Part 10: Introduction to Relational Normal Forms

References:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Ed., Ch. 14, “Functional Dependencies and Normalization for Relational Databases” Ch. 15, “Relational Database Design Algorithms and Further Dependencies”
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed., Ch. 7, “Relational Database Design”
- Ramakrishnan/Gehrke: Database Management Systems, 2nd Ed., Mc-Graw Hill, 2000. Ch. 15, “Schema Refinement and Normal Forms”
- Simsion/Witt: Data Modeling Essentials, 2nd Edition. Coriolis, 2001. Ch. 2: “Basic Normalization”, Ch. 8: “Advanced Normalization”.
- Kemper/Eickler: Datenbanksysteme (in German), Oldenbourg, 1997. Ch. 6, “Relationale Entwurfstheorie”
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German). Teubner, 1997.
- Kent: A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM 26(2), 120–125, 1983.
- Thalheim: Dependencies in Relational Databases. Teubner, 1991.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

- Detect bad relational database designs (that contain redundancies).
- Determine functional dependencies.
- Check whether a given table is in BCNF for given functional dependencies.
- Detect redundancy and normalization problems already during the conceptional design in the ER-model.

Overview

1. Introduction (Anomalies)

2. Functional Dependencies

3. BCNF

Introduction (1)

- Relational database design theory is based mainly on a class of constraints called “Functional Dependencies” (FDs). FDs are a generalization of keys.
- This theory defines when a relation is in a certain normal form (e.g. Third Normal Form, 3NF) for a given set of FDs.
- It is usually bad if a schema contains relations that violate the conditions of a normal form.

  However, there are exceptions and tradeoffs.

Introduction (2)

- If a normal form is violated, data is stored redundantly, and information about different concepts is intermixed. E.g. consider the following table:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB	Brass	9404
42232	DS	Brass	9404
31822	IS	Spring	9429

- The phone number of "Brass" is stored two times. In general, the phone number of an instructor will be stored once for every course he/she teaches.

Introduction (3)

- Of course, it is no problem if a column contains the same value two times (e.g. consider a `Y/N` column).
- But in this case, the following holds: If two rows have the same value in the column `INAME`, they must have the same value in the column `PHONE`.
- This is an example of a functional dependency: `INAME` $\longrightarrow$ `PHONE`.
- Because of this rule, one of the two `PHONE` entries for `Brass` is redundant.

Introduction (4)

- Table entries are redundant if they can be reconstructed from other table entries and additional information (like the FD in this case).

  E.g. if an employee table contains the date of birth, the additional column `AGE` would be redundant: The age can be computed from the date of birth (and the knowledge about today's date).

- Redundant information in database schemas is bad:
  - ◇ Storage space is wasted.
  - ◇ If the information is updated, all redundant copies must be updated. If one is not careful, the copies become inconsistent (Update Anomaly).

Introduction (5)

- Redundant information is sometimes convenient for easy query formulation (e.g. a precomputed join).
- But in relational databases, one can define virtual tables (views) that are computed by a query.
- Since the contents of a view is not explicitly stored and not directly updated, redundant information is no problem for views.

  The entire view is redundant, since it is computed from the stored relations.

Introduction (6)

- Sometimes, redundant information might also be needed for efficient query evaluation.
- There is a tradeoff: Storing redundant information is bad, but slow query evaluation is also bad.
- But adding redundant information should only be discussed during physical design. There must be really good and quantifiable reasons.
- Avoid storing redundant data whenever you can!

  Many cases of redundant information can be detected by checking for normal forms.

Introduction (7)

- In the example, the information about the two concepts "Course" and "Instructor" are intermixed in one table. This is bad:
  - ◇ The phone number of a new faculty member can be stored in the table only together with a course (Insertion Anomaly).

    Null values also do not help since the course reference number is the key of the table, and the key must be not null.
  - ◇ When the last course of a faculty member is deleted, his/her phone number is lost (Deletion Anomaly).

Introduction (8)

- If one does a good Entity-Relationship design and translates it into the relational model, all normal forms will be automatically satisfied.
- However, normal forms are generally accepted. If one should have to argue about design alternatives in a team, saying that one schema violates a normal form is a strong and formal reason against it.
- Normal forms give another possibility for checking a proposed schema. However, it is much better to detect the problems already on the ER-level.

Introduction (9)

- Today, Third Normal Form (3NF) is considered part of general database education.
- Boyce-Codd Normal Form (BCNF) is slightly stronger, easier to define, and better matches intuition.
- Intuitively, BCNF means that all FDs are already enforced by keys (so one can forget about FDs after the normalization check).
- Only BCNF is defined here.
- If a table is in BCNF, it is automatically in 3NF.

Overview

1. Introduction (Anomalies)
2. Functional Dependencies
3. BCNF

Functional Dependencies (1)

- Functional dependencies (FDs) are generalizations of keys.
- A functional dependency specifies that an attribute (or attribute combination) uniquely determines another attribute (or other attributes).
- Functional dependencies are written in the form

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m.$$

- This means that whenever two rows have the same values in the attributes $A_1, \ldots, A_n$, then they must also agree in the attributes $B_1, \ldots, B_m$.

Functional Dependencies (2)

- As noted above, the FD "INAME $\longrightarrow$ PHONE" is satisfied in the following example:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB	Brass	9404
42232	DS	Brass	9404
31822	IS	Spring	9429

- If two rows agree in the instructor name, they must have the same phone number.

  If two rows do not have the same value for INAME, the condition is void for them.

Functional Dependencies (3)

- A key uniquely determines every attribute, i.e. the FDs "CRN→TITLE", "CRN→INAME", "CRN→PHONE" are trivially satisfied:
  - ◇ There are no two distinct rows that have the same value for a key (CRN in this case).
  - ◇ Therefore, whenever rows t and u agree in the key (CRN), they must actually be the same row, and therefore agree in all other attributes, too.
- Instead of the three FDs above, one can also write the single FD "CRN → TITLE, INAME, PHONE".

Functional Dependencies (4)

- In the example, the FD "INAME $\longrightarrow$ TITLE" is not satisfied: There are two rows with the same INAME, but different values for TITLE.
- In the example, the FD "TITLE $\longrightarrow$ CRN" is satisfied.
- However, like keys, FDs are constraints: They must hold in all possible database states, not only in a single example state.

  Of course, if an FD does not hold in a valid example state, it is clear that it cannot hold in general. E.g. "INAME $\longrightarrow$ TITLE" does not have to be considered any further.

Functional Dependencies (5)

- Therefore, it is a database design task to determine which FDs should hold. This cannot be decided automatically, and the FDs are needed as input for the normalization check.
- In the example, the DB designer must find out whether it can ever happen that two courses are offered with the same title (e.g. two sessions of a course that is overbooked).
- If this can happen, the FD "TITLE $\longrightarrow$ CRN" does not hold in general.

Functional Dependencies (6)

- Sequence and multiplicity of attributes in an FD are unimportant, since both sides are formally sets of attributes: $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$.
- In discussing FDs, the focus is on a single relation R. All attributes A_i, B_i are from this relation.
- The FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ is equivalent to the m FDs:

$$\begin{array}{ccc} A_1, \ldots, A_n & \rightarrow & B_1 \\ \vdots & \vdots & \vdots \\ A_1, \ldots, A_n & \rightarrow & B_m. \end{array}$$

FDs vs. Keys

- FDs are a generalization of keys: $A_1, \ldots, A_n$ is a key of $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$ if and only if the FD "$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$" holds.

  Under the assumption that there are no duplicate rows. Two distinct rows that are identical in every attribute would not violate the FD, but they would violate the key. In theory, this cannot happen, because relations are sets of tuples, and tuples are defined only by their attribute values. In practice, SQL permits two identical rows in a table as long as one did not define a key (therefore, always define a key).

- Given the FDs for a relation, one can compute a key by finding a set of attributes $A_1, \ldots, A_n$ that functionally determines the other attributes.

Trivial FDs

- A functional dependency $\alpha \longrightarrow \beta$ such that $\beta \subseteq \alpha$ is called trivial.
- Examples are:
  - ◇ TITLE $\longrightarrow$ TITLE
  - ◇ INAME, PHONE $\longrightarrow$ PHONE
- Trivial functional dependencies are always satisfied (in every database state, no matter whether it satisfies other constraints or not).

  In logic, they would be called a tautology.
- Trivial FDs are not interesting.

Implication of FDs (1)

- CRN→PHONE is nothing new when one knows already CRN→INAME and INAME→PHONE.

  Whenever $A \rightarrow B$ and $B \rightarrow C$ are satisfied, $A \rightarrow C$ automatically holds.

- A set of FDs $\{\alpha_1 \longrightarrow \beta_1, \ldots, \alpha_n \longrightarrow \beta_n\}$ implies an FD $\alpha \longrightarrow \beta$ if and only if every DB state which satisfies the $\alpha_i \longrightarrow \beta_i$ for $i = 1, \ldots, n$ also satisfies $\alpha \longrightarrow \beta$.

  α and β stand here for sets of attributes/columns. Note that this is simply the definition of logical implication known from Chapter 2. But since FDs are usually not written in the syntax of logical formulas, one formally needs this definition. Of course, FDs could be easily written as logical formulas (Exercise). The notation as a pair of sets of attributes is only an abbreviation for the corresponding formula.

Implication of FDs (2)

- One is normally not interested in all FDs which hold, but only in a representative set that implies all other FDs.
- Implied dependencies can be computed by applying the Armstrong Axioms:
  - ◇ If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ trivially holds (Reflexivity).
  - ◇ If $\alpha \rightarrow \beta$, then $\alpha \cup \gamma \ \rightarrow \ \beta \cup \gamma$ (Augmentation).
  - ◇ If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (Transitivity).

Implication of FDs (3)

- A simpler way to check whether $\alpha \rightarrow \beta$ is implied by given FDs is to compute first the attribute cover α^+ of α and then to check whether $\beta \subseteq \alpha^+$.
- The attribute cover α^+ of a set of attributes α is the set of all attributes B that are uniquely determined by α (with respect to given FDs).

$$\alpha^+ := \{B \mid \text{The given FDs imply } \alpha \rightarrow B\}.$$

  The attribute cover α^+ depends on the given FDs $\mathcal{F}$, although $\mathcal{F}$ is not explicitly shown in the usual notation α^+. If necessary, write $\alpha^+_{\mathcal{F}}$.

- A set of FDs $\mathcal{F}$ implies $\alpha \rightarrow \beta$ if and only if $\beta \subseteq \alpha^+_{\mathcal{F}}$.

Implication of FDs (4)

- The cover is computed as follows:

Input: α (Set of attributes)
$\alpha_1 \longrightarrow \beta_1, \ldots, \alpha_n \longrightarrow \beta_n$ (Set of FDs)

Output: α^+ (Set of attributes, Cover of α)

Method: $x := \alpha$;
while x did change **do**
  for each given FD $\alpha_i \longrightarrow \beta_i$ **do**
    if $\alpha_i \subseteq x$ **then**
      $x := x \cup \beta_i$;
output x;

Implication of FDs (5)

- Consider the following FDs:

$$\texttt{ISBN} \longrightarrow \texttt{TITLE, PUBLISHER}$$
$$\texttt{ISBN, NO} \longrightarrow \texttt{AUTHOR}$$
$$\texttt{PUBLISHER} \longrightarrow \texttt{PUB\_URL}$$

- Suppose we want to compute $\{\texttt{ISBN}\}^+$.

- We start with $x = \{\texttt{ISBN}\}$.

  x is the set of attributes for which we know that there can be only a single value. We start with the assumption that for the given attributes in α, i.e. ISBN, there is only one value. Then the cover α^+ is the set of attributes for which we can derive under this assumption that their value is uniquely determined (using the given FDs).

Implication of FDs (6)

- The first of the given FDs, namely

$$\texttt{ISBN} \longrightarrow \texttt{TITLE, PUBLISHER}$$

  has a left hand side (ISBN) that is contained in the current set x (actually, $x = \{\texttt{ISBN}\}$).

  I.e. there is a unique value for these attributes. Then the FD means that also for the attributes on the right hand side have a unique value.

- Therefore, we can extend x by the attributes on the right hand side of this FD, i.e. TITLE, and PUBLISHER:

$$x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}\}.$$

Implication of FDs (7)

- Now the third of the FDs, namely

  $$\texttt{PUBLISHER} \longrightarrow \texttt{PUB\_URL}$$

  is applicable: Its left hand side is contained in x.

- Therefore, we can add the right hand side of this FD to x and get

  $$x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}.$$

- The last FD, namely

  $$\texttt{ISBN, NO} \longrightarrow \texttt{AUTHOR}$$

  is still not applicable, because NO is missing in x.

Implication of FDs (8)

- After checking again that there is no way to extend the set x any further with the given FDs, the algorithm terminates and prints

$$\{\texttt{ISBN}\}^+ = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}.$$

- From this, we can conclude that the given FDs imply e.g. $\texttt{ISBN} \longrightarrow \texttt{PUB\_URL}$.

- In the same way, one can compute e.g. the cover of $\{\texttt{ISBN}, \texttt{NO}\}$. It is the entire set of attributes.

  This means that $\{\texttt{ISBN}, \texttt{NO}\}$ is a key of the relation, see next slide.

How to Determine Keys (1)

- Given a set of FDs (and the set of all attributes $\mathcal{A}$ of a relation), one can determine all possible keys for that relation.

  Again, one must assume that duplicate rows are excluded.

- $\alpha \subseteq \mathcal{A}$ is a key if and only if $\alpha^+ = \mathcal{A}$.

- Normally, one is only interested in minimal keys.

  The superset of a key is again a key, e.g. if $\{\mathtt{ISBN}, \mathtt{NO}\}$ uniquely identifies all other attributes, this automatically holds also for $\{\mathtt{ISBN}, \mathtt{NO}, \mathtt{TITLE}\}$. Therefore, one usually requires in addition that every $A \in \alpha$ must be necessary, i.e. $(\alpha - \{A\})^+ \neq \mathcal{A}$. Most authors make the minimality requirement part of the key definition. But then a key is not only a constraint, it also says that stronger constraints do not hold.

How to Determine Keys (2)

- One constructs a key x iteratively starting with the empty set ($x = \emptyset$).
- In order to avoid non-minimal keys, one makes sure that the set x never contains the right hand side B of an FD $\alpha \rightarrow B$, when it already contains the left hand side (i.e. $\alpha \subseteq x$).

  We assume here that all FDs have only a single attribute on the right hand side, this is no restriction. Only FDs with $B \notin \alpha$ can destroy the minimality.

How to Determine Keys (3)

- As long as x does not yet determine all attributes (i.e. $x^+ \neq \mathcal{A}$), one chooses any of the remaining attributes $X \in \mathcal{A} - x^+$.

  It is easier if one first chooses attributes that appear on as few as possible right hand sides of FDs.

- Now there are different possibilities, and one must try all:
  - ◇ One can simply add the attribute X to x.
  - ◇ For each FD $\alpha \longrightarrow X$ with X on the right hand side, one can add the left hand side α to x.

How to Determine Keys (4)

- I.e. the search for the key branches. If
  - ◇ one runs into a dead end, i.e. x became non-minimal, or
  - ◇ one has found a key and wants to look for an alternative one,

  one must backtrack to the last branch and try another option.
- In the procedure shown on the next slide, the backtracking is implicitly done by the recursive calls (with "call by value" parameters).

How to Determine Keys (5)

procedure key(x, $\mathcal{A}$, $\mathcal{F}$) /* start with $x = \emptyset$ */
    for each $\alpha \rightarrow B \in \mathcal{F}$ **do**
        if $\alpha \subseteq x$ **and** $B \in x$ **and** $B \notin \alpha$ **then**
            return; /* not minimal */
    if $x^+ = \mathcal{A}$ **then**
        print x; /* minimal key found */
    else
        let X **be any element of** $\mathcal{A} - x^+$;
        key($x \cup \{X\}$, $\mathcal{A}$, $\mathcal{F}$);
        for each $\alpha \rightarrow X \in \mathcal{F}$ **do**
            key($x \cup \alpha$, $\mathcal{A}$, $\mathcal{F}$);

How to Determine Keys (6)

- One can construct a key also in a less formal way.
- So one starts with the set of required attributes (that do not appear on any right side).

  In the example one is already done: ISBN and NO appear at no right side, but their cover is the set of all attributes.

- If the required attributes do not already form a key, one adds attributes: The left hand side of an FD or directly one of the missing attributes.

  Only make sure at the end that the set is minimal. If it contains attributes that are functionally determined by other attributes in the set, remove them.

Exercise

- The following relation is used for storing orders:

  `ORDER(ORD_NO, DATE, CUST_NO, PROD_NO, QUANTITY)`

- Please list FDs which hold for this relation:

  One order can be about multiple products.

- Do these FDs imply the following FD?

  $$\texttt{ORD\_NO, PROD\_NO} \longrightarrow \texttt{DATE}$$

- Determine a key of the relation `ORDER`.

Overview

1. Introduction (Anomalies)

2. Functional Dependencies

3. BCNF

Motivation (1)

- Consider again the example:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB	Brass	9404
42232	DS	Brass	9404
31822	IS	Spring	9429

- As noted above, the FD INAME$\longrightarrow$PHONE leads to problems, one of which is the redundant storage of certain facts (e.g. the phone number of "Brass").

Motivation (2)

- Actually, any FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ will cause redundant storage unless $A_1, \ldots, A_n$ is a key, so that each combination of attribute values for $A_1, \ldots, A_n$ can occur only once.

  Trivial constraints must be excluded here, i.e. at least one of the B_i should not appear among the A_j.

- In general, whenever one stores redundant data, one needs a constraint that ensures that the different copies of the same information remain consistent (i.e. do not contradict each other).

Motivation (3)

- In the cases of redundant data considered here, the constraints are precisely the FDs, e.g. INAME$\longrightarrow$PHONE.
- But FDs are not one of the standard constraints of the relational model. They cannot be specified in the CREATE TABLE statement of current DBMSs.
- Only the special case of keys is supported.
- Thus: Avoid (proper) FDs by transforming them into key constraints. This is what normalization does.

Motivation (4)

- The problem in the example is also caused by the fact that information about different concepts is stored together (faculty members and courses).
- Formally, this follows also from "INAME$\longrightarrow$PHONE":
  - ◇ INAME is like a key for only part of the attributes.
  - ◇ It identifies faculty members, and PHONE depends only on the faculty member, not on the course.
- Again: The left hand side of an FD should be a key.

  It is not a problem if a relation has two keys: Then there are only two ways to identify the same concept.

Boyce-Codd Normal Form

- A Relation R is in BCNF if and only if all its FDs are already implied by key constraints.

  Thus, a relation in BCNF does not require FD constraints, only key constraints.

- I.e. for every FD "$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$" one of the following conditions must hold:
  - ◇ The FD is trivial, i.e. $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$.
  - ◇ The FD follows from a key, because $\{A_1, \ldots, A_n\}$ or some subset of it is already a key.

    It can be any key, not necessarily the primary key.

Examples (1)

- COURSES(CRN, TITLE, INAME, PHONE) with the FDs
  - CRN → TITLE, INAME, PHONE
  - INAME → PHONE

  is not in BCNF because the FD "INAME → PHONE" is not implied by a key:
  - "INAME" is not a key of the entire relation.
  - The FD is not trivial.
- However, without the attribute PHONE (and its FD), the relation is in BCNF:
  - CRN → TITLE, INAME corresponds to the key.

Examples (2)

- Suppose that each course meets only once per week and that there are no cross-listed courses. Then

  `CLASS(CRN, TITLE, DAY, TIME, ROOM)`

  satisfies the following FDs (plus implied ones):

  ◇ CRN $\longrightarrow$ TITLE, DAY, TIME, ROOM

  ◇ DAY, TIME, ROOM $\longrightarrow$ CRN

- The keys are CRN and DAY, TIME, ROOM.
- Both FDs have a key on the left hand side, so the relation is in BCNF.

Examples (3)

- Suppose that PRODUCT(NO, NAME, PRICE) has these FDs:

  (1) NO $\longrightarrow$ NAME
  (2) NO $\longrightarrow$ PRICE
  (3) PRICE, NAME $\longrightarrow$ NAME
  (4) NO, PRICE $\longrightarrow$ NAME

- This relation is in BCNF:
  - The first two FDs show that NO is a key. Since their left hand side is a key, they are no problem.
  - The third FD is trivial and can be ignored.
  - The fourth FD has a superset of the key on the left hand side, which is also no problem.

Exercises

- Is RESULTS(STUD_ID, EX_NO, POINTS, MAX_POINTS) with the following FDs in BCNF?

  (1) STUD_ID, EX_NO $\longrightarrow$ POINTS

  (2) EX_NO $\longrightarrow$ MAX_POINTS

  First determine all minimal keys (there is only one).

- Is the relation

  ORDER(ORD_NO, DATE, CUST_NO, PROD_NO, QUANTITY),

  for which you determined FDs above, in BCNF?

Splitting Relations (1)

- A table which is not in BCNF can be split into two tables ("decomposition"), e.g. split COURSES into

  COURSES_NEW(<u>CRN</u>, TITLE, INAME→INSTRUCTORS)
  INSTRUCTORS(<u>INAME</u>, PHONE)

- General case: If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ violates BCNF, create a relation $S(\underline{A_1}, \ldots, \underline{A_n}, B_1, \ldots, B_m)$ and remove $B_1, \ldots, B_m$ from the original relation.

  $B_1, \ldots, B_m$ should be all attributes that are functionally determined by $A_1, \ldots, A_n$. No B_i should appear among the A_j. $A_1, \ldots, A_n$ become a foreign key in the original relation. In unusual cases (multiple violations), it is necessary to repeat the splitting step with one or both of the resulting relations. Then also implied FDs must be considered.

Splitting Relations (2)

- When splitting relations, it is of course important that the transformation is "lossless", i.e. that the original relation can be reconstructed by means of a join:

$$\mathtt{COURSES} = \mathtt{COURSES\_NEW} \bowtie \mathtt{INSTRUCTORS}.$$

- I.e. the original relation can be defined as a view:

```
CREATE VIEW COURSES(CRN, TITLE, INAME, PHONE)
AS
SELECT C.CRN, C.TITLE, C.INAME, I.PHONE
FROM   COURSES_NEW C, INSTRUCTORS I
WHERE  C.INAME = I.INAME
```

Splitting Relations (3)

- The split of the relations is guaranteed to be lossless if the intersection of the attributes of the new tables is a key of at least one of them ("decomposition theorem"):

$$\{\texttt{CRN}, \texttt{TITLE}, \texttt{INAME}\} \cap \{\texttt{INAME}, \texttt{PHONE}\} = \{\texttt{INAME}\}.$$

- The above method for transforming relations into BCNF does only splits that satisfy this condition.
- It is always possible to transform a relation into BCNF by lossless splitting (if necessary repeated).

Splitting Relations (4)

- Not every lossless split is reasonable:

STUDENTS		
SSN	FIRST_NAME	LAST_NAME
111-22-3333	John	Smith
123-45-6789	Maria	Brown

- Splitting this into STUD_FIRST(SSN,FIRST_NAME) and STUD_LAST(SSN,LAST_NAME) is lossless, but
  - ◇ is not necessary to enforce a normal form and
  - ◇ only requires costly joins in later queries.

Splitting Relations (5)

- Losslessness means that the resulting schema can represent all states which were possible before.

  We can translate states from the old schema into the new schema and back (if the FD was satisfied). The new schema supports all queries which the old schema supported: We can define the old relations as views.

- However, the new schema allows states which do not correspond to a state in the old schema: Now instructors without courses can be stored.
- Thus, the two schemas are not equivalent: The new one is more general.

Splitting Relations (6)

- If instructors without courses are possible in the real world, the decomposition removes a fault in the old schema (insertion and deletion anomaly).
- If they are not,
  - ◇ a new constraint is needed that is not necessarily easier to enforce than the FD, but at least

    None of the two can be specified declaratively in the `CREATE TABLE` statement. Thus, nothing is gained or lost.
  - ◇ the redundancy is avoided (update anomaly).