencies:

Mine

Full Dependency

mine	state-code

State

Full Dependency

state-code	state-name

Commodity

Full Dependency

commodity	abbrv.

MineProduction

Full Dependency

mine	commodity	capacity

In which normal form are these relations?

Boyce-Codd Normal Form (BCNF)

- When a relation has more than one candidate key, even 3NF relations may show anomalies.
  - Hence we can decompose one step further
- A relation *R* is in <u>BCNF</u> if the only non-trivial FDs that hold over *R* are key constraints.
  - Formal:

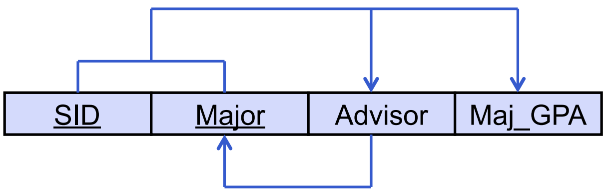
    $$\forall \textit{ non-trivial } X \rightarrow Y\text{: } X \textit{ is a superkey}$$

  - All determinants are superkeys… the only nontrivial FDs are those in which a key functionally determines one or more attributes.
  - "All attributes describe the key,the whole key,and nothing but the key"

Example: Relation 3NF, but not BCNF

Student-Advisor			
<u>SID</u>	<u>Major</u>	Advisor	Maj_GPA
123	Physics	Hawking	4.0
123	Music	Mahler	3.3
456	Literature	Mann	3.2
789	Music	Bach	3.7
678	Physics	Hawking	3.5

Relation in 3NF, but not BCNF

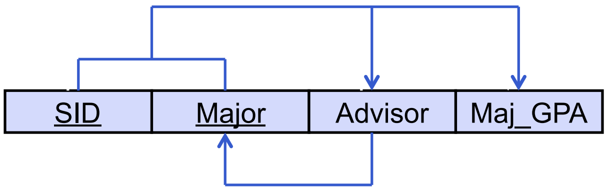


Functional Dependencies

Major is part of a key, hence no *transitive* dependency.

Example: Decomposition into BCNF

split along the
second FD



Overall Design Process

- Consider a proposed schema
- Find out application domain properties expressed as functional dependencies
- See whether every relation is in BCNF
- If not, use a bad FD to decompose one of the relations; start with partial dependencies
  - Replace the original relation by its decomposed tables
- Repeat the above, until you find that every relation is in BCNF

Evaluating a Decomposition

- A proposal to use decomposition on a schema should be evaluated
- Does it allow all the original information to be captured properly?
- Does it allow all the original application domain constraints to be captured properly?
- These issues are formalized as properties of a decomposition
  - It must be ***lossless-join***
  - We want it to be ***dependency-preserving***
- If you do the decomposition as described based on a bad FD, the decomposition is always *lossless-join*

Today's Agenda

- **Motivation**
- **Functional Dependencies and Normal Forms**
  - 1st and 2nd normal form
  - 3rd normal form
  - BCNF
- **Table Decompositions**
  - Lossless-join and dependency preserving
- **Making it precise**

Textbook, Chapter 19

Properties of Table Decomposition

- A decomposition of *R* into *S* and *T* is **lossless-join** w.r.t. a set of FDs *F* if, for every instance *R* that satisfies *F*:

$$\prod_S (R) \bowtie \prod_T (R) \;=\; R$$

  - I.e. the common attributes of *S* and *T* are a key of either *S* or *T*
- **Dependency-preserving**: If *R* is decomposed into *S* and *T*, then all FDs that were given to hold on *R* must also hold on *S* and/or *T*.
  - Dependency preserving does not imply lossless join
  - And vice-versa!
- *It is essential that all decompositions used to deal with redundancy be lossless!*

Examples for Dependency-Preserving

- The decomposition of Orders(*date, cid, pid, descr, price, weight, amount*) with F = {*date cid pid* → *amount, pid* → *descr price weight* } into
  - Ordered(*date*,*cid*,*pid*,*amount*) with *F1* = {*date cid pid* → *amount*} and
  - Product (*pid*,*descr*,*price*,*weight*) with *F2* = {*pid* → *descr price weight*}

Central Theorem of Schema Refinement

- Theorem:
  A relation R with schema $\boldsymbol{R}$ and a set of FDs *F*.
  Let X $\rightarrow$ Y a functional dependency with $X \cap Y = \varnothing$.
  Then is the decomposition of R into XY and $\boldsymbol{R}$ -Y a *lossless-join decomposition*.

- Every relation R with functional dependencies F can be decomposed into **3NF** relations, which is both *lossless* and *dependency-preserving*.

- For every relation R with set of FDs *F* exists a *lossless-join* decomposition into **BCNF** relations.

How to Identify Candidate Keys?

- An important step in schema normalization is the identification of candidate keys
- We can do so by:
  - Identifying all functional dependencies that hold on our data set
  - Then reasoning over those FDs using a set of rules to on how we can combine FDs to infer candidate keys
  - Or alternatively, using these FDs top verify whether a given set of attributes is a candidate key or not.

- To be able to do so, we first need to formalise what a FD actually is – and when it represents a key constraint

Formal Definition for FD

- Given a relation with schema $\boldsymbol{R}$, and two sets of attributes $X = \{X_1, \ldots, X_m\} \subseteq \boldsymbol{R}$ and $Y = \{Y_1, \ldots, Y_n\} \subseteq \boldsymbol{R}$.
  A **functional dependency (FD)** $X \rightarrow Y$ holds over relation schema $\boldsymbol{R}$ if, for every allowable instance R of $\boldsymbol{R}$:

$$\forall \ r, s \in R: \quad r.X = s.X \quad \Rightarrow \quad r.Y = s.Y$$

- A functional dependency $X \rightarrow Y$ *is said to be* ***trivial*** if $Y \subseteq X$

Formal Definition of a Candidate Key

- Main Idea: Only allow FDs of form of a *key constraint*
  - Each non-key field is functionally dependent on every candidate key
- Definition: **Superkey**
  Given a relation R with schema $\boldsymbol{R}$ and a set F of functional dependencies. A set of attributes $K \subseteq \boldsymbol{R}$ is a **superkey** for R if $K \rightarrow \boldsymbol{R} \in F^+$
  - where F^+ is the *attribute closure* of F
- Note that $K \rightarrow R$ does not require K to be minimal!
  - K is a **candidate key** if no real subset of K has above property.
  - A unique identifier. One of the candidate keys will become the PK

From FDs to Keys

- Candidate keys are defined by functional dependencies
- Consequently, FDs help us to identify candidate keys
- Two Approaches:
  - **Approach 1: Attribute Closure**
    Given a set of FDs, we can verify whether a set of attributes is a candidate key ('verifying an educated guess').
    - Tool: an algorithm called *The Chase*
  - **Approach 2: Closure of *F***
    Given a set *F* of FDs, we can deduce all additional FDs that hold on the schema. Candidate keys are then defined by some implied FDs where the whole schema is functionally depending on the key (and the key itself is minimal)
    - Tool: *Armstrong Axioms*

Approach 1: *The Chase*
Computing the Closure of Attributes

- Determining the **Close of Attributes (X^+)** with *The Chase:* Starting with the given set of attributes, one repeatedly expands the set by adding the right side of an FD as soon as the left side is present:

1. Initialise *result* with the given set of attributes: $X=\{A_1, \ldots, A_n\}$
2. Repeatedly search for some FD $A_1 A_2 \ldots A_m \rightarrow C$
   such that all $A_1, \ldots, A_m$ are already in the set of attributes *result*, but C is not.
   Add C to the set *result.*
3. Repeat step 2 until no more attributes can be added to *result*
4. The set *result* is the correct value of X^+

From the Attribute Closure to Keys

- The set of Functional Dependencies can be used to find candidate keys.
  - Rationale: $K \rightarrow R$ holds iff $K^+ = R$

- Hence to find all candidate keys for a relation R:
  - Look at each set of attributes K
  - Calculate the attribute closure K^+
  - If K^+ contains all columns, then K is a **superkey** (a superset of a candidate key)
  - The superkeys which are minimal are the **candidate keys**
  - Pick one candidate key to be the primary key

Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Check $(AG)^+$
  1. $result = AG$
  2. $result = ABG$ $\quad (A \rightarrow B$ and $A \subseteq AG)$
  2. $result = ABCG$ $\quad (A \rightarrow C$ and $A \subseteq ABG)$
  3. $result = ABCGH$ $\quad (CG \rightarrow H$ and $CG \subseteq ABCG)$
  4. $result = ABCGHI$ $\quad (CG \rightarrow I$ and $CG \subseteq ABCGH)$
- Is (AG) a candidate key?
  1. Is (AG) a super key? YES!
     1. Does $AG \rightarrow R$? *ie* Is $(AG)^+ = R$
  2. Is any subset of (AG) a superkey? NO!
     1. Does $A \rightarrow R$? *ie* Is $(A)^+ = R$
     2. Does $G \rightarrow R$? *ie* Is $(G)^+ = R$

Don't forget to test for minimality!

Exercise

- Suppose R(A,B,C,D,E) with FDs given as
  - F = {A → B, B → C, C D → E }
- What is $(AD)^+$?

- What is A^+?
- What is D^+?

- Is (AD) a candidate key for R?
- Can you find any other candidate keys?

Approach 2: Deducing more FDs

- Given some FDs, we can usually infer additional FDs:
  - Example:
    uos_code → *cpoints*, *cpoints* → *wload* implies *uos_code* → *wload*

- A FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.

- **F^+ : closure of F** is the set of all FDs that are implied by *F*

- **Armstrong's Axioms** (*X, Y, Z* are sets of attributes):

1. Reflexivity rule: If $X \subseteq Y$, then $Y \rightarrow X$
2. Augmentation rule: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
3. Transitivity rule: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ and $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ and $CG \rightarrow I$: this is called the "union rule"; it follows by
      - Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

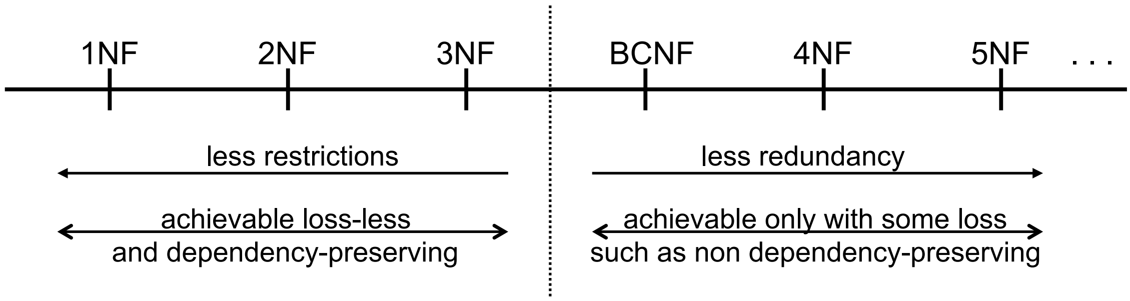
Reasoning about FDs

- Armstrong's Axioms are
  - Sound: they generate only FDs in ***F*⁺** when applied to a set F of FDs
  - Complete: repeated application of these rules will generate all FDs in the closure F^+
  - But computing F^+ is not efficient
- A couple of additional rules (that follow from Armstrong's Axioms.):
  4. ***Union rule***: If X → Y and X → Z, then X → YZ
  5. ***Decomposition rule***: If X → YZ, then X → Y and X → Z
  6. ***Pseudotransitivity rule:*** If X → Y and SY → Z, then XS → Z

- Example: Orders (*date,cid,pid,descr,price,weight,amount*) and FDs {*date cid pid* → *amount*, *pid* → *descr price weight* }.
- It follows:

date, cid, pid → pid	(reflexivity rule)
descr, price, weight → descr	(reflexivity rule)
pid → descr	(decomposition rule)
date, cid, pid → descr	(transitivity rule)

Summary



- Functional dependencies (FD): tool to detect redundancies in schemas
- Relations can be in different normal forms - the higher, the less redundancies. But there is a trade-off (see above).
  - If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to decompose into BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Decompositions can be loss-less and/or dependency-preserving
  - Must consider whether all FDs are preserved. If a dependency-preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.

References

- Kifer/Bernstein/Lewis (2nd edition)
  - Chapter 6
- Ramakrishnan/Gehrke (3rd edition - the ‘Cow’ book)
  - Chapter 19
- Ullmann/Widom (3rd edition of 1st Course in Database Systems)
  - Chapter 3

Next Lecture

- Integrity and Security
  - User Authorization, GRANT and REVOKE commands
  - Static Integrity Constraints, Assertions
  - Triggers

- Readings:
  - Kifer/Bernstein/Lewis book, Sections 3.2.2-3.3 and Chapter 7
  - <u>or</u> alternatively (if you prefer those books):
    - Ramakrishnan/Gehrke (Cow book), Sections 3.2-3.3 and Sections 5.7-5.9
    - Ullman/Widom, Chapter 7