ISYS2120: Data & Information Management

Week 6: Evaluating and improving relational schema, Schema Normalization

Alan Fekete

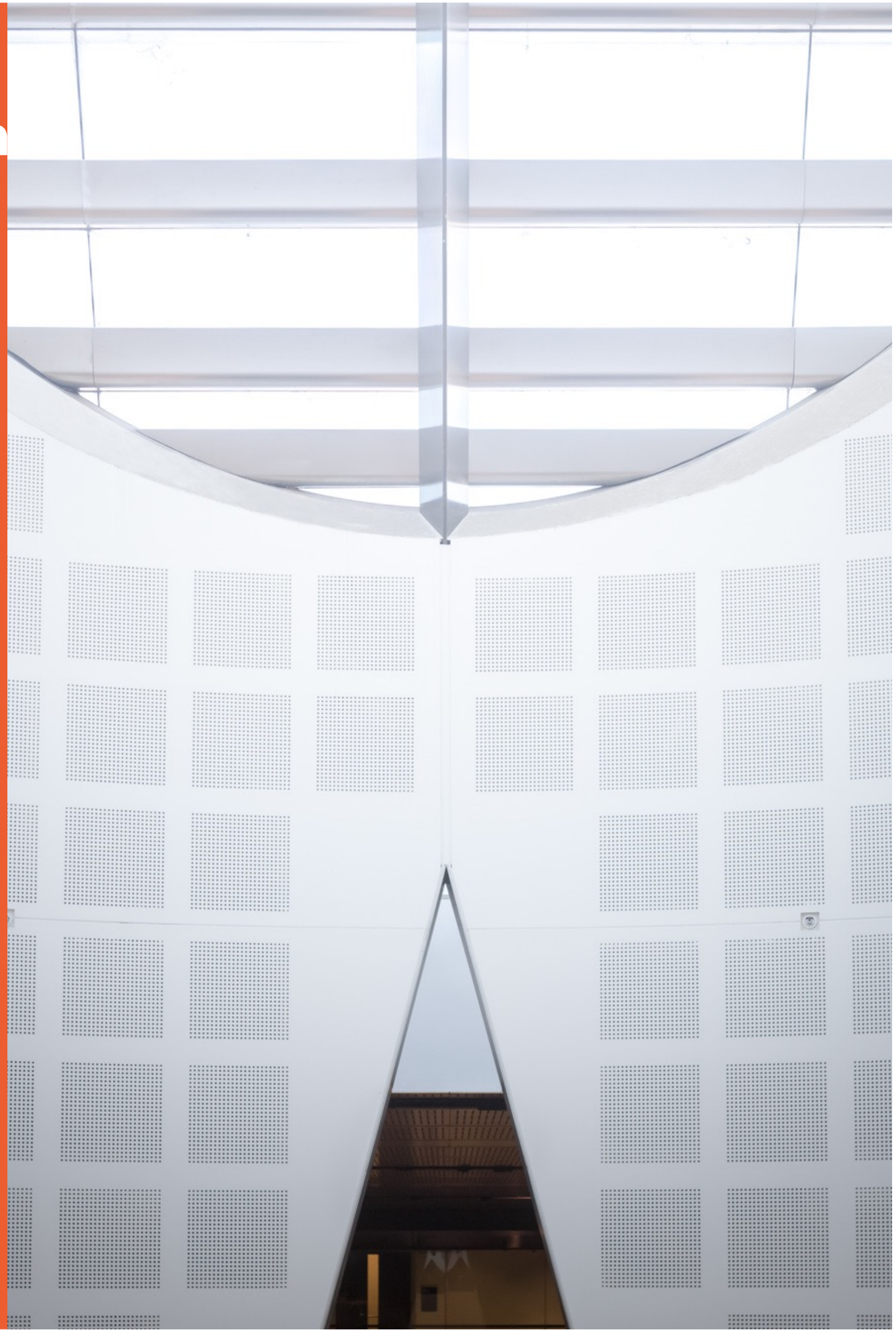
Based on slides from Kifer/Bernstein/Lewis (2006) "Database Systems"
and from Ramakrishnan/Gehrke (2003) "Database Management Systems",
and also including material from Fekete, Röhm.

Cf. Kifer/Bernstein/Lewis – Chapter 6

Ramakrishnan/Gehrke – Chapter 19;

› Silberschatz/Korth/Sudarshan - Chapter 7

THE UNIVERSITY OF SYDNEY



Today's Agenda

- Motivation
- Making it precise: Functional Dependency
- Making it precise: Normal Form
- Making it precise: Schema Decomposition

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**	**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
Blair Athol	QLD	Uranium	U	76	Rio Tinto	www.riotinto.com

Inconsistent Information

Redundant 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
Hunter Valley	NSW	Coal	Cbl	1430	Coal & Allied	www.coalandallied.com.au
Blair Athol	QLD	Uranium	U	76	Rio Tinto	www.riotinto.com

Use a version with inconsistency corrected

Question – Is this a relation?

Answer – Yes: unique rows and no multivalued attributes

Question – What's the primary key?

Answer – Composite: (Mine, Commodity, Company)

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/state' information, also "commodity/abbrv/capacity", 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 Uranium mines, we loose the information that 'U' is the chemical identifier for the commodity 'Uranium'!
  - 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 multiple themes (entity types) placed into one relation. This results in duplication and an unnecessary dependencies

Normal Forms

- There is a theory that allows one to see whether or not a particular schema risks having redundancy anomalies
- This theory looks at the design and also at constraints on the application, expressed in a mathematical form as *functional dependencies*
- If the design and its dependencies have certain properties, we say that the design is in a particular *normal form*
  - ***There are several that are used; we focus on BCNF***
- Our goal is to use schema which are in BCNF

Decomposition

- Decomposes changes a schema by replacing one relation by other relations
  - Between them, the decomposed relations have all the attributes of the original
  - The data in each decomposed table is a projection of the data in the original table

Decomposition

Olympic Dam	SA
Blair Athol	QLD
Hunter Valley	NSW

Mine information

Olympic Dam	Uranium	100
Blair Athol	Coal	920
Hunter Valley	Coal	1430
Blair Athol	Uranium	76

Mine-commodity connection

Uranium	U
Coal	Cbl

Commodity information

Olympic Dam	BHP Billiton
Blair Athol	Rio Tinto
Hunter Valley	Rio Tinto
Hunter Valley	Coal & Allied

Ownership connection

BHP Billiton	www.bhpbilliton.com
Rio Tinto	www.riotinto.com
Coal & Allied	www.coalandallied.com.au

Company information

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 should be *lossless-join*
  - It should be *dependency-preserving*

Overall design process

- Consider a proposed schema
- Find out application domain properties expressed as functional dependencies
- See whether the schema has a certain property (it is in BCNF)
- If not, pick a relation in the schema which can be decomposed in a particular way (the decomposition is lossless-join and dependency-preserving)
- Replace the original relation by its decomposed tables
- Repeat the evaluation

Today's Agenda

- Motivation
- Making it precise: Functional Dependency
- Making it precise: Normal Form
- Making it precise: Schema Decomposition

Functional Dependency (FD)

- Domain constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
- **Functional Dependency**:
  - Intuitively: "If two tuples of a relation R agree on values in X, then they must also agree on the Y values."
  - Note that X and Y are each a *set of columns*
    - Include as a special case where X or Y is a single column
  - Formally: Given a relation R with schema ***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 ***R*** if, for every allowable instance *R* of ***R***:

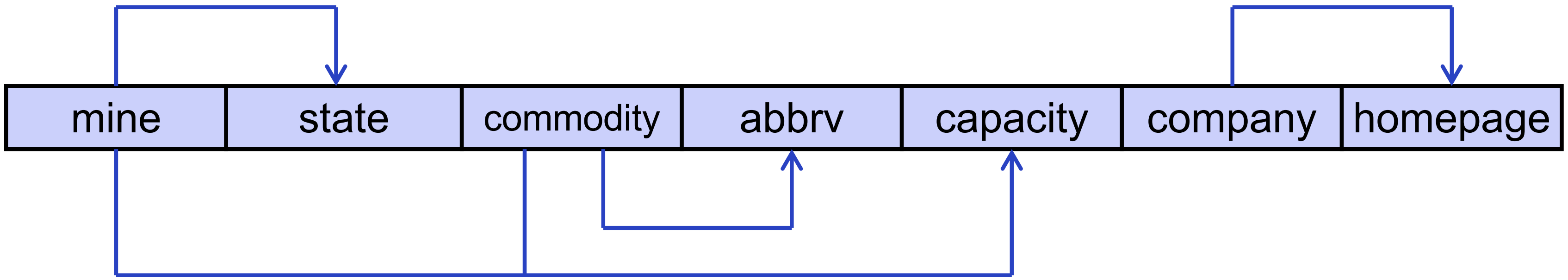
    $$\forall\ r, s \in R:\quad r.X = s.X \;\Rightarrow\; r.Y = s.Y$$
- We write $X \rightarrow Y$
  - "X (functionally) determines Y"
  - "Y is functionally dependent on X"

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 its 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

FDs and candidate keys

- A set of columns X makes up a candidate key for R provided
  - For every attribute A, $X \rightarrow A$
  - No subset of X has this property
- So, given a set F of FDs, we can find the candidate keys by looking at each set of columns and seeing whether they functionally determine all the other columns
- But, $X \rightarrow A$ might be true even though it is not be explicitly listed in the set F

Keys Example

$ABC \rightarrow D$

ABC determines all attributes.

ABC is a Superkey

$AB \rightarrow D$, $AB \rightarrow C$

AB is minimal set of attributes

that finds all other attributes.

AB is candidate key and can be assigned as primary key (PK).

Table			
<u>A</u>	<u>B</u>	C	D
1	3	X	V1
2	4	Y	V2
2	5	Z	V1
3	3	Y	V2
…	…	…	..

Given the above dependencies,

If we also have a FD $B \rightarrow C$ this is called **partial dependency** since B is part of candidate key

If we also have a FD C $\rightarrow D$ this is called **transitive dependency** since C is not a key

If we also have a FD A$B \rightarrow A$ this is called **trivial dependency** since A is part of the known key

Deducing more FDs

- Given some FDs, we can usually infer additional FDs:
  - Example: *cid* $\rightarrow$ *points*, *points* $\rightarrow$ *lot* implies *cid* $\rightarrow$ *lot*
- 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 $X \rightarrow Z$

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*⁺**
- A couple of additional rules (that follow from Armstrong's Axioms.):

4. Union rule: If $X \to Y$ and $X \to Z$, then $X \to YZ$

5. Decomposition rule: If $X \to YZ$, then $X \to Y$ and $X \to Z$

6. Pseudotransitivity rule: If $X \to Y$ and $SY \to Z$, then $XS \to Z$

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

date, cid, pid $\to$ pid	(reflexivity rule)
descr, price, weight $\to$ descr	(reflexivity rule)
pid $\to$ descr	(decomposition rule)
date, cid, pid $\to$ descr	(transitivity rule)

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

FD Attribute Closure

- Computing the closure of a set of FDs can be expensive . (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F.
  - An efficient check is:
  - Compute FD *attribute closure* of X (denoted X^+) wrt F:
    - To compute the set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- E.G. Does F = {A $\rightarrow$ B, B $\rightarrow$ C, C D $\rightarrow$ E } imply A $\rightarrow$ E?
  - i.e, is A $\rightarrow$ E in the closure F^+ ? Equivalently, is E in A^+?

Computing the Closure of Attributes

- Starting with a given set of attributes, one repeatedly expands the set by adding the right sides of FD's as soon as their left side is added:
  1. Initialise X 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 X, but C is not.
     Add C to the set X.
  3. Repeat step 2 until no more attributes can be added to X
  4. The set X is the correct value of A^+

This calculation is often called "The Chase"

From FDs to Keys

- The set of Functional Dependencies can be used to find candidate keys:
  - Look at each set of attributes *A*
  - Calculate A^+
  - If A^+ contains all columns, then *A* 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

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

Today's Agenda

- Motivation
- Making it precise: Functional Dependency
- Making it precise: Normal Form
  - Especially BCNF
- Making it precise: Schema Decomposition

Schema Normalization

- FDs can be used to identify schemas with problems and to suggest refinements.
- Main Idea: Have a schema where every FD has form of key constraint.
  - Each non-key field is anyway functionally dependent on every candidate key
  - Normal form: no other FDs

- **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: Starting with ER diagram conceptual model, we often get very close to a fully normalised schema.
  - But to be sure we have to check…

Normal Forms

- Before doing schema refinement, the first question to ask is whether any refinement is needed!

- If a relation is in a certain *normal form* (such as **BCNF**), it is known that certain kinds of problems are avoided/minimised. This can be used to help us decide whether decomposing the relation will help.

- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold:
      There is no redundancy here.
    - Given $A \rightarrow B$:
      Several tuples could have the same A value, and if so, they'll all have the same B value!

BCNF

- Be careful with the use of mathematics and logic!
  - Definitions are precise
- A relation with schema ***R*** and FDs *F* is in BCNF (Boyce-Codd Normal From) if, for all $X \rightarrow Y$ in F^+, either
  - $Y \subseteq X$ (i.e., the FD is trivial) or
  - *X* is a superkey of **R**
- Definition says: Check each fd that applies to the relation (including those deduced from the ones you were given)
  - If the rhs is contained in the lhs; ok (but uninteresting)
  - If the lhs is a superkey for the whole relation; ok
- If every fd is ok, the relation is BCNF
  - In fact, you only need to check the ones you are given!
- A schema is in BCNF if every relation in it is BCNF

BCNF Examples

- **Example**:
  Person1(*SSN, Name, Address*)
  - Suppose that the only FD is *SSN* $\rightarrow$ *Name, Address*
  - Since *SSN* is a key, Person1 is in BCNF
- **Example:**
  Order(*date*,*cid*,*pid*,*amount*) with *F*={*date cid pid* $\rightarrow$ *amount*} is in BCNF,

  and Product (*pid*,*descr*,*price*,*weight*) with

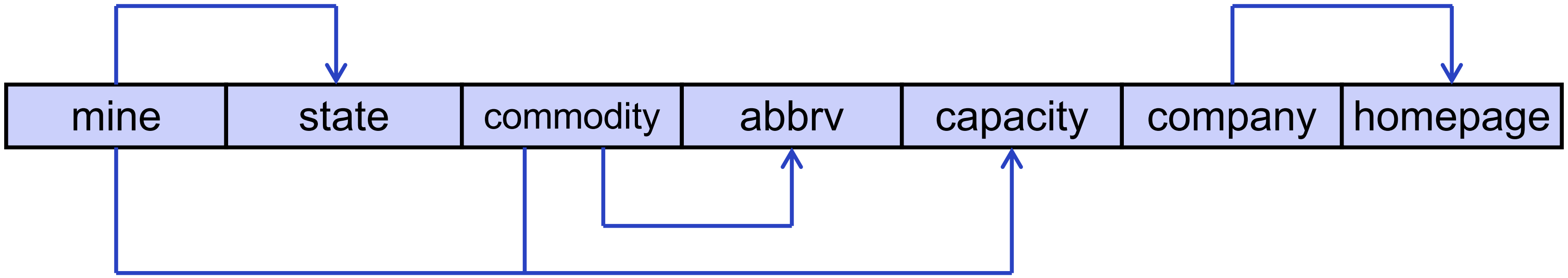
  F = {*pid* $\rightarrow$ *descr price weight*} is in BCNF.

(non) BCNF Examples

- Person (*SSN, Name, Address, Hobby*)
  - If the FD is *SSN* $\rightarrow$ *Name, Address* the relation does <u>not</u> satisfy requirements of BCNF
    - *SSN* is not a superkey; it does not determine Hobby
      - The FD is neither trivial, nor is the lhs a superkey. OOPS!
    - the key is (*SSN, Hobby*)
- HasAccount (*AcctNum, ClientId, OfficeId*)
  - If the FDs are *AcctNum* $\rightarrow$ *OfficeId* and *ClientId, OfficeId* $\rightarrow$ *Acctnum*, then the relation is <u>not</u> in BCNF
    - (ClientId, OfficeId) is a superkey; one FD is OK!
    - AcctNum is not a superkey; it does not determine ClientId; OOPS!
    - A relation is in BCNF when every FD is trivial or has superkey on lhs
    - Candidate keys are (*ClientId, OfficeId*) and (*AcctNum, ClientId*)

Normalisation Example

Mining relation



- What the Key? How can you demonstrate this?
- Is this schema in BCNF?

Other normal forms

- **Design theory has provided other definitions, each with its own uses**
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)
  - Fourth normal form (4NF) etc
- But we concentrate on BCNF!

First Normal Form

- A relational *R* is in **first normal form (1NF)** if the domains of all attributes of *R* are *atomic.*
  - This is built in to the relational model as we defined it!

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data

<u>SID</u>	CourseID	Date	Name
1235 1412 1311	312, 333, 454 234, 454 213	10/10 10/11 10/01	James Albert Nemo Davis Tasos West

Second Normal Form

- Second Normal Form (2NF): if no nonkey attribute is functionally dependent on just a part of the key.
  - Composite key

SID	CourseID	Grade	FName
1235	312	A	James
1412	232	F	Nemo
1235	515	A	James
1412	460	C	Nemo

Ssn	Name	lot	rating	Hourly_wage	Hours_worked
3666	‘Attishoo’	48	8	10	40
5368	‘Smiley’	22	8	10	30
3650	‘Setting’	35	8	10	30
3751	‘Guldu’	35	5	7	32
4134	‘Madayan’	35	5	7	40

Third normal form (3NF)

- Relation R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (a *trivial* FD), or
  - X is a superkey, or
  - A is part of some ***candidate key*** for R.

Example

- Example
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Two candidate keys: JK and JL
  - R is in 3NF

    $JK \rightarrow L$ JK is a superkey
    $L \rightarrow K$ K is contained in a candidate key
- There is some redundancy in this schema
- Example:

  Advisor-schema = (Client, Problem, Consultant)

  Consultant $\rightarrow$ Problem

  Client, Problem $\rightarrow$ Consultant

Client	Consultant	Problem
James	Gomez	Marketing
James	Jay	Production

BCNF and 3NF

- If R is in BCNF, obviously R is in 3NF.
  - The 3NF definition includes an extra possibility for each FD
- If R is in 3NF, it may not in BCNF.
- Lectures (*course*, *time*, *room*) with
  $F = \{course \rightarrow room,\ time\ room \rightarrow course\}$
  is in 3NF
  - Candidate keys: {*course*, *time*} and {*time*, *room*}

but not in BCNF.

  - FD *course* $\rightarrow$ *room* violates the BCNF requirements.
  - Example instance:

lectures		
course	time	room
Comp5138	Mo, 6-8 pm	CAR273
Comp5138	Mo, 8-9 pm	CAR273
Info3005	Tue, 9-11	Physics 1
Info3005	Thu, 12-14	Physics 1

Today's Agenda

- Motivation
- Making it precise: Functional Dependency
- Making it precise: Normal Form
- Making it precise: Schema Decomposition
  - Purpose: normalization

Decomposition of Relational Schema

- Suppose that relation R contains attributes *A1 ... An.*
  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 or more of the new relations.
- E.g., Can decompose OrderPlus(*date, cid, pid, descr, price, weight, amount)* into Order(*date*,*cid*,*pid*,*amount*) and Product (*pid*,*descr*,*price*,*weight*).
- Decomposing R means we will store instances which are projected from instances of R.
  - Each instance in the new schema has a subset of the columns in the old instance
  - Notation: $\prod_X (R)$
- How do we know we should decompose as above, and not as O1(*date, pid, cid, amount*) and O2(*date, desc, price, weight*)???

Decomposition and the Data

- Suppose we replace R by a decomposition into R1 and R2
- The information we have about the world in the new schema is what we can deduce from the instance of R1 and the instance of R2
- These are all the facts which are produced by joining the two tables
  - That is, take a tuple from R1 and a tuple from R2, which agree in the values for the common attributes
  - Recall notation: $R1 \bowtie R2$
- How does this compare to the information in the original relation, before decomposition?
  - Answer: it always has the information in the original, but it may have introduced extra (spurious) information
    - Whether this has happened depends on the choice of decomposition, and the constraints on the domain

Example for Decomposition with spurious tuples in join

OrderPlus						
date	cid	pid	descr	price	weight	amount
03.05.	1001	1	Paper	20.0	2.0	100
03.05	1001	6	Disks	5.0	0.4	50

O1			
date	cid	pid	amount
03.05.	1001	1	100
03.05	1001	6	50

O2			
date	descr	price	weight
03.05.	Paper	20.0	2.0
03.05	Disks	5.0	0.4

O1 $\bowtie$ *O2*						
date	cid	pid	descr	price	weight	amount
03.05.	1001	1	Paper	20.0	2.0	100
03.05	1001	6	Disks	5.0	0.4	50
03.05.	1001	1	Disks	5.0	0.4	100
03.05	1001	6	Paper	20.0	2.0	50

Definition: Lossless-Join Decomposition

- Decomposition of *R* into *X* and *Y* is *lossless-join* w.r.t. a set of FDs *F* if, *for every instance R* that satisfies *F*:
  - $\Pi_X(R) \bowtie \Pi_Y(R) = R$
- It is always true that $R \subseteq \Pi_X(R) \bowtie \Pi_Y(R)$
  - In general, the other direction does not hold! If it does for every instance, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless, so that we are not losing knowledge by having a database which implies spurious (incorrect) statements!*

What is lost?

- In the decomposed tables, the join has *extra* tuples
  - So why do we call the decomposition “not lossless”?
- Knowledge has been lost
  - We no longer know that certain combinations of facts are untrue!
  - Eg we lose knowledge that there is NOT an OrderPlus tuple with (03.05, 1001, 1, Disks, 5.0, 0.4, 100)

O1 ⋈ *O2*						
date	cid	pid	descr	price	weight	amount
03.05.	1001	1	Paper	20.0	2.0	100
03.05	1001	6	Disks	5.0	0.4	50
03.05.	1001	1	Disks	5.0	0.4	100
03.05	1001	6	Paper	20.0	2.0	50

Checking Lossless-Join Decomposition

- The definition requires considering every possible instance
  - Instead, just use the following theorem
- **Theorem:** The decomposition of *R* into X and Y is lossless-join wrt *F* if and only if the closure of F contains either
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$, or both
- Example OrderPlus on earlier slide:
  - $F = \{date\ cid\ pid \rightarrow amount,\ \ pid \rightarrow descr\ price\ weight\}$
  - O1(*date, pid, cid, amount*) and O2(*date, desc, price, weight*)
  - $X \cap Y$ is {date}
  - F^+ does not contain $date \rightarrow descr\ price\ weight$
  - F^+ does not contain $date \rightarrow cid\ pid\ amount$

Dependency-Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - therefore JSD → C so,
  - BCNF decomposition: CJSDQV and SDP
  - Problem: Checking JP → C requires a join!

- Intuitive:
  - If *R* is decomposed into *S* and *T*, and we enforce the FDs that hold on *S* and on *T*, then all FDs that were given to hold on *R* must also hold.

- *Projection of set of FDs F*:
  If a relational schema ***R*** is decomposed into ***S***, *T* …, the **projection of *F* onto *S*** (denoted F_S) is the set of
  FDs U → V in F^+ (*closure of F*) such that *U, V* are in ***S***.

Dependency Preserving Decomposition Definition

- Given a relation R with schema $\boldsymbol{R}$ and set of FDs F.
  A decomposition of $\boldsymbol{R}$ into $\boldsymbol{S}$ and $\boldsymbol{T}$ is a dependency preserving decomposition if

$$(F_{\boldsymbol{S}} \cup F_{\boldsymbol{T}})^{+} = F^{+}$$

  - i.e., if we consider only dependencies in the closure F^{+} that can be checked in $\boldsymbol{S}$ without considering $\boldsymbol{T}$, and in $\boldsymbol{T}$ without considering $\boldsymbol{S}$, these imply all dependencies in F^{+}.

- It is important to consider F^{+}, not F, in this definition:
  - ABC, F={A → B, B → C, A → C}, decomposed into AB and BC.
  - Is this dependency preserving? Is A → C preserved?????
  - As F_{AB}={A → B}, F_{BC}={B → C}, then ($F_{AB} \cup F_{BC}$) ={A → B, B → C}
  - So $F \neq (F_{AB} \cup F_{BC})$
  - However A → C is in $(F_{AB} \cup F_{BC})^{+}$
  - This decomposition is dependency preserving: $F = (F_{AB} \cup F_{BC})^{+}$
- Dependency preserving does not imply lossless join, and vice-versa!
  - Lossless-join is more important if we can't have both

Examples for Dependency-Preserving

- The decomposition of OrderPlus(*date, cid, pid, descr, price, weight, amount*) with $F = \{date\ cid\ pid \rightarrow amount,\ pid \rightarrow descr\ price\ weight\}$ into
  - O3(*date*,*cid*,*pid*,*amount*) with $F1 = \{date\ cid\ pid \rightarrow amount\}$ and
  - O4 (*pid*,*descr*,*price*,*weight*) with $F2 = \{pid \rightarrow descr\ price\ weight\}$

  is dependency-preserving.

- The decomposition of Lectures (*course*, *time*, *room*) with $F = \{course \rightarrow room,\ time\ room \rightarrow course\}$ into
  - Rooms(*course, room*) with $F1=\{course \rightarrow room\}$ and
  - Times(*course, time*) with F2 = {}

  is not dependency-preserving.

Central Facts of Normalization

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

- Every relation *R* with set of FDs *F can be* decomposed into **BCNF** relations which is a *lossless-join*.
  - Unfortunately, there may be no dependency-preserving decomposition into a collection of BCNF relational schema

Algorithm for Normalization

In general, several dependencies may cause violation of BCNF…

- Consider relation *R* with FDs *F*.
  If *X* → *Y* violates BCNF, decompose *R* into *R* - *Y* and *XY*.
  - Recall: X and Y are sets of columns
  - Recall R – Y means: all columns of R except those in Y
  - Recall XY means: the columns in X together with columns in Y
- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
- The order in which we "deal with" them could lead to very different sets of relations!

Example: BCNF Decomposition

- Given the following schema:
  course-schema (cid, title, cpoints, workload, lecturer, sid, grade)

- $F = \{ cid \rightarrow title\ cpoints\ lecturer,$
  $cpoints \rightarrow workload,$
  $cid\ sid \rightarrow grade \}$

- $\{cid\}^+ = \{ cid, title, cpoints, lecturer, workload \}$
  $\{cpoints\}^+ = \{ cpoints, workload \}$
  $\{cid, sid\}^+ = \{cid, sid, title, cpoints, lecturer, workload, grade\}$

- Candidate Key: { cid, sid }

Example: BCNF Decomposition (cont'd)

- *Course-schema* is not BCNF, because, e.g., first and second FD violate BCNF

- Decompose *Course-schema* along *cpoints* → *workload* first!
  - *Workloads (cpoint, workload)*
  - *Courses-schema2* (*cid, title, cpoints, lecturer, sid, grade)*

- *Course-schema2* is still not BCNF; further decompose along *cid* → *title cpoints lecturer*
  - *Workloads (cpoints, workload) with F2 = {cpoints → workload}*
  - *Courses* (*cid, title, cpoints, lecturer)*
    with F1 ={cid → title cpoints lecturer }
  - *Enrolled* (*cid, sid, grade) with F3 = {cid sid → grade}*

- *Is this decomposition lossless-join & dependency-preserving?*

Example: BCNF Decomposition (cont'd)

- If you decompose along *cid* → *title cpoints lecturer* **first**, there is a different schema!
  - *Courses* (*cid, title, cpoints, lecturer),*

    with F1 ={cid → title cpoints lecturer }
  - *Workload_enroll(cid, workload, sid, grade),*

    with F3 = {cid sid → grade}
  - *Can we decompose any of above schema further?*
  - *How about FD cpoints → workload?*

Another Example

Given: R = (R; F) where $R = ABCDEGHK$ and

$F = \{ABH \rightarrow C,\ A \rightarrow DE,\ BGH \rightarrow K,\ K \rightarrow ADH,\ BH \rightarrow GE\}$

step 1: Find a FD that violates BCNF

Not $ABH \rightarrow C$ since $(ABH)^+$ includes all attributes

(BH is a key)

$A \rightarrow DE$ violates BCNF since A is not a superkey ($A^+ = ADE$)

step 2: Split R into:

$R_1 = (ADE,\ F_1 = \{A \rightarrow DE\})$

$R_2 = (ABCGHK;\ F_1 = \{ABH \rightarrow C,\ BGH \rightarrow K,\ K \rightarrow AH,\ BH \rightarrow G\})$

Note 1: R_1 is in BCNF

Note 2: Decomposition is *lossless* since A is a key of R_1.

Note 3: FDs $K \rightarrow D$ and $BH \rightarrow E$ are not in F_1 or F_2. But both can be derived from $F_1 \cup F_2$

($E.g.$, $K \rightarrow A$ and $A \rightarrow D$ implies $K \rightarrow D$)

Hence, decomposition *is dependency preserving*.

Another Example (con't)

Given: $R_2 = (ABCGHK;\ \{ABH \to C,\ BGH \to K,\ K \to AH,\ BH \to G\})$

step 1: Find a FD that violates BCNF.

Not $ABH \to C$ or $BGH \to K$, since BH is a key of R_2

$K \to AH$ violates BCNF since K is not a superkey ($K^+ = AH$)

step 2: Split R_2 into:

$R_{21} = (KAH,\ F_{21} = \{K \to AH\})$

$R_{22} = (BCGK;\ F_{22} = \{\})$

Note 1: Both R_{21} and R_{22} are in BCNF.

Note 2: The decomposition is *lossless* (since K is a key of R_{21})

Note 3: FDs $ABH \to C$, $BGH \to K$, $BH \to G$ are not in F_{21} or F_{22}, and they can't be derived from $F_1 \cup F_{21} \cup F_{22}$.

Hence the decomposition is *not* dependency-preserving

Summary

- Motivation: avoid redundancy
- Functional Dependency
  - Capture constraints
  - Work with definitions
- Normal Forms
  - BCNF
  - Check whether schema is in BCNF, based on FDs
- Decomposition
  - Lossless-join and dependency-preserving?
  - Normalize by doing lossless-join decomposition till all relations are BCNF