Functional Dependencies and Normalization

5DV119 — Introduction to Database Management
Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

The Idea of Normalization

- One might think that the design of a relational schema is as simple as coming up with the relations and declaring the appropriate dependencies.
- However, things are more complicated.
- If the design is not done properly, essential dependencies may not be representable directly using the tools which SQL provides.
- As a consequence, such nonrepresentable dependencies are often simply ignored in defective designs, resulting in:
  - fundamental errors in the results of queries, particularly complex queries which combine several relations;
  - redundancy in both information and storage.
- It is therefore essential to have a uniform way of managing basic dependencies and to ensure that they are represented correctly in the schema.
- In these slides, only *functional dependencies*, which are a generalization of key dependencies, will be considered.
- The presentation is theoretical but a natural one.

Functional Dependencies

- Key constraints are the most important kind of dependencies in the relational model.
- Sometimes, key constraints may apply only on a projection of a relation.
- This can occur...
  - ... during the design process when the relations are "too big" and need to be decomposed.
  - ... in some situations in which such embedded dependencies are unavoidable.
- A (super)key dependency on a projection of a relation is called a *functional dependency*.

Examples of Functional Dependencies

Firm

SSN	Name	Dept	Bldg
000112222	Alice	3	8
000113333	Bruce	3	8
000114444	Carol	3	8
000115555	David	5	7
000116666	Alice	4	7

$$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$$
$$\text{Dept} \rightarrow \text{Bldg}$$

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$ is the functional dependency which states that SSN is a (super)key on the projection $\pi_{\{\text{SSN},\text{Name},\text{Dept}\}}(\text{Firm})$.

- In words, SSN *functionally determines* Name and Dept.

$\text{Dept} \rightarrow \text{Bldg}$ is the functional dependency which states that Dept is a (super)key on the projection $\pi_{\{\text{Dept},\text{Bldg}\}}(\text{Firm})$.

- A simple transitivity argument (to be formalized later) shows that SSN is a key for the whole relation Firm: $\text{SSN} \rightarrow \{\text{Name}, \text{Dept}, \text{Bldg}\}$.
- However, $\text{Dept} \rightarrow \text{Bldg}$ is not a (super)key dependency on Firm in any reasonable sense.
  - It is not representable in native SQL.

Examples of Functional Dependencies — 2

Employee

<u>ID</u>	Name	StreetAddr	City	State	PostCode

$$\mathrm{ID} \rightarrow \{\mathrm{Name}, \mathrm{StreetAddr}, \mathrm{City}, \mathrm{State}, \mathrm{PostCode}\}$$
$$\{\mathrm{StreetAddr}, \mathrm{City}, \mathrm{State}\} \rightarrow \mathrm{PostCode}$$
$$\mathrm{PostCode} \rightarrow \{\mathrm{City}, \mathrm{State}\}$$

- The above schema illustrates a common situation regarding addresses with postal codes.
- Here State is *län*, *Bundesland*, *région*, and the like.
- There is a complex overlap of functional dependencies.
- The street address, city, and state determine the postal code.
- The postal code determines the city and state, but not necessarily the street address.
- Note that set brackets are omitted for singletons (sets with one element).

Examples of Functional Dependencies — 3

Part	Site	Distributor
bolt	Umeå	Ola
screw	Umeå	Ola
bolt	Tromsø	Tone
widget	Tromsø	Kari
nut	Aalborg	Anne

$$\{\text{Part}, \text{Site}\} \rightarrow \text{Distributor}$$
$$\text{Distributor} \rightarrow \text{Site}$$

- The previous example is unlikely to pose a problem for a corporation, since the relationship between addresses and postal codes is fixed by the postal authority, and so need not be verified.

- A simple example of a similar form of overlapping dependencies in corporate setting is illustrated above.

- The part and site together determine the distributor for that part.

- Distributors are local; each distributor provides parts to only one site.

Formalization of Functional Dependency

- Recall the notation from Slides 5–9 of "The Relational Model of Data".
- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$.
- Let $\mathbf{X}, \mathbf{Y}$ be subsequences of $(A_1, A_2, \ldots, A_k)$.
- The *functional dependency (FD)* $\mathbf{X} \rightarrow \mathbf{Y}$ holds on an instance M_R of R if for any two tuples $t_1, t_2 \in M_R$,
$$t_1[\mathbf{X}] = t_2[\mathbf{X}] \Rightarrow t_1[\mathbf{Y}] = t_2[\mathbf{Y}]$$
- The *FD constraint* $\mathbf{X} \rightarrow \mathbf{Y}$ on the scheme R mandates that all allowed instances M_R satisfy this FD.
- Observe that if $\mathbf{X} \cup \mathbf{Y} = \mathbf{U} = (A_1., A_2, \ldots, A_k)$, then this definition reduces to requiring that $\mathbf{X}$ be a superkey.
- In other words, $\mathbf{X} \rightarrow \mathbf{Y}$ holds iff $\mathbf{X}$ is a superkey on the projection $\pi_{\mathbf{X} \cup \mathbf{Y}}(R)$ of R onto the attributes in $\mathbf{X} \cup \mathbf{Y}$.

Special Kinds of Functional Dependencies

- The FD $\mathbf{X} \to \mathbf{Y}$ is *degenerate* if $\mathbf{X}$ is empty.
  - Degenerate FDs will not be considered in these slides.
  - *FD* will always mean nondegenerate FD.
- The FD $\mathbf{X} \to \mathbf{Y}$ is *trivial* if $\mathbf{Y} \subseteq \mathbf{X}$.
  - Trivial FDs are uninteresting in that they always hold, but they may arise in certain constructions.
- The FD $\mathbf{X} \to \mathbf{Y}$ is *fully nontrivial* if $\mathbf{X} \cap \mathbf{Y} = \emptyset$.
  - The FD may always be replaced by a fully nontrivial one by removing from $\mathbf{Y}$ all attributes in $\mathbf{X}$.
- A set $\mathcal{F}$ of FDs is *fully nontrivial* if each of its members has that property.

Convention: Unless stated to the contrary, when considering a set $\mathcal{F}$ of FDs on a schema, it will always be assumed that it consists of fully nontrivial elements.

  - This applies to a single FD of the form $\mathbf{X} \to \mathbf{Y}$ as well

Entailment of Functional Dependencies

- When it is known that certain FDs hold on a relation, it can be deduced that others hold as well.

Example:

Firm

SSN	Name	Dept	Bldg
000112222	Alice	3	8
000113333	Bruce	3	8
000114444	Carol	3	8
000115555	David	5	7
000116666	Alice	4	7

$\text{SSN} \to \{\text{Name}, \text{Dept}\}$
$\text{Dept} \to \text{Bldg}$

- FDs are closed under transitivity, and so it is easy to see that $\text{SSN} \to \text{Bldg}$ holds.
- Right-hand sides may always be broken up, so $\text{SSN} \to \text{Name}$ and $\text{SSN} \to \text{Dept}$ also hold.
- Right-hand sides may be combined, so $\text{SSN} \to \{\text{Name}, \text{Dept}, \text{Bldg}\}$ also holds.

Entailment of Functional Dependencies — 2

- Here is an example of entailment at a more abstract level.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E\}$.

- FDs are closed under transitivity, and so it is easy to see that each of $A \rightarrow BCDE$, $B \rightarrow CDE$, and $C \rightarrow DE$ hold.

- Right-hand sides can always be broken up, so $B \rightarrow BCDE$ implies each of $B \rightarrow B$, $B \rightarrow C$, $B \rightarrow D$, and $B \rightarrow E$.

- Conversely, right-hand sides may be combined for identical left-hand sides. For example, the set $\{B \rightarrow B, B \rightarrow C, B \rightarrow D,\ B \rightarrow E\}$ implies that $B \rightarrow BCDE$ holds.

Entailment of Functional Dependencies — 3

Caution: Left hand sides of FDs may not be broken up in general.

Examples:

A	B	C

$\mathcal{F}_1 = \{AB \rightarrow C\}$
$\mathcal{F}_2 = \{A \rightarrow C, B \rightarrow C\}$

- $\mathcal{F}_2$ is strictly stronger than $\mathcal{F}_1$.

Example instance to illustrate :

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_3
a_2	b_2	c_4

- This instance satisfies $\mathcal{F}_1$ but not $\mathcal{F}_2$.

Concrete example: Think of (the key dependency) $\{\text{ESSN}, \text{PNo}\} \rightarrow \text{Hours}$ of the Works_On relation of the Company schema.

- Neither ESSN nor PNo is a key by itself.

Formalization and Notation for Entailment

- Entailment of FDs occurs so frequently that it is useful to have a special notation for it.

Entailment of FDs Let $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ be a relation scheme with FDs $\mathcal{F}$, and let $\mathbf{X} \to \mathbf{Y}$ be an FD. $\mathbf{X} \to \mathbf{Y}$ is *entailed by* $\mathcal{F}$, written $\mathcal{F} \models \mathbf{X} \to \mathbf{Y}$, if $\mathbf{X} \to \mathbf{Y}$ holds on every relation on which $\mathcal{F}$ holds.

- If $\mathcal{G}$ is a set of FDs, then $\mathcal{F} \models \mathcal{G}$ means that $\mathcal{F} \models \varphi$ for every $\varphi \in \mathcal{G}$.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \to B, B \to C, C \to D, D \to E\}$.

- $\mathcal{F} \models A \to BCDE$, $\mathcal{F} \models B \to CDE$, $\mathcal{F} \models C \to DE$, and $\mathcal{F} \models \{A \to BCDE,\ B \to CDE,\ C \to DE\}$.

- Also may write $\mathcal{F} \models A \to BCDE,\ B \to CDE,\ C \to DE$.

Further Examples of Entailment of FDs

- There may be “loops” in sets of FDs.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \to B, B \to C, C \to D, D \to B\}$.

- Here B, C, and D are all equivalent.
- Write $\mathbf{X} \leftrightarrow \mathbf{Y}$ to mean that both $\mathbf{X} \to \mathbf{Y}$ and $\mathbf{Y} \to \mathbf{X}$ hold.
- Then $\mathcal{F} \models B \leftrightarrow C,\ B \leftrightarrow D,\ C \leftrightarrow D$.
- Composition may also be on subsets of attributes.

Example:

A	B	C	D	E

$\mathcal{F} = \{AB \to C, CD \to E\}$.

- $\mathcal{F} \models ABD \to CE$; *i.e.*, ABD is a key.

Observation: It seems that inference for FDs is governed largely by a transitivity operation.

Question: How can $\models$ be computed systematically?

- ... to be used in algorithms, for example.

Inference Systems for FDs

- An *inference system* $\mathfrak{I}$ is a mathematical proof system for $\models$.
- It is a way to compute $\models$ via rules.
- Write $\mathcal{F} \vdash_{\mathfrak{I}} \varphi$ if the FD φ may be proven from $\mathcal{F}$ using $\mathfrak{I}$.
- There are three key properties for any proof system $\mathfrak{I}$.

Soundness: The inference system $\mathfrak{I}$ is *sound* if everything which can be proven is true: $\mathcal{F} \vdash_{\mathfrak{I}} \varphi$ implies $\mathcal{F} \models \varphi$.

Completeness: The inference system $\mathfrak{I}$ is *complete* if everything which is true can be proven: $\mathcal{F} \models \varphi$ implies $\mathcal{F} \vdash_{\mathfrak{I}} \varphi$.

Decidability: The inference system $\vdash_{\mathfrak{I}}$ is *decidable* if there is an algorithm (which always halts) which can compute $\vdash_{\mathfrak{I}}$.

Armstrong's Axioms

- The classical inference system $\mathfrak{A}$ for FDs is defined by *Armstrong's Axioms*, which are as follows.

- In all cases, $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are sequences of attributes over some universe $\mathbf{U}$.

A1 (triviality): If $\mathbf{Y} \subseteq \mathbf{X}$, then $\vdash_{\mathfrak{A}} \mathbf{X} \to \mathbf{Y}$.

- This means that $\mathbf{X} \to \mathbf{Y}$ follows from the empty set of FDs.
- This rule is sometimes (incorrectly) called *reflexivity*.

A2 (augmentation): $\mathbf{X} \to \mathbf{Y} \vdash_{\mathfrak{A}} \mathbf{XZ} \to \mathbf{YZ}$

- Here $\mathbf{XZ}$ means merge the two sequences of attributes (union of elements with the proper order).

A3 (transitivity): $\{\mathbf{X} \to \mathbf{Y}, \mathbf{Y} \to \mathbf{Z}\} \vdash_{\mathfrak{A}} \mathbf{X} \to \mathbf{Z}$.

Properties of Armstrong's Axioms

- Armstrong's Axioms cannot be "proven". It does not make sense to "prove" inference rules.
- However, there is the following key result.

Theorem: Armstrong's Axioms are sound and complete for inference on FDs. $\square$

- This result will not be established here, although it is relatively straightforward to prove.
- In words, it says that transitivity is the only "nontrivial" way in which inference occurs with FDs.

Theorem: Armstrong's Axioms provide a decidable system for inference on FDs.

Proof: This is immediate because there are only a finite number of FDs over any given finite universe $\mathbf{U}$ of attributes. $\square$

Proofs Using Armstrong's Axioms

- Proofs are written out as a list of assertions, with each assertion either given or else following from the previous ones by Armstrong's Axioms.

Example:

A	B	C	D	E

$\mathcal{F} = \{AB \to C, CD \to E\}$.

- Prove that ABD is a key for this system using Armstrong's Axioms.
  (1) $AB \to C$ given.
  (2) $CD \to E$ given.
  (3) $ABD \to ABCD$ (Augmentation of (1) with $\mathbf{Z} = ABD$).
  (4) $ABCD \to ABCDE$ (Augmentation of (2) with $\mathbf{Z} = ABCD$).
  (5) $ABD \to ABCDE$ (Transitivity on (3)+(4)).

- Thus, $\mathcal{F} \vdash_{\mathfrak{A}} ABD \to ABCDE$.

- By the soundness of Armstrong's Axioms, it follows that
$$\mathcal{F} \models ABD \to ABCDE.$$

Other Axioms Systems

- Armstrong's Axioms $\mathfrak{A}$ form but one possibility for a sound and complete set of inference rules for FDs.
- Many others have been developed.
- One possibility is to augment $\mathfrak{A}$ with additional rules.
- These rules do not add power, since $\mathfrak{A}$ is already complete.
- However, they often allow proofs to be shorter by recapturing as "macros" steps with occur frequently.
- See the textbook for one such possibility.

Closure and Covers

- It is often the case that two distinct sets of FDs are equivalent in that they entail exactly the same FDs.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F}_1 = \{A \to BCD, CE \to D, CD \to E\}$,
$\mathcal{F}_2 = \{A \to BCE, CE \to D, CD \to E\}$.

- Exactly the same set of FDs may be derived from each of these two sets.
  - Need to show $\mathcal{F}_1 \models A \to E$ and $\mathcal{F}_2 \models A \to D$.

General Context: Let $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ be a relation scheme with with FDs $\mathcal{F}$.

Closure: $\mathcal{F}^+ = \{\mathbf{X} \to \mathbf{Y} \mid \mathcal{F} \models \mathbf{X} \to \mathbf{Y}\}$.

Cover: A *cover* for $\mathcal{F}$ is any set $\mathcal{C}$ of FDs with the property that $\mathcal{F}^+ = \mathcal{C}^+$.

Example: ${\mathcal{F}_1}^+ = {\mathcal{F}_2}^+$.

  - Each $\mathcal{F}_i$ is a cover for the other.

Embedding of Functional Dependencies

- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with $\mathcal{F}$ a set of FDs on R.
- Operations on subsequences of $\mathbf{U}$ may be written using set-theoretic notation.

  Examples: $\mathbf{W} \subseteq \mathbf{U}$, $\mathbf{X} \cup \mathbf{Y} \subseteq \mathbf{W}$.

  - This is a slight abuse of notation mathematically but it is widely used and this context the intended meaning is always clear.
  - The assumption is that the elements in the "sets" always occur in the order induced by the base set $\mathbf{U}$.
  - $\mathbf{X} \cup \mathbf{Y}$ is also written $\mathbf{XY}$.
- Say that the FD $\mathbf{X} \to \mathbf{Y} \in \mathcal{F}$ *embeds* into $\mathbf{W}$ if $\mathbf{X} \cup \mathbf{Y} \subseteq \mathbf{W}$.
- Let $\Pi_{\mathbf{W}}$ denote the view of R which is the projection onto $\mathbf{W}$.
- Say that the FD $\mathbf{X} \to \mathbf{Y} \in \mathcal{F}$ *embeds* into $\Pi_{\mathbf{W}}$ if it embeds into $\mathbf{W}$.

Normalization

Normalization: is the process of “fixing” relational schemata so that they avoid three closely related kinds of problems.

Storage redundancy: The same information is repeated many times.

Unnecessary information dependency: Information about some x cannot be represented without having at least corresponding instance of y.

Update anomalies: The way in which data is represented complicates the support of certain kinds of updates.

Illustration of Problems of an Unnormalized Schema

Firm

SSN	Name	Dept	Bldg
000112222	Alice	3	8
000113333	Bruce	3	8
000114444	Carol	3	8
000115555	David	5	7
000116666	Alice	4	7

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$
$\text{Dept} \rightarrow \text{Bldg}$

- The FD Dept $\rightarrow$ Bldg does not define a key and leads to problems.

Storage redundancy: The information about Department 3 is repeated three times.

Update anomaly: If the building of Department 3 is to be changed, three updates are necessary.

Unnecessary information dependency:

- Information about an employee who does not have a department requires null values.
- Information about a department cannot be represented unless at least one employee works in it.

Approaches to Normalization

Approaches to normalization: There are two principle approaches to normalization, and each will be considered in these slides.

Decomposition: Break larger relations into smaller ones.

Synthesis: Begin with a set of dependencies (usually FDs), and construct a corresponding relational schema.

The changes forced by normalization: Generally speaking, by forcing FDs to define (super)key dependencies, the problems identified above are minimized or disappear completely... but the devil is in the details.

Normal Forms

Normal forms: In early research on the relational model, a number of so called *normal forms* were developed.

- The principal ones which are based upon FDs were developed in the following order:

$$1NF \quad \rightarrow \quad 2NF \quad \rightarrow \quad 3NF \quad \rightarrow \quad BCNF$$

- There are some others which are based upon other types of dependencies: 4NF, 5NF, DKNF.
- In these slides, only those normal forms based upon FDs will be considered.
- For pedagogical reasons, they will considered in the reverse order of development:

$$BCNF \quad \rightarrow \quad 3NF \quad \rightarrow \quad 2NF \quad \rightarrow \quad 1NF$$

- The main focus will be upon BCNF and 3NF, as 2NF is largely of historical interest and 1NF is just a constraint on domains.

LDBN — an On-Line Resource for Learning Normalization

Learn DataBase Normalization: LDBN is an online resource for doing examples of normalization.

- It was developed as part of two thesis projects at Umeå University by Nikolay Georgiev.
- It implements several of the algorithms which will be considered in these slides.
- It allows one to test a given solution for various properties.
- It will also find a solution, if possible, which satisfies given properties.
- It has a very nice drag-and-drop interface.
- It can be found here:

`http://ldbnonline.com/Ldbn.html`

Boyce-Codd Normal Form — BCNF

- The main idea behind BCNF is that all FDs should be (super)key dependencies.
- Such a normalization is highly desirable since then (a cover for) the FDs can be represented within standard SQL.

Idea for a decomposition algorithm: If a relation scheme is constrained by an FD which does not define a superkey, decompose the scheme into subschemes which avoid that problem.

- Unfortunately, such a decomposition is not always possible without introducing other problems.
- Before developing the theory, an example will help illustrate why BCNF is desirable.

An Illustration of BCNF Normalization

- Recall the following unnormalized example.

Firm

SSN	Name	Dept	Bldg
000112222	Alice	3	8
000113333	Bruce	3	8
000114444	Carol	3	8
000115555	David	5	7
000116666	Alice	4	7

$$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$$
$$\text{Dept} \rightarrow \text{Bldg}$$

- The problems may be fixed by decomposing it into two relations.

Employee

SSN	Name	Dept
000112222	Alice	3
000113333	Bruce	3
000114444	Carol	3
000115555	David	5
000116666	Alice	4

Department

Dept	Bldg
3	8
4	7
5	7

- Note that each relation now has only a key dependency.
- Note also the added foreign-key dependency.

Lossless Joins

- In decomposing one relation into two, it must be ensured that no information is lost.
- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R.
- Let $P = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ be subsequences of $\mathbf{U}$.
- Let $\Pi_{\mathbf{W}_i}$ denote the view which is the projection of R onto the attributes in $\mathbf{W}_i$.
- The decomposition of R into the projections $\{\Pi_{\mathbf{W}_1}, \Pi_{\mathbf{W}_2}, \ldots, \Pi_{\mathbf{W}_k}\}$ satisfies the *lossless join property* (or is *lossless*) if the original schema may be recovered via the natural join operation.

Formal description: R *decomposes losslessly* into $\{\Pi_{\mathbf{W}_1}, \Pi_{\mathbf{W}_2}, \ldots, \Pi_{\mathbf{W}_k}\}$ (or just into P) *for* $\mathcal{F}$ if for any instance M_R of R which satisfies $\mathcal{F}$,

$$M_R = \pi_{\mathbf{W}_1}(M_R) \bowtie \pi_{\mathbf{W}_2}(M_R) \bowtie \ldots \bowtie \pi_{\mathbf{W}_k}(M_R)$$

The textbook uses the term *nonadditive* for *lossless*, but this is not standard terminology.

Guaranteeing Lossless Joins

- For a decomposition into two projections, there is a very simple condition which is both necessary and sufficient to ensure lossless recovery.

Theorem: Let $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ be a relation scheme with with FDs $\mathcal{F}$, and let $\mathbf{W}_1$ and $\mathbf{W}_2$ be subsequences of $\mathbf{U}$. Then R decomposes losslessly into $\{\Pi_{\mathbf{W}_1}, \Pi_{\mathbf{W}_2}\}$ for $\mathcal{F}$ iff (if and only if) the FD $\mathbf{W}_1 \cap \mathbf{W}_2 \to \mathbf{W}_i$ holds for at least one $i \in \{1, 2\}$. $\square$

- In words, there is a lossless decomposition iff the attributes common to $\mathbf{W}_1$ and $\mathbf{W}_2$ form a superkey for one of the resulting relations.

- Observe that the decomposition of the Firm example on the previous slides satisfies this condition.

Lossless Decomposition of the Example Schema Firm

Firm

SSN	Name	Dept	Bldg
000112222	Alice	3	8
000113333	Bruce	3	8
000114444	Carol	3	8
000115555	David	5	7
000116666	Alice	4	7

$\text{SSN} \to \{\text{Name}, \text{Dept}\}$
$\text{Dept} \to \text{Bldg}$

- $\mathbf{W}_1 = \{\text{SSN}, \text{Name}, \text{Dept}\}$ $\mathbf{W}_2 = \{\text{Dept}, \text{Bldg}\}$

Employee

SSN	Name	Dept
000112222	Alice	3
000113333	Bruce	3
000114444	Carol	3
000115555	David	5
000116666	Alice	4

Department

Dept	Bldg
3	8
4	7
5	7

- $\mathbf{W}_1 \cap \mathbf{W}_2 = \{\text{Dept}\}$ is a key for $\mathbf{W}_2 = \{\text{Dept}, \text{Bldg}\}$ since $\text{Dept} \to \text{Bldg}$.

A Simple Algorithm for Realizing BCNF

- There is a simple way to achieve BCNF.
- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with $\mathcal{F}$ a set of FDs on R.
- Select a cover $\mathcal{C}$ of $\mathcal{F}$. Without loss of generality, take $\mathcal{C}$ to consist of fully nontrivial dependencies.
- If R is not in BCNF for $\mathcal{C}$:
  - Pick an FD $\mathbf{X} \to \mathbf{Y} \in \mathcal{C}$ for which $\mathbf{X}$ is not a superkey.
  - Define $\mathbf{W}_1 = \mathbf{U} \setminus \mathbf{Y}$ and $\mathbf{W}_2 = \mathbf{X} \cup \mathbf{Y}$.
  - Decompose R into $\Pi_{\mathbf{W}_1}$ and $\Pi_{\mathbf{W}_2}$.
  - Make $\mathbf{X}$ in $\Pi_{\mathbf{W}_1}$ a foreign key which references $\mathbf{X}$ in $\Pi_{\mathbf{W}_2}$.
  - Repeat this process on the resulting schemata until all such schemata are constrained by key dependencies only.
- The resulting decompositions are all lossless.

Question: Does this algorithm have any complications or drawbacks?

BCNF and Alternative Decompositions

Firm

SSN	Name	Dept	Proj	Bldg

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$

$\text{Dept} \rightarrow \text{Proj} \quad \text{Proj} \rightarrow \text{Bldg}$

- There are two choices for an initial step.
- Using Proj $\rightarrow$ Bldg:

SSN	Name	Dept	Proj

Proj	Bldg

- Using Dept $\rightarrow$ Proj on the left relation, each FD embeds into a view:

SSN	Name	Dept

Dept	Proj

Proj	Bldg

- Using Dept $\rightarrow$ Proj first on the original schema, the FD Proj $\rightarrow$ Bldg does not embed into either view:

SSN	Name	Dept	Bldg

Dept	Proj

- This second decomposition is not very desirable because it introduces *inter-relational* FDs, which are not supported by DBMSs.

Dependency Preservation

- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R, and let $\mathbf{W}_1$ and $\mathbf{W}_2$ be subsequences of $\mathbf{U}$.

Preliminary definition: The decomposition of R into $\Pi_{\mathbf{W}_1}$ and $\Pi_{\mathbf{W}_2}$ *preserves* $\mathcal{F}$ if each $\varphi \in \mathcal{F}$ embeds into either $\Pi_{\mathbf{W}_1}$ or else $\Pi_{\mathbf{W}_2}$.

- Unfortunately, this definition is not very satisfactory.
- Consider (essentially) the same example as on the previous slide, but with the FD SSN $\rightarrow$ {Dept, Proj, Bldg} added.

Firm

SSN	Name	Dept	Proj	Bldg

SSN $\rightarrow$ {Name, Dept}

Dept $\rightarrow$ Proj   Proj $\rightarrow$ Bldg

SSN $\rightarrow$ {Dept, Proj, Bldg}

- It is easy to see that this new FD may be derived from the others by simple transitivity...
- But the following decomposition is not dependency preserving under this definition.

SSN	Name	Dept	Proj

Proj	Bldg

Dependency Preservation — 2

- It might seem sufficient to require that elements of the set $\mathcal{F}$ which can be derived from others simply be removed.
- This is not satisfactory either!

Example:

$\underline{\text{A}}$	B	C	D

$\mathcal{F} = \{A \to B, B \to C, C \to D, D \to B\}$.

- No element of $\mathcal{F}$ is implied by the others.
- Thus, the following is not dependency preserving under this definition.

$A \to B \quad B \to C$

$\underline{\text{A}}$	B	C

$D \to B$

$\underline{\text{C}}$	D

$C \to D$

- Note that there is a cycle $B \to C \to D \to B$.
- This cycle may be represented also as $\{B \to C, C \to B, C \to D, D \to C\}$
  - ...which is preserved by this decomposition.
- Thus, the above decomposition should qualify as dependency preserving.
- It is necessary to work with *covers* of $\mathcal{F}$.

Dependency Preservation and Covers

Implication of FDs Let $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ be a relation scheme with with FDs $\mathcal{F}$. Let $\mathcal{F}$ be a set of FDs on R.

Closure: $\mathcal{F}^+ = \{\mathbf{X} \to \mathbf{Y} \mid \mathcal{F} \models \mathbf{X} \to \mathbf{Y}\}$.

Cover: A *cover* for $\mathcal{F}$ is any set $\mathcal{C}$ of FDs with the property that $\mathcal{F}^+ = \mathcal{C}^+$.

- Finally, the appropriate definition for a dependency-preserving decomposition may be made.

Definition: Let $\mathcal{V} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ be subsequences of $\mathbf{U}$. The decomposition of R into $\{\Pi_{\mathbf{W}_1}, \Pi_{\mathbf{W}_2}, \ldots, \Pi_{\mathbf{W}_k}\}$ is *dependency preserving* for $\mathcal{F}$ if there is a cover $\mathcal{C}$ of $\mathcal{F}$ such that each FD in $\mathcal{C}$ embeds into $\Pi_{\mathbf{W}_i}$ for some $i \in \{1, \ldots, k\}$.

  - Say that $\mathcal{V}$ is *dependency preserving* for $\mathcal{F}$ as well.

Example: $\mathcal{C} = \{A \to B, B \to C, C \to B, C \to D, D \to C\}$ is a cover of $\mathcal{F} = \{A \to B, B \to C, C \to D, D \to B\}$ which embeds into

Computing Covers

Question: How does one compute a "good" cover?

- This question is somewhat involved, and will be addressed in more detail later in these slides.
- For now, the assumption that a "good" cover has been found will be made.

BCNF So Far

- A good BCNF decomposition should have two properties:

Lossless: The original schema should be recoverable from the decomposition by knowing only:

(i) the FDs which are embedded into the component schemes of the decomposition, and

(ii) The induced foreign-key dependencies.

  - The simple algorithm which chooses a non-superkey FD and splits the relation into two based upon that FD always delivers a lossless decomposition, as has already been shown.

Dependency preserving: All of the FDs of the original schema should be recaptured in the decomposition.

  - The cover formalism provides the correct mathematical concept of “recaptured”.
  - However, the question remains as to whether this is always possible.

BCNF with Dependency Preservation Is Not Always Possible

Example:

<u>Dept</u>	<u>Proj</u>	Bldg

$\{\text{Dept}, \text{Proj}\} \rightarrow \text{Bldg}, \text{Bldg} \rightarrow \text{Proj}$

- Bldg $\rightarrow$ Proj is not a (super)key dependency.
- $\{\text{Dept}, \text{Proj}\} \rightarrow$ Bldg involves all three attributes.
- There is no "better" cover for these FDs.

Conclusion: There is no lossless and dependency-preserving decomposition of this schema into BCNF.

- Even when there is a lossless and dependency-preserving decomposition into BCNF, it need not be unique.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \rightarrow BCDE, CE \rightarrow D, CD \rightarrow E\}$.

- There are two distinct lossless and dependency-preserving solutions:

- It takes a little thought to see this!

BCNF with Dependency Preservation is Difficult to Decide

- Suppose a "large" schema with many FDs is given, and it is likely to require many steps to decompose it into BCNF.

Theorem: The problem of deciding whether or not a given schema constrained by FDs has a lossless and dependency-preserving decomposition into BCNF is NP-complete. □

- Less formally, this means that no known algorithm is substantially better in the worst case than just trying all possibilities.
- Finding such a *tractable* solution would entail finding tractable solutions to thousands of other problems with the same characteristics.
- The major problem in the case of BCNF decomposition is not only to choose FDs in a suitable order from a cover $\mathcal{C}$ of the set $\mathcal{F}$ (in order to build the "right" *join tree*), but also that alternate covers of $\mathcal{F}$ need to be considered.

Join Trees for Decompositions

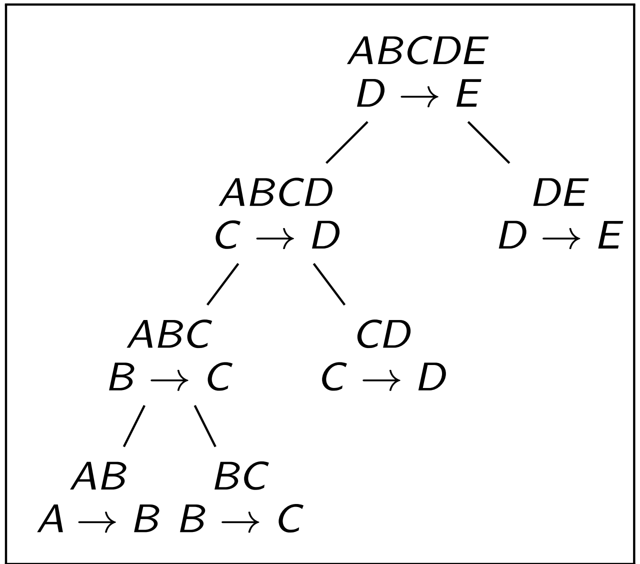
- It is often useful to represent the decomposition process via *join trees*.
- For each interior vertex, the FD under the attribute is that used to define the decomposition.
- For each leaf vertex, the FDs shown are preserved by that projection.
- The leaves represent the final schemes of the decomposition.

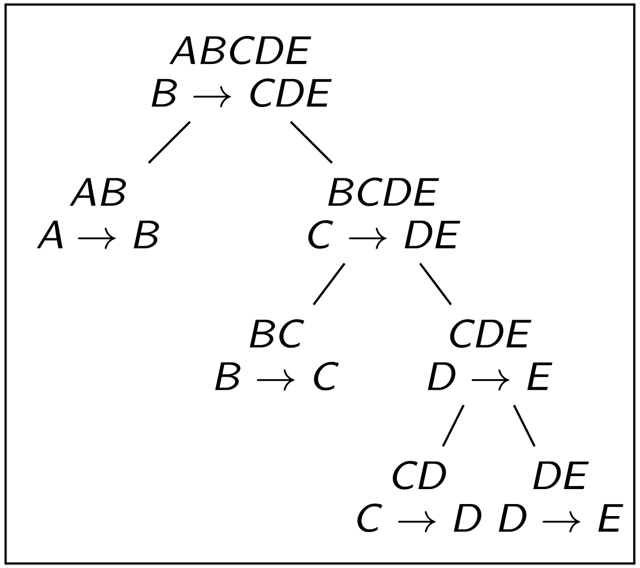
Example:

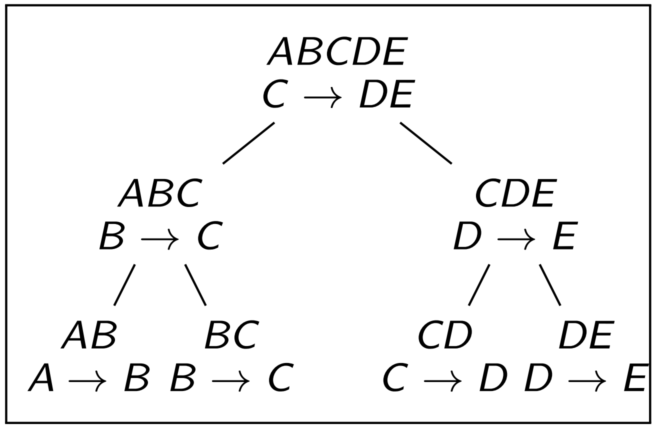
<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E\}$.

- Shown below are three join trees for the same dependency-preserving BCNF decomposition: $\{AB, BC, CD, DE\}$.







Join Trees for Decompositions

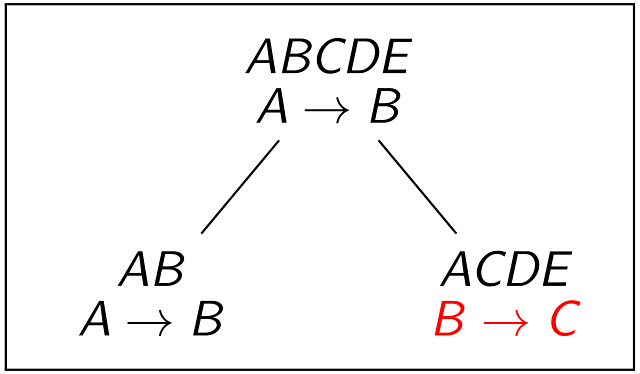
- Join trees can also illustrate that a strategy cannot succeed.

Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \to B, B \to C, C \to D, D \to E\}$.

- The tree below shows that beginning with $A \to B$ as the non-(super)key FD cannot lead to a dependency preserving decomposition, since $B \to C$ is not preserved (and cannot be derived from the other FDs).



- This FD is shown in red to emphasize this.

Decomposition and Foreign-Key Dependencies

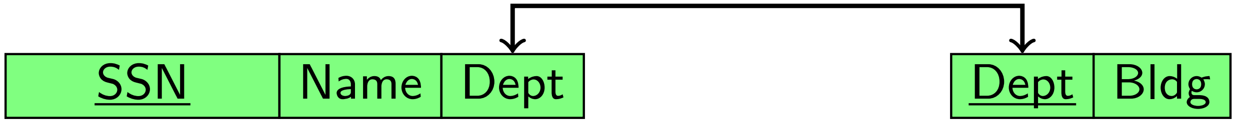
- Most (all?) treatments of normalization ignore foreign-key dependencies.
- The hidden constraint for normalization by decomposition is that the matching columns must agree.

Example:

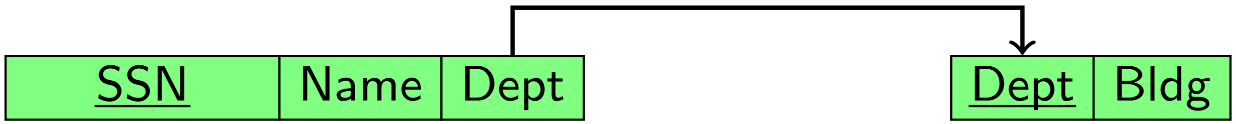
SSN	Name	Dept	Bldg

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\} \quad \text{Dept} \rightarrow \text{Bldg}$

- The values for attributes which occur in both relations must be the same to achieve a truly equivalent schema of two relations.



- However, this is relaxed to a foreign-key dependency in order to:
  (i) Allow expanded modelling and avoid unnecessary information dependencies.
  (ii) Match traditional SQL (which admits foreign-key constraints but not such equality constraints).



Join Trees and Foreign-Key Dependencies

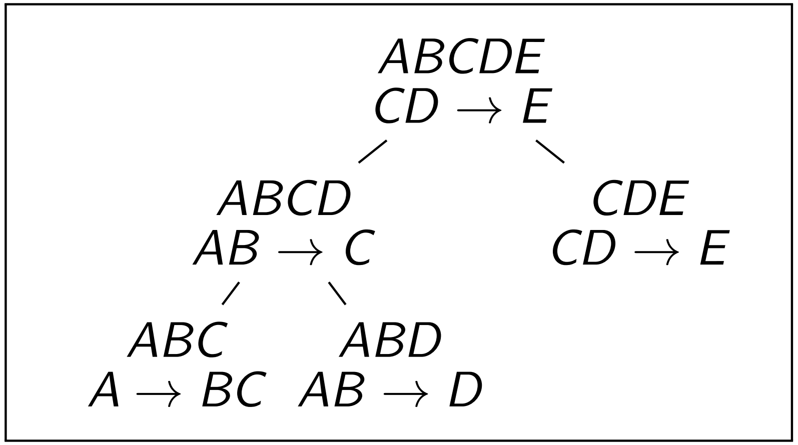
- When the join tree involves more than one decomposition, it may not be possible to identify true foreign keys.

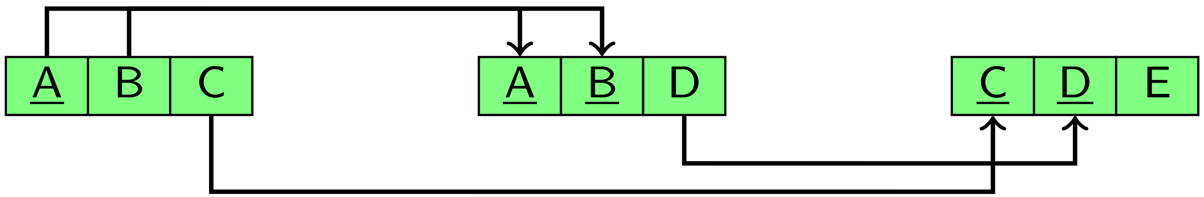
Example:

<u>A</u>	B	C	D	E

$\mathcal{F} = \{A \to BC,\ AB \to D,\ CD \to E\}$.

- In the tree to the right, the key CD for CDE is split into C and D by the decomposition on the left side of the tree.
- They are thus not true foreign keys since they do not reference a primary key.
- As shown in the diagram below, foreign keys may be <u>parts</u> of primary keys in this construction.





- The SQL model of foreign key would have to be extended slightly to accommodate this.

Question: Do such partial foreign keys occur in "real-world" examples?

A "Trick" for Realizing BCNF with Severe Drawbacks

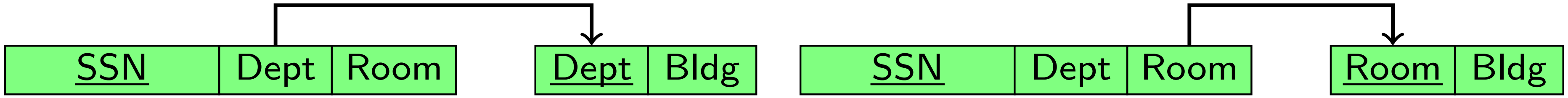
Firm

<u>SSN</u>	Dept	Room	Bldg

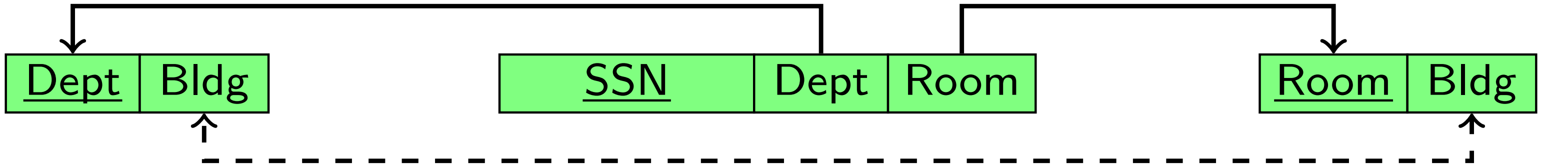
$\text{SSN} \to \{\text{Dept}, \text{Room}\}$

$\text{Dept} \to \text{Bldg} \quad \text{Room} \to \text{Bldg}$

- This schema has two BCNF decompositions, neither of which is dependency preserving.



- It is possible to combine these to obtain a lossless and dependency-preserving BCNF decomposition.



- However, this leads to a *cyclic* (redundant) decomposition which makes integrity checking of updates difficult.
- It is not possible to ensure that the values for Bldg on the two paths $\text{SSN} \to \text{Dept} \to \text{Bldg}$ and $\text{SSN} \to \text{Room} \to \text{Bldg}$ agree without recombining relations.
    - It is necessary to join on Bldg but it is not a key in either relation.

Decompositions Which Are Too Weak or Too Strong

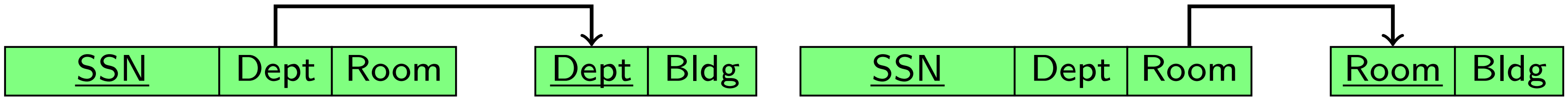
Firm

<u>SSN</u>	Dept	Room	Bldg

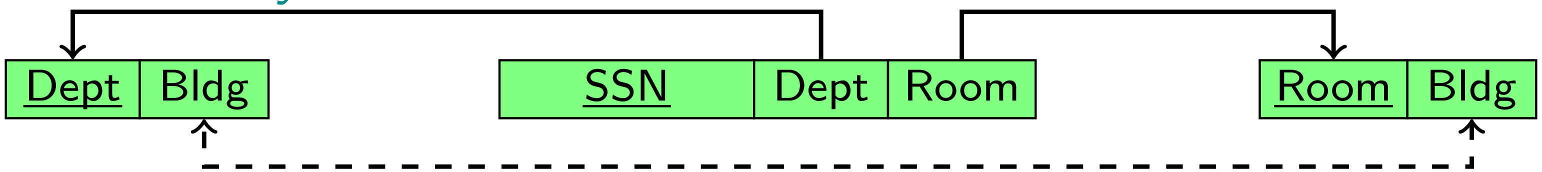
$\text{SSN} \rightarrow \{\text{Dept}, \text{Room}\}$

$\text{Dept} \rightarrow \text{Bldg} \quad \text{Room} \rightarrow \text{Bldg}$

- The following two decompositions are too *weak*, in that they fail to recapture all of the FDs in the original relation.



- Thus, when an update is performed on the decomposed relations, satisfaction of an FD which spans several relations must be verified.
- The following decomposition is too *strong*, in that embodies certain information more than once, and requires a check to ensure that all is consistent. It is *cyclic*.



- Thus, when an update is performed, it must be verified that the third relation is consistent with the other two.
- The goal is to find decompositions which are neither too weak nor too strong.

Cyclicity as a Property of Hypergraphs

- The term *cyclic*, which characterizes redundancy in decompositions, arises from the properties of the *hypergraph* underlying the multi-relation schema which is the result of the decomposition.
- The topic of schema hypergraphs and their properties is beyond the scope of this course, but it is nevertheless useful to be a aware that such a theory exists.
- A decomposition which is not cyclic is called *acyclic*.
- It is thus *acyclic* decompositions which are desirable, which are not too strong; that is, which do not embody redundancy in the representation.
- Such a decomposition must nevertheless be strong enough to be dependency preserving.
- For this course, it will suffice to characterize the combination of acyclicity and dependency preservation indirectly, primarily via *full independence*.

Comment: Full independence is not standard terminology in the field, but is introduced here as a convenient way to recapture acyclicity without resorting to the explicit use of hypergraphs.

Formalization of Full Independence

Context: $R = (A_1, A_2, \ldots, A_k)$; FDs $\mathcal{F}$; $\{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ subsequences.

Local database: For $i \in \{1, \ldots, k\}$, the database N_i on attributes $\mathbf{W}_i$ is called a *local database* for $\mathbf{W}_i$ if it is the projection onto $\mathbf{W}_i$ of some M_R for R which satisfies the constraints of $\mathcal{F}$: $\pi_{\mathbf{W}_i}(M_R) = N_i$.

Join compatibility: A sequence $(N_1, N_3, \ldots, N_k)$ with N_i a local database for $\mathbf{W}_i$ is *join compatible* if any two $\{N_i, N_j\}$ agree on their common columns: for any $i, j \in \{1, \ldots, k\}$, $\pi_{\mathbf{W}_i \cap \mathbf{W}_j}(N_i) = \pi_{\mathbf{W}_i \cap \mathbf{W}_j}(N_j)$

Fully independent decomposition: $\{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ is *fully independent* for $\langle R, \mathcal{F} \rangle$ if any join-compatible sequence of local databases joins to a database on R which satisfies $\mathcal{F}$.

- Join compatibility $\Rightarrow$ each $\Pi_{\mathbf{W}_i}$ may be updated independently of the others... ... provided only that the common columns do not change.
- This is <u>the</u> goal of decomposition.

Example: The "trick" solution on Slides 44-45 is not fully independent.

Fact: A decomposition is fully independent iff it is dependency preserving and (its underlying hypergraph is) acyclic. $\square$

Testing for Full Independence

Theorem: Let $\langle R, \mathcal{F} \rangle$ be a database schema. A lossless decomposition $\mathfrak{D} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ of $\langle R, \mathcal{F} \rangle$ is fully independent iff there is a dependency-preserving join tree whose leaves are exactly the members of $\mathfrak{D}$ and whose root is R. $\square$

Corollary: Any dependency-preserving decomposition of $\langle R, \mathcal{F} \rangle$ obtained from the BCNF decomposition algorithm is fully independent. $\square$

Example: The decomposition $\{ABC, ABD, CDE\}$ of the schema $\langle ABCDE, \{A \rightarrow BC,\ AB \rightarrow D,\ CD \rightarrow E\} \rangle$ is fully independent. (See Slide 43.)

Fact: Context as in the theorem above, if any proper subset of $\mathfrak{D}$ forms a lossless decomposition, then $\mathfrak{D}$ cannot be fully independent. $\square$

Example: The decomposition $\{BD, BR, DRS\}$ of the schema $\langle BDRS, \{S \rightarrow DR,\ D \rightarrow B,\ R \rightarrow B\} \rangle$ is not fully independent, since both $\{BD, DRS\}$ and $\{BR, DRS\}$ form lossless decompositions. (See Slides 44-45.) $\square$

Evaluation of the BCNF-by-Decomposition Approach

Question: How useful is the BCNF-by-decomposition approach?

☝ It always delivers lossless decompositions.

☟ It may not always deliver dependency preservation, but:

- ☝ When it does, the result is fully independent (no redundancy).
- ☝ Success or failure is always clear from the process.

☝ Such fully independent decompositions addresses the three normalization issues effectively.

☞ It may not provide true foreign keys.

- ... but this is an issue of lack of support of a relatively simple feature in current SQL, not a fundamental design issue.

☟ In the worst case, exhaustive search is required to determine whether a lossless and dependency-preserving decomposition exists.

Bottom line: It may not always work, but when it does, it works well, and the quality of the result is known from the process itself.

Constraints Induced by Normalization

- The goal of the normalization process is to obtain designs which may be implemented using SQL.
- A relational schema (in SQL), there are two kinds of constraints:
  (a) FDs (keys only) are specified on individual relations, never across relations.
  (b) Foreign-key dependencies connect matching attributes on different relations.
- The decomposition approach satisfies (a) by construction.
- Clearly, the "trick" schema of Slides 44-45 involves other dependencies.
- Fortunately, the decomposition algorithm can never introduce such dependencies; it never embodies redundancy in its representation of the original schema.
- Unfortunately, it can fail to be dependency preserving, and so can underrepresent the original schema.

Question: Is there an alternative which avoids this drawback?

Third Normal Form — 3NF — a Compromise

- Third Normal Form (3NF) may be viewed as a compromise.
- In contrast to BCNF, a lossless and dependency decomposition into 3NF always exists.
- Fix a relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R.

Prime attributes: An attribute $A_i \in \mathbf{U}$ is *prime* (for $\langle R, \mathcal{F} \rangle$) if it is a member of some candidate key.

3NF: $\mathcal{F}$ is in *Third Normal Form* (*3NF*) if for any (by convention, fully nontrivial) $\mathbf{X} \to \mathbf{Y} \in \mathcal{F}$, either

(a) $\mathbf{X}$ is a superkey for $\mathcal{F}$, or

(b) every $A \in \mathbf{Y} \setminus \mathbf{X}$ is a prime attribute.

Example:

<u>Dept</u>	<u>Proj</u>	Bldg

$\{\text{Dept}, \text{Proj}\} \to \text{Bldg}$, $\text{Bldg} \to \text{Proj}$

is not in BCNF, but it is in 3NF since Proj is a prime attribute.

Finding a 3NF Representation of a Schema

- In contrast to the approach for BCNF, the most common algorithm for realizing 3NF is based upon *synthesis*.
- Rather than decomposing a single relation, smaller relations are built up using properties of the underlying set of constraints.
- For this synthesis approach to work, the set of FDs must be put into a special form, in which the set of FDs is *canonical*.
- Algorithms for accomplishing this task are discussed next.

Canonical Sets of FDs

General Context: Let $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ be a relation scheme with with FDs $\mathcal{F}$.

RHS-simple: The FD $\mathbf{X} \to \mathbf{Y}$ is *RHS-simple* if $\mathbf{Y}$ consists of just one attribute; *i.e.*, it is of the form $\mathbf{X} \to A$.

LHS-reduced: The FD $\mathbf{X} \to \mathbf{Y}$ is *LHS-reduced* (or *full*) in $\mathcal{F}$ if for no proper subsequence $\mathbf{X}' \subsetneq \mathbf{X}$ is it the case that

$$((\mathcal{F} \setminus \{\mathbf{X} \to \mathbf{Y}\}) \cup \{\mathbf{X}' \to \mathbf{Y}\})^+ = \mathcal{F}^+.$$

Nonredundancy: The set $\mathcal{F}$ of FDs is *nonredundant* if for no proper subset $\mathcal{F}' \subsetneq \mathcal{F}$ is it the case that $\mathcal{F}'^+ = \mathcal{F}^+$.

Canonicity: The set $\mathcal{F}$ of FDs is *canonical* if each of its members is RHS-simple and LHS-reduced in $\mathcal{F}$, and in addition, $\mathcal{F}$ is nonredundant.

Note on terminology: The textbook calls a canonical set of FDs *minimal*, but this terminology is nonstandard and can easily be confused with *minimum*, which has an entirely different meaning.

Computing a Canonical Cover

Canonical cover: A *canonical cover* of $\mathcal{F}$ is a cover $\mathcal{C}$ which is also canonical.

Algorithm for computing a canonical cover:

(1) Decompose each FD into RHS-simple form.

(2) LHS-reduce each FD.

(3) Test each remaining FD for redundancy of the resulting set of FDs, removing the ones which are not needed to preserve the closure.

- Steps (2) and (3) may involve choices.

Example: $\mathcal{F} = \{A \to BC,\ B \to C,\ AB \to D,\ AC \to D\}$.

Step 1: $\{A \to B,\ A \to C,\ B \to C,\ AB \to D,\ AC \to D\}$.

Step 2: $\{A \to B,\ A \to C,\ B \to C,\ A \to D,\ A \to D\}$

$= \{A \to B,\ A \to C,\ B \to C,\ A \to D\}$.

Step 3: $\{A \to B,\ A \to C,\ B \to C,\ A \to D\}$ reduces to $\{A \to B,\ B \to C,\ A \to D\}$ (since $\{A \to B,\ B \to C\} \models A \to C$).

Example of Computing a Canonical Cover

Example: $\mathcal{F} = \{A \rightarrow B,\ ABCD \rightarrow E,\ EF \rightarrow GH,\ ACDF \rightarrow EG\}$.

Step 1:

$$\{A \rightarrow B,\ ABCD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H,\ ACDF \rightarrow E,\ ACDF \rightarrow G\}.$$

Step 2: $\{A \rightarrow B,\ ACD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H,\ ACD \rightarrow E,\ ACDF \rightarrow G\}$
$= \{A \rightarrow B,\ ACD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H,\ ACDF \rightarrow G\}$

Step 3: $\{A \rightarrow B,\ ACD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H\}$
(since $\{ACD \rightarrow E,\ EF \rightarrow G\} \models ACDF \rightarrow G$).

- In general, Steps 2 and 3 may be very complex, but they can often be solved by inspection for small examples.
- Implementing them involves using an inference algorithm for FDs.
- In implementation, the use of RHS-simple form may also be avoided ...
    - ... but it is perhaps easier for humans to work with that form.

The Synthesis Algorithm

Consolidation: Let $\mathcal{G}$ be any set of FDs. The *consolidation* $\mathsf{Consol}\langle\mathcal{G}\rangle$ of $\mathcal{G}$ is formed by combining all FDs of $\mathcal{G}$ with the same left-hand side into one.

Example: The consolidation of

$\mathcal{G} = \{A \to B,\ ACD \to E,\ EF \to G,\ EF \to H\}$ is

$\mathsf{Consol}\langle\mathcal{G}\rangle = \{A \to B,\ ACD \to E,\ EF \to GH\}$

Context: $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ a relation scheme with FDs $\mathcal{F}$,

The 3NF algorithm:

(1) Construct a canonical cover $\mathcal{C}$ for $\mathcal{F}$.

(ii) Define $\mathsf{Schemes}'_{\mathsf{3NF}}\langle\mathcal{C}\rangle = \{\mathbf{X} \cup \mathbf{Y} \mid \mathbf{X} \to \mathbf{Y} \in \mathsf{Consol}\langle\mathcal{C}\rangle\}$

(iii) Define $\mathsf{Schemes}_{\mathsf{3NF}}\langle\mathcal{C}\rangle$ to be the subset of $\mathsf{Schemes}'_{\mathsf{3NF}}\langle\mathcal{C}\rangle$ obtained by removing any relation which is subsumed by another.

Theorem: For any canonical cover $\mathcal{C}$ of $\mathcal{F}$, $\mathsf{Schemes}_{\mathsf{3NF}}\langle\mathcal{C}\rangle$ is in 3NF and dependency preserving for $\mathcal{F}$. $\square$

Example: Let $\mathcal{G} = \{A \to B,\ ABCD \to E,\ EF \to GH,\ ACDF \to EG\}$. Then $\mathcal{G}' = \{A \to B,\ ACD \to E,\ EF \to G,\ EF \to H\}$ is a canonical cover for $\mathcal{G}$ and $\mathsf{Schemes}'_{\mathsf{3NF}}\langle\mathcal{G}\rangle = \mathsf{Schemes}_{\mathsf{3NF}}\langle\mathcal{G}\rangle = \{AB, ACDE, EFGH\}$.

Subsumption in the Synthesis Algorithm

- It is important to be clear about subsumption.

Subsumption: Schema R_1 *subsumes* schema R_2 (and R_2 is *subsumed by* R_1) if the attributes of R_2 are a subset of those of R_1.

Example: $\mathcal{G} = \{AB \to C,\ C \to A\}$.

- $\mathcal{G}$ is a canonical cover of itself.
- $\text{Schemes}'_{\text{3NF}}\langle \mathcal{G} \rangle = \{ABC, AC\}$.
- $\text{Schemes}_{\text{3NF}}\langle \mathcal{G} \rangle = \{ABC\}$, since ABC subsumes AC.

- It is natural to remove AC in this case, since all of the information which it embodies is also found in ABC.

Further Examples of the Synthesis Algorithm

- Many of the simple examples which were presented for BCNF produce the same result with the 3NF synthesis algorithm.

Example:

SSN	Name	Dept	Bldg

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\} \quad \text{Dept} \rightarrow \text{Bldg}$

- The 3NF synthesis algorithm produces:

SSN	Name	Dept

Dept	Bldg

  - But the foreign-key dependencies must be identified another way.

Example:

Firm

SSN	Name	Dept	Proj	Bldg

$\text{SSN} \rightarrow \{\text{Name}, \text{Dept}\}$
$\text{Dept} \rightarrow \text{Proj}, \ \text{Proj} \rightarrow \text{Bldg}$

- In this example, only the dependency-preserving decomposition is obtained, as expected.

SSN	Name	Dept

Dept	Proj

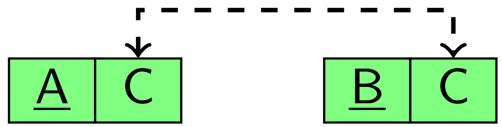
Proj	Bldg

Losslessness and the Synthesis Algorithm

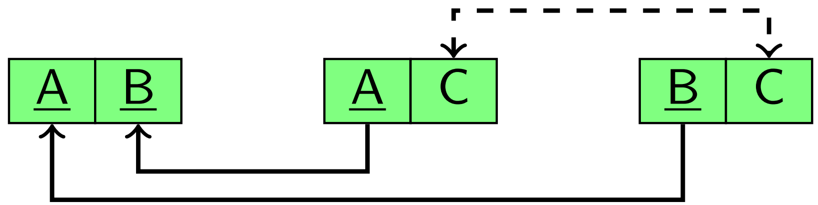
- The synthesis algorithm does not necessarily guarantee lossless decompositions.

Example: A B C $\quad A \to C \quad B \to C$

- The 3NF synthesis algorithm produces the following schema:



- ... which is not lossless since C is not a key for either schema.
- A solution is to add a third relation which contains a key for the original relation:



- This idea applies in general.

Losslessness + Dependency Preservation

- The key to the extension is based upon the following result.

Context:

- Relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R.
- $\mathcal{V} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ subsequences of $\mathbf{U}$.

Theorem: If $\mathcal{V}$ is dependency preserving, then the decomposition of R into $\{\Pi_{\mathbf{W}_1}, \Pi_{\mathbf{W}_2}, \ldots, \Pi_{\mathbf{W}_k}\}$ is lossless iff $\mathbf{W}_i$ is a superkey (*i.e.*, contains a key) for R for some $i \in \{1, \ldots, k\}$. $\square$

Remark: In the above, the superkey must be for all of R, and not just the attributes which occur in $\mathcal{F}$.

Example:

<u>A</u>	<u>B</u>	C

$B \to C$

- The key of R is AB, not just B.

Keys Again

- It is very important in this context to understand what is meant by a key when there are attributes which do not occur in any FD.

Context:

- Relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R.
- $P = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ subsequences of $\mathbf{U}$.

Superkey again and precisely: Call $\mathbf{K}$ a *superkey* for $\langle R, \mathcal{F}\rangle$ if every instance M_R of R which is constrained by $\mathcal{F}$ and every pair $t_1, t_2 \in M_R$, if $t_1[\mathbf{K}] = t_2[\mathbf{K}]$, then $t_1 = t_2$.

- A *key* is a minimal superkey, as usual.

Definition: $\text{Attrset}(\mathcal{F}) = \bigcup\{\mathbf{X} \cup \mathbf{Y} \mid \mathbf{X} \to \mathbf{Y} \in \mathcal{F}\}$.

Observation: $\mathbf{K}$ is a superkey for $\langle R, \mathcal{F}\rangle$ iff $\mathbf{K} = \mathbf{K}_1 \cup \mathbf{K}_2$ with:

(i) $\mathcal{F} \models \mathbf{K}_1 \to \text{Attrset}(\mathcal{F})$, and

(ii) $\mathbf{K}_2 = \mathbf{U} \setminus \text{Attrset}(\mathcal{F})$. $\square$

Extending the Synthesis Algorithm to Achieve Losslessness

Context: Relation scheme $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$, with FDs $\mathcal{F}$ a set of FDs on R.

The Lossless 3NF algorithm:

(1) Construct a canonical cover $\mathcal{C}$ for $\mathcal{F}$.

(ii) Define $\text{Schemes}'_{\text{3NF}}\langle\mathcal{C}\rangle = \{\mathbf{X} \cup \mathbf{Y} \mid \mathbf{X} \to \mathbf{Y} \in \text{Consol}\langle\mathcal{C}\rangle\}$

(iii) Define $\text{Schemes}_{\text{3NF}}\langle\mathcal{C}\rangle$ to be the subset of $\text{Schemes}'_{\text{3NF}}\langle\mathcal{C}\rangle$ obtained by removing any relation which is subsumed by another.

(iv) Define $\text{Schemes}_{\overline{\text{3NF}}}\langle\mathcal{C}\rangle$ to be $\text{Schemes}_{\text{3NF}}\langle\mathcal{C}\rangle$ if $\text{Schemes}_{\text{3NF}}\langle\mathcal{C}\rangle$ if that set already contains a superkey for R, and $\text{Schemes}_{\text{3NF}}\langle\mathcal{C}\rangle \cup \mathbf{K}$ for some key $\mathbf{K}$ of $\langle R, \mathcal{F}\rangle$ otherwise.

Theorem: For any canonical cover $\mathcal{C}$ of $\mathcal{F}$, $\text{Schemes}_{\overline{\text{3NF}}}\langle\mathcal{C}\rangle$ is a lossless and dependency preserving decomposition of R for $\mathcal{F}$. $\square$

Example of the Synthesis Algorithm and Losslessness

Example: Let $\mathcal{G} = \{A \rightarrow B,\ ABCD \rightarrow E,\ EF \rightarrow GH,\ ACDF \rightarrow EG\}$.

- As shown on Slide 56, $\mathcal{G}' = \{A \rightarrow B,\ ACD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H\}$ is a canonical cover for $\mathcal{G}$ and
$$\mathrm{Schemes}'_{\mathrm{3NF}}\langle\mathcal{G}\rangle = \mathrm{Schemes}_{\mathrm{3NF}}\langle\mathcal{G}\rangle = \{AB, ACDE, EFGH\}.$$
- However, this decomposition is not lossless, since no member of $\{AB, ACDE, EFGH\}$ is a superkey for R.
- To remedy this, first identify the unique key of R as $ACDF$.
  - None of the attributes in $ACDF$ occur on the right-hand side of any FD in $\mathcal{G}$, and so each must be part of any key.
  - All other attributes are derivable from these, so $ACDF$ must be unique as a key.
- Thus, $\mathrm{Schemes}_{\overline{\mathrm{3NF}}}\langle\mathcal{G}\rangle = \mathrm{Schemes}_{\mathrm{3NF}}\langle\mathcal{G}\rangle \cup \{ACDF\}$
$$= \{AB, ACDE, EFGH, ACDF\}.$$

Properties of Synthesized 3NF Schemata

- The 3NF synthesis algorithm delivers as advertised.
- It is a nice theoretical result.

Questions: Are the schemata which it synthesizes otherwise free of problems?

Answer: Not at all.

Example:

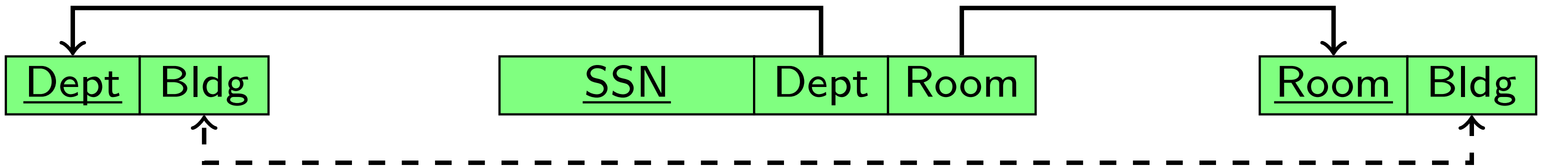
Firm

SSN	Dept	Room	Bldg

$\text{SSN} \to \{\text{Dept}, \text{Room}\}$

$\text{Dept} \to \text{Bldg} \quad \text{Room} \to \text{Bldg}$

- Recall the following lossless and dependency-preserving but cyclic decomposition obtained by the BCNF algorithm plus a "trick".



- This is exactly the 3NF schema which the synthesis algorithm delivers.

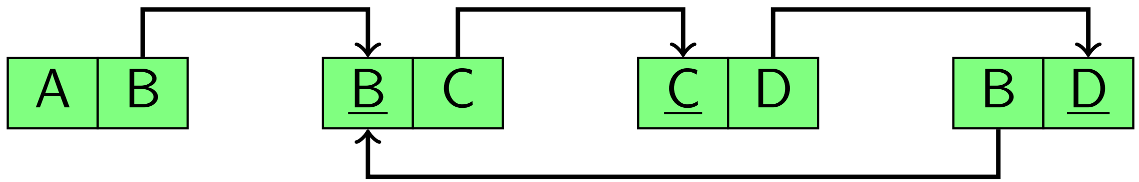
Theorem: The synthesis algorithm need not deliver fully independent decompositions, even when the result is lossless. □

- In contrast to the decomposition algorithm, the synthesis algorithm provides no flags as to potential problems.

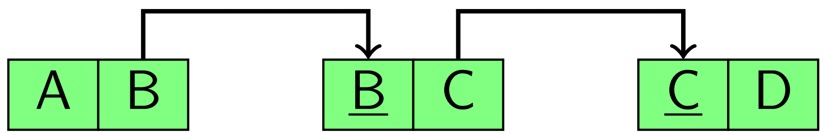
Further Examples of Cyclicity in 3NF Synthesis

Example: | <u>A</u> | B | C | D | $\mathcal{F} = \{A \to B, B \to C, C \to D, D \to B\}$.

- $\mathcal{F}$ is its own canonical cover.
- The 3NF synthesis algorithm delivers the following cyclic result:

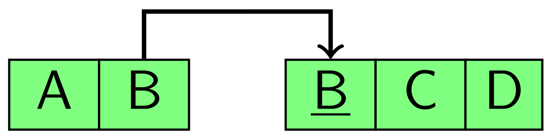


- Using the cover $\mathcal{F}' = \{A \to B, B \leftrightarrow C, C \leftrightarrow D\}$ instead, an acyclic decomposition is obtained:



☞ There is nothing in the 3NF synthesis algorithm which flags cyclicity.

- There is a better acyclic realization which is not the result of applying the algorithm for any canonical cover:



- It is better because $\{AB, BCD\}$ is already in BCNF.

Testing a 3NF Decomposition for Acyclicity

- The result on Slide 48 is not limited to BCNF decompositions.
- It may be applied to any decompositions, including 3NF:

Corollary: Let $\mathfrak{D} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_k\}$ be a dependency-preserving 3NF synthesis of $\langle R, \mathcal{F} \rangle$. Then $\mathfrak{D}$ is lossless and fully independent iff there is a join tree whose leaves are exactly the members of $\mathfrak{D}$ and whose root is R. $\square$

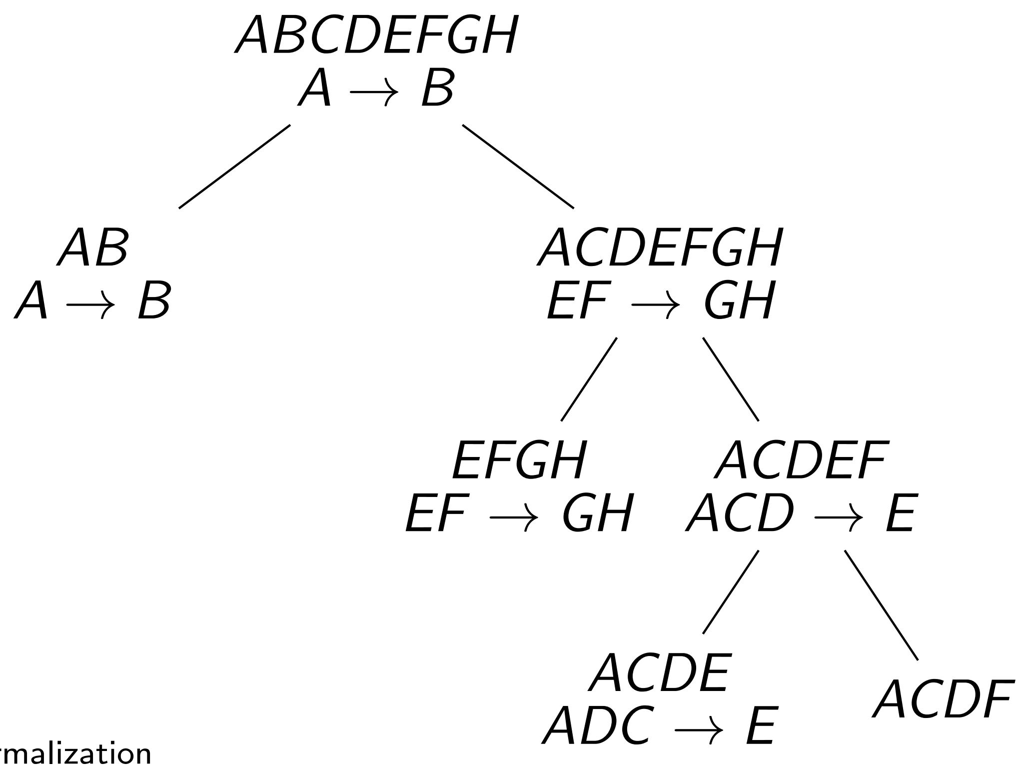
Algorithm: Try to build suitable join trees "bottom up", starting by combining losslessly joinable pairs of elements from $\mathfrak{D}$.

- If the number k of schemes is relatively small, the testing may be done by hand.
- Use the property of the theorem on Slide 29 to test for joinabilty.

Testing for Acyclicity — Example

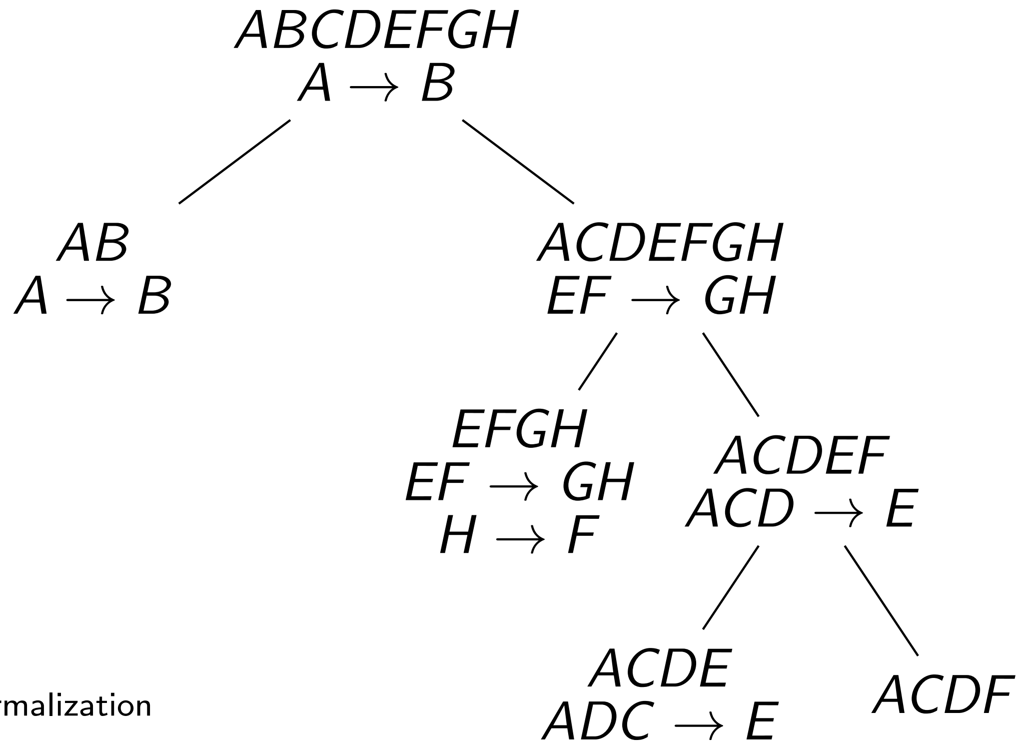
- Shown below is a join tree for the lossless synthesis of the problem on Slide 63 with $\text{Schemes}_{\overline{\text{3NF}}}\langle\mathcal{G}\rangle = \text{Schemes}_{\text{3NF}}\langle\mathcal{G}\rangle \cup \{ACDF\}$
$= \{AB, ACDE, EFGH, ACDF\}$.

- Therefore, this synthesis is not only lossless (as already known) but acyclic (and so fully independent).



Testing for Acyclicity — Example 2

- Suppose that the original example also included the FD $H \to F$:
  $\mathcal{G}' = \{A \to B,\ ABCD \to E,\ EF \to GH,\ ACDF \to EG,\ H \to F\}$.
- $\text{Schemes}_{\overline{3NF}}\langle \mathcal{G}' \rangle = \{AB, ACDE, EFGH, ACDF\}$ is also a synthesis.
- However, $EFGH$ is no longer in BCNF.
- Nevertheless, the join tree is the same, and so the synthesis is lossless and acyclic (and so fully independent).



Decompositions Which Share Dependencies

- A cyclic decomposition is not the only problem which may arise from 3NF synthesis.

Example:

Part	Site	Distributor	Address

$$\{\text{Part}, \text{Site}\} \to \text{Distributor}$$
$$\text{Distributor} \to \{\text{Site}, \text{Address}\}$$

- The 3NF synthesis algorithm yields

Part	Site	Distributor

Site	Distributor	Address

- The two relations have the *shared dependency* Distributor $\to$ Site.
- The common attributes $\{\text{Distributor}, \text{Site}\}$ form a superkey, but not a candidate key, of the relation on the right.
- This is highly undesirable; the connection does not define a foreign-key dependency; an FD must be maintained within the common attributes!
- A better realization, without these problems, but not found by the basic synthesis algorithm, is:

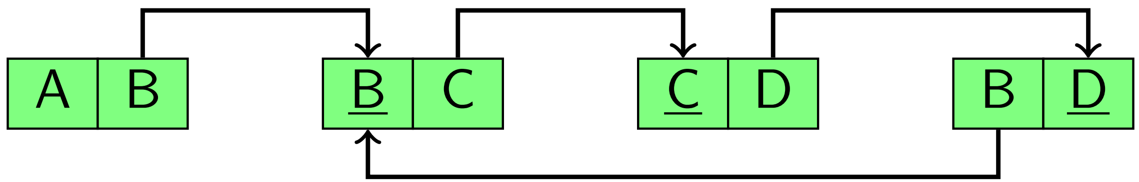
Part	Site	Distributor

Distributor	Address

- This problem cannot occur with the BCNF decomposition algorithm.

Improving the 3NF Synthesis Algorithm

- There is an improved 3NF synthesis algorithm which is substantially more complex than the original one given here.
  - Tok-Wang Wing, Frank W. Tompa, and Tiko Kameda, An Improved Third Normal Form for Relational Databases, *ACM Transactions on Database Systems* **6**(2), 1981, pp. 329-346.
- It removes *superfluous* attributes and so can fix the decomposition on the previous slide, as well as the decomposition

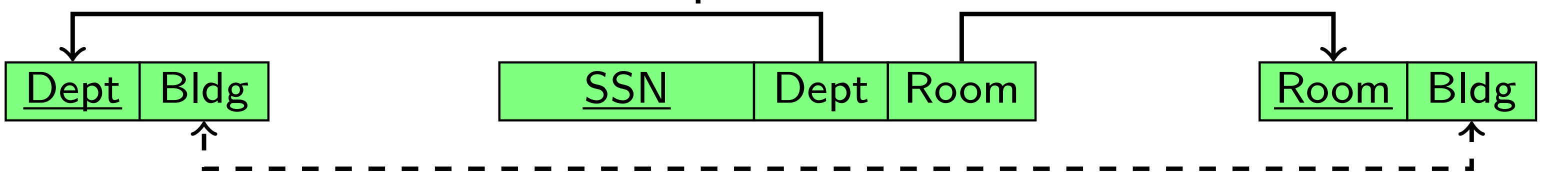


of

<u>A</u>	B	C	D

$\mathcal{F} = \{A \to B, B \to C, C \to D, D \to B\}$

by removing the entire redundant scheme BD, for example.
- However, it cannot fix the decomposition



of

Firm

<u>SSN</u>	Dept	Room	Bldg

SSN $\to$ {Dept, Room}

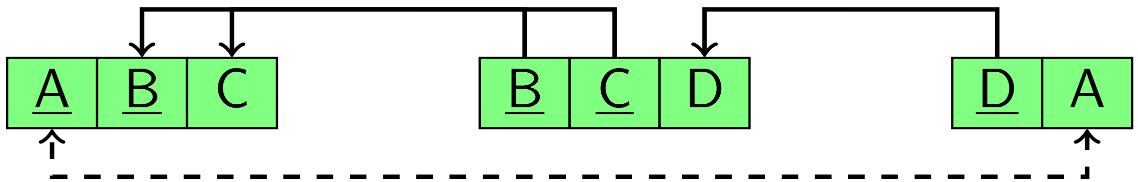
Dept $\to$ Bldg   Room $\to$ Bldg

since there are no superfluous attributes.

Repairing Cyclic Solutions

- Some cyclic solutions (*i.e.*, dependency preserving but not fully independent) may be repaired by removing superfluous attributes or relations, while others are unrepairable, as illustrated on Slide 70.
- Here is another example of a cyclic decomposition which cannot be repaired in this way to achieve full independence.

Example: | A | B | C | D | $\mathcal{F} = \{AB \rightarrow C,\ BC \rightarrow D, D \rightarrow A\}$.



- In such cases, the design must be altered in more complex ways order to achieve acyclicity.

A Comment on Testing for Losslessness

- A general algorithm for testing a decomposition for losslessness is given in the textbook.
- This is a very general algorithm, and useful for computer implementation.
- However, it works <u>too</u> well, in a sense.
  - It does not detect cyclicity, as illustrated in several examples.
- For small examples which are done by hand, it is better to work with join trees.
  - If a dependency-preserving join tree is constructed for a 3NF synthesis, and there are schemata which were not used in the join tree, then the synthesis is cyclic.

Comparison: Decomposition and Synthesis Approach

Decomposition into BCNF:

- ☟ Involves many choices, with a possibly different solution for each choice.
  - Choose a cover for the FDs.
  - Choose an FD upon which to base a decomposition step.
- ☟ Does not always yield a dependency-preserving solution.
- ☝ When it produces a dependency-preserving solution, that solution is also guaranteed to be fully independent.
- ☝ The solution never contains shared depdendencies.
- ☝ Provides direct information about foreign-keys and partial foreign keys.
- ☝ Provides clear information about what has succeeded and what has failed.

Comparison: Decomposition and Synthesis Approach — 2

Synthesis for 3NF:

- ☞ Only involves a choice of cover, with a possibly different solution for each choice.
  - But the quality of the solution is highly dependent upon this choice.
- Always yields a dependency-preserving solution.
- May produce a solution which is cyclic and/or which involves shared dependencies, even when solutions which avoid these problems are possible, with absolutely no indication whatsoever that this is the case.
  - The improved algorithm of Wing *et al* provides a fix in some cases, but is far more complex.
- May not find the "best" solution (see Slide 65).
- Foreign keys and partial foreign keys not identified by the algorithm.
- Provides no information about the quality of the solution.

The Bottom Line for the 3NF Synthesis Algorithm

- It is a nice theoretical result, but it may produce a normalization, which while 3NF, dependency preserving, and lossless, nevertheless embodies many highly undesirable features including:
  - cyclicity (not fully independent);
  - dependency sharing.
- If 3NF synthesis is to be used, it is mandatory to go to something more complex, such as the improved algorithm of Ling *et al*.
- Even then, it is necessary to examine the result for other types of cyclicity.
  - This examination may be very complex and identifying a suitable repair may be equally complex.
- There is far more to 3NF synthesis than is found in textbooks on database management.
- Used alone, it is not a suitable design tool.

Enforcing Non-Key FD and Non-FK Inclusion Constraints

- Sometimes, the normalization process is not completely satisfactory, and constraints not representable in SQL must be enforced.
- There are two main ways to do this.

Scout's-honor programming: Require the application programs to enforce the constraints in any updates they make.

- ☟ Such a distributed, trusting approach is likely to fail sooner or later, and is best avoided.
- ☟ One bad apple spoils the barrel.

Triggers: Implement enforcement of the constraints in special SQL directives.

- A *trigger* is an imperative procedure which is executed in conjunction with an SQL update.
- ☝ In principle, almost any constraint may be enforced via triggers.
- ☝ Triggers provide a single, centralized solution.
- ☟ Triggers can be extremely inefficient and impact performance unless the database systems is configured properly for them.
- ☟ Trigger code is often far from transparent with understanding of the associated constraint often difficult.

Second Normal Form — 2NF

- This form is largely of historical interest.

Context: $R = (A_1, A_2, \ldots, A_k) = \mathbf{U}$ a relation scheme with FDs $\mathcal{F}$,

Full dependency: Let $\mathbf{W} \subseteq \mathbf{U}$ and $A \in \mathbf{U}$. Say that A is *fully dependent* upon $\mathbf{W}$ if the following two conditions are met:

- $\mathcal{F}^+ \models \mathbf{W} \to A$.
- For no proper subsequence $\mathbf{W}' \subsetneq \mathbf{W}$ is it the case that $\mathcal{F}^+ \models \mathbf{W}' \to A$.

2NF: R is in *second normal form (2NF)* for $\mathcal{F}$ if every nonprime attribute is fully dependent upon every candidate key.

Example:

<u>A</u>	<u>B</u>	C	D

$\mathcal{F} = \{AB \to C, B \to D\}$.

- D is a nonprime attribute which is not fully dependent upon the key AB.

Theorem: Every schema which is in 3NF is also in 2NF. □

First Normal Form — 1NF

- First normal form simply states that all domains are atomic.
- It has already been discussed in in the introductory slides on the the relational model.
- This slide is reproduced next.

First Normal Form

- It would be desirable to be able to decompose attributes such as address into tuples of subattributes, as illustrated below.
- With the classical relational model, this is not possible.
- In so-called *first normal form (1NF)*, all domains are atomic.
- In the Employee relation, the attribute Address could be replaced by the three attributes Street, City, and State, but the ability to refer to Address as their composite would be lost.
- A representation as illustrated below is available in some relational systems as an *object-relational* extension.
- It has even become part of the latest SQL standard.
- However, support is still far from universal or uniform.
- Object-relational extensions will not be considered in this course.

Employee

FName	MInit	LName	<u>SSN</u>	BDate	Address	Sex	Salary	Super_SSN	DNo

Address

Street	City	State