Database Systems

Session 7 – Main Theme

Functional Dependencies and Normalization

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Presentation material partially based on textbook slides
by Ramez Elmasri and Shamkant Navathe
Slides copyright © 2016
Session Agenda

- Functional Dependencies and Normalization for RDBs
- RDB Design Algorithms and Further Dependencies
- Summary & Conclusion
What is the class about?

- **Course description and syllabus:**
  - [http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001](http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001)
  - [http://cs.nyu.edu/courses/spring16/CSCI-GA.2433-001/](http://cs.nyu.edu/courses/spring16/CSCI-GA.2433-001/)

- **Textbooks:**
    - Ramez Elmasri and Shamkant Navathe
    - Pearson
Icons / Metaphors

- Information
- Common Realization
- Knowledge/Competency Pattern
- Governance
- Alignment
- Solution Approach
1 Informal Design Guidelines for Relational Databases
   » 1.1 Semantics of the Relation Attributes
   » 1.2 Redundant Information in Tuples and Update Anomalies
   » 1.3 Null Values in Tuples
   » 1.4 Spurious Tuples

2 Functional Dependencies (FDs)
   » 2.1 Definition of Functional Dependency
Session Outline (2/3)

3 Normal Forms Based on Primary Keys
- 3.1 Normalization of Relations
- 3.2 Practical Use of Normal Forms
- 3.3 Definitions of Keys and Attributes Participating in Keys
- 3.4 First Normal Form
- 3.5 Second Normal Form
- 3.6 Third Normal Form

4 General Normal Form Definitions for 2NF and 3NF (For Multiple Candidate Keys)

5 BCNF (Boyce-Codd Normal Form)
Session Outline (3/3)

- 6 Multivalued Dependency and Fourth Normal Form
- 7 Join Dependencies and Fifth Normal Form
What is relational database design?
» The grouping of attributes to form "good" relation schemas

Two levels of relation schemas
» The logical "user view" level
» The storage "base relation" level

Design is concerned mainly with base relations

What are the criteria for "good" base relations?
We first discuss informal guidelines for good relational design.

Then we discuss formal concepts of functional dependencies and normal forms:
- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)

Additional types of dependencies, further normal forms, relational design algorithms by synthesis are discussed in the next section.
1.1 Semantics of the Relational Attributes must be clear

- **GUIDELINE 1**: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEES, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.

- **Bottom Line**: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*
Figure 14.1 A simplified COMPANY relational database schema.

**Figure 14.1** A simplified COMPANY relational database schema.
1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies
Example of an Update Anomaly

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.
Example of an Insert Anomaly

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Insert Anomaly:
  - Cannot insert a project unless an employee is assigned to it.

- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.
Example of a Delete Anomaly

- Consider the relation:
  » EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Delete Anomaly:
  » When a project is deleted, it will result in deleting all the employees who work on that project.
  » Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.
Figure 14.3 Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.
Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.
GUIDELINE 2:

» Design a schema that does not suffer from the insertion, deletion and update anomalies.

» If there are any anomalies present, then note them so that applications can be made to take them into account.
GUIDELINE 3:

» Relations should be designed such that their tuples will have as few NULL values as possible

» Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Reasons for nulls:

» Attribute not applicable or invalid

» Attribute value unknown (may exist)

» Value known to exist, but unavailable
1.4 Generation of Spurious Tuples – avoid at any cost (1/2)

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

GUIDELINE 4:
- The relations should be designed to satisfy the lossless join condition.
- No spurious tuples should be generated by doing a natural-join of any relations.
There are two important properties of decompositions:

a) Non-additive or losslessness of the corresponding join
b) Preservation of the functional dependencies.

Note that:

- Property (a) is extremely important and cannot be sacrificed.
- Property (b) is less stringent and may be sacrificed. (See next section).
2. Functional Dependencies

- Functional dependencies (FDs)
  - Are used to specify formal measures of the "goodness" of relational designs
  - And keys are used to define normal forms for relations
  - Are constraints that are derived from the meaning and interrelationships of the data attributes

- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y
2.1 Defining Functional Dependencies

- X → Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
  
  » For any two tuples t1 and t2 in any relation instance r(R): If \( t1[X]=t2[X] \), then \( t1[Y]=t2[Y] \)

- X → Y in R specifies a *constraint* on all relation instances r(R)

- Written as X → Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).

- FDs are derived from the real-world constraints on the attributes
Examples of FD constraints (1/2)

- Social security number determines employee name
  \[ \text{SSN} \rightarrow \text{ENAME} \]

- Project number determines project name and location
  \[ \text{PNUMBER} \rightarrow \{\text{PNAME, PLOCATION}\} \]

- Employee ssn and project number determines the hours per week that the employee works on the project
  \[ \{\text{SSN, PNUMBER}\} \rightarrow \text{HOURS} \]
Examples of FD constraints (2/2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on every relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])
Defining FDs from instances

- Note that in order to define the FDs, we need to understand the meaning of the attributes involved and the relationship between them.
- An FD is a property of the attributes in the schema R.
- Given the instance (population) of a relation, all we can conclude is that an FD may exist between certain attributes.
- What we can definitely conclude is – that certain FDs do not exist because there are tuples that show a violation of those dependencies.
Note that given the state of the TEACH relation, we can say that the FD: Text → Course may exist. However, the FDs Teacher → Course, Teacher → Text and Course → Text are ruled out.

<table>
<thead>
<tr>
<th>Teacher</th>
<th>Course</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Data Structures</td>
<td>Bartram</td>
</tr>
<tr>
<td>Smith</td>
<td>Data Management</td>
<td>Martin</td>
</tr>
<tr>
<td>Hall</td>
<td>Compilers</td>
<td>Hoffman</td>
</tr>
<tr>
<td>Brown</td>
<td>Data Structures</td>
<td>Horowitz</td>
</tr>
</tbody>
</table>
A relation $R(A, B, C, D)$ with its extension.

Which FDs may exist in this relation?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d3</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c4</td>
<td>d3</td>
</tr>
</tbody>
</table>
3 Normal Forms Based on Primary Keys

- 3.1 Normalization of Relations
- 3.2 Practical Use of Normal Forms
- 3.3 Definitions of Keys and Attributes Participating in Keys
- 3.4 First Normal Form
- 3.5 Second Normal Form
- 3.6 Third Normal Form
Normalization:

- The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

Normal form:

- Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
Normalization of Relations (2/2)

- **2NF, 3NF, BCNF**
  - based on keys and FDs of a relation schema

- **4NF**
  - based on keys, multi-valued dependencies: MVDs;

- **5NF**
  - based on keys, join dependencies: JDs

- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; see next section)
3.2 Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties.

- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*.

- The database designers *need not* normalize to the highest possible normal form.
  - (usually up to 3NF and BCNF. 4NF rarely used in practice.)

- **Denormalization:**
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form.
3.3 Definitions of Keys and Attributes - Participating in Keys (1/2)

- **A superkey** of a relation schema $R = \{A_1, A_2, \ldots, A_n\}$ is a set of attributes $S$ *subset* of $R$ with the property that no two tuples $t_1$ and $t_2$ in any legal relation state $r$ of $R$ will have $t_1[S] = t_2[S]$.

- **A key** $K$ is a superkey with the *additional property* that removal of any attribute from $K$ will cause $K$ not to be a superkey any more.
If a relation schema has more than one key, each is called a **candidate** key. 

- One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.

- A **Prime attribute** must be a member of **some** candidate key.

- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.
3.4 First Normal Form

- Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**: attributes whose values for an *individual tuple* are non-atomic
- Considered to be part of the definition of a relation
- Most RDBMSs allow only those relations to be defined that are in First Normal Form
Figure 14.9 Normalization into 1NF

(a) DEPARTMENT

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Dmgr_ssn</th>
<th>Dlocations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) DEPARTMENT

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Dmgr_ssn</th>
<th>Dlocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>(Bellaire, Sugarland, Houston)</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
<td>(Stafford)</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
<td>(Houston)</td>
</tr>
</tbody>
</table>

(c) DEPARTMENT

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Dmgr_ssn</th>
<th>Dlocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>Bellaire</td>
</tr>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>Sugarland</td>
</tr>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>Houston</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
<td>Stafford</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
<td>Houston</td>
</tr>
</tbody>
</table>
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a nested relation attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.
3.5 Second Normal Form (1/2)

- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Full functional dependency:** A FD $Y \rightarrow Z$ where removal of any attribute from $Y$ means the FD does not hold any more
- Examples:
  - $\{\text{SSN, PNUMBER}\} \rightarrow \text{HOURS}$ is a full FD since neither $\text{SSN} \rightarrow \text{HOURS}$ nor $\text{PNUMBER} \rightarrow \text{HOURS}$ hold
  - $\{\text{SSN, PNUMBER}\} \rightarrow \text{ENAME}$ is not a full FD (it is called a partial dependency) since $\text{SSN} \rightarrow \text{ENAME}$ also holds
A relation schema \( R \) is in **second normal form (2NF)** if every non-prime attribute \( A \) in \( R \) is fully functionally dependent on the primary key.

\( R \) can be decomposed into 2NF relations via the process of 2NF normalization or “second normalization”
Figure 14.11 Normalizing into 2NF and 3NF

(a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.
Figure 14.12 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.
3.6 Third Normal Form (1/2)

- **Definition:**
  - **Transitive functional dependency:** a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$

- **Examples:**
  - SSN $\rightarrow$ DMGRSSN is a **transitive** FD
    - Since SSN $\rightarrow$ DNUMBER and DNUMBER $\rightarrow$ DMGRSSN hold
  - SSN $\rightarrow$ ENAME is **non-transitive**
    - Since there is no set of attributes $X$ where SSN $\rightarrow$ $X$ and $X \rightarrow$ ENAME
A relation schema $R$ is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute $A$ in $R$ is transitively dependent on the primary key.

$R$ can be decomposed into 3NF relations via the process of 3NF normalization.

**NOTE:**

- In $X \rightarrow Y$ and $Y \rightarrow Z$, with $X$ as the primary key, we consider this a problem only if $Y$ is not a candidate key.
- When $Y$ is a candidate key, there is no problem with the transitive dependency.
- E.g., Consider EMP (SSN, Emp#, Salary).
  - Here, $SSN \rightarrow Emp# \rightarrow Salary$ and $Emp#$ is a candidate key.
Normal Forms Defined Informally

- 1\textsuperscript{st} normal form
  - All attributes depend on the key
- 2\textsuperscript{nd} normal form
  - All attributes depend on the whole key
- 3\textsuperscript{rd} normal form
  - All attributes depend on nothing but the key
The above definitions consider the primary key only.

The following more general definitions take into account relations with multiple candidate keys.

Any attribute involved in a candidate key is a **prime attribute**.

All other attributes are called **non-prime attributes**.
A relation schema \( R \) is in **second normal form (2NF)** if every non-prime attribute \( A \) in \( R \) is fully functionally dependent on every key of \( R \).

In Figure 14.12 the FD \( \text{County\_name} \rightarrow \text{Tax\_rate} \) violates 2NF.

So second normalization converts LOTS into:

LOTS1 (\( \text{Property\_id}\#, \text{County\_name}, \text{Lot}\#, \text{Area}, \text{Price} \))
LOTS2 (\( \text{County\_name}, \text{Tax\_rate} \))
4.2 General Definition of Third Normal Form

- Definition:
  » **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
  » A relation schema R is in **third normal form (3NF)** if whenever a FD X → A holds in R, then either:
    • (a) X is a superkey of R, or
    • (b) A is a prime attribute of R

- LOTS1 relation violates 3NF because Area → Price; and Area is not a superkey in LOTS1. (see Figure 14.12).
Consider the 2 conditions in the Definition of 3NF:

A relation schema $R$ is in **third normal form (3NF)** if whenever a FD $X \rightarrow A$ holds in $R$, then either:

- (a) $X$ is a superkey of $R$, or
- (b) $A$ is a prime attribute of $R$

Condition (a) catches two types of violations:

- one where a prime attribute functionally determines a non-prime attribute. This catches 2NF violations due to non-full functional dependencies.

- second, where a non-prime attribute functionally determines a non-prime attribute. This catches 3NF violations due to a transitive dependency.
ALTERNATIVE DEFINITION of 3NF: We can restate the definition as:
A relation schema R is in **third normal form (3NF)** if every non-prime attribute in R meets both of these conditions:
» It is fully functionally dependent on every key of R
» It is non-transitively dependent on every key of R
Note that stated this way, a relation in 3NF also meets the requirements for 2NF.

The condition (b) from the last slide takes care of the dependencies that “slip through” (are allowable to) 3NF but are “caught by” BCNF which we discuss next.
A relation schema \( R \) is in **Boyce-Codd Normal Form (BCNF)** if whenever an FD \( X \rightarrow A \) holds in \( R \), then \( X \) is a superkey of \( R \).

Each normal form is strictly stronger than the previous one:
- Every 2NF relation is in 1NF
- Every 3NF relation is in 2NF
- Every BCNF relation is in 3NF

There exist relations that are in 3NF but not in BCNF.

Hence BCNF is considered a **stronger form of 3NF**.

The goal is to have each relation in BCNF (or 3NF).
Figure 14.13 Boyce-Codd normal form

(a) LOTS1A

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>County_name</th>
<th>Lot#</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BCNF Normalization

LOTS1AX

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>Area</th>
<th>Lot#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>County_name</td>
</tr>
</tbody>
</table>

LOTS1AY

(b) R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.
A relation TEACH that is in 3NF but not in BCNF.

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narayan</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Smith</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Smith</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
<tr>
<td>Smith</td>
<td>Theory</td>
<td>Schulman</td>
</tr>
<tr>
<td>Wallace</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Wallace</td>
<td>Operating Systems</td>
<td>Ahamad</td>
</tr>
<tr>
<td>Wong</td>
<td>Database</td>
<td>Omiecinski</td>
</tr>
<tr>
<td>Zelaya</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Narayan</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
</tbody>
</table>
Achieving the BCNF by Decomposition (1/2)

- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course

- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 14.13 (b).
  - So this relation is in 3NF but not in BCNF

- A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.
  - (See Algorithm 15.3)
Achieving the BCNF by Decomposition (2/2)

- Three possible decompositions for relation TEACH
  - D1: \{student, instructor\} and \{student, course\}
  - D2: \{course, instructor\} and \{course, student\}
  - D3: \{instructor, course\} and \{instructor, student\}

- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.

- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).

- A test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless) is discussed under Property NJB on the next slide. We then show how the third decomposition above meets the property.
Testing Binary Decompositions for Lossless Join (Non-additive Join) Property

» Binary Decomposition: Decomposition of a relation R into two relations.

» PROPERTY NJB (non-additive join test for binary decompositions): A decomposition D = {R1, R2} of R has the lossless join property with respect to a set of functional dependencies F on R if and only if either
  • The f.d. ((R1 \cap R2) \rightarrow (R1 - R2)) is in F^+, or
  • The f.d. ((R1 \cap R2) \rightarrow (R2 - R1)) is in F^+.
If you apply the NJB test to the 3 decompositions of the TEACH relation:

- D1 gives **Student** → Instructor or **Student** → Course, none of which is true.
- D2 gives **Course** → Instructor or **Course** → Student, none of which is true.
- However, in D3 we get **Instructor** → Course or **Instructor** → Student.

Since **Instructor** → Course is indeed true, the NJB property is satisfied and D3 is determined as a non-additive (good) decomposition.
Here we make use the algorithm from next section (Algorithm 15.5):

- Let R be the relation not in BCNF, let X be a subset-of R, and let $X \rightarrow A$ be the FD that causes a violation of BCNF. Then R may be decomposed into two relations:
  
  - (i) $R - A$ and (ii) $X \cup A$.
  
  - If either $R - A$ or $X \cup A$ is not in BCNF, repeat the process.

Note that the f.d. that violated BCNF in TEACH was Instructor $\rightarrow$ Course. Hence its BCNF decomposition would be:

(TEACH – COURSE) and (Instructor $\cup$ Course), which gives the relations: (Instructor, Student) and (Instructor, Course) that we obtained before in decomposition D3.
Definition:

- A multivalued dependency (MVD) \( X \rightarrow Y \) specified on relation schema \( R \), where \( X \) and \( Y \) are both subsets of \( R \), specifies the following constraint on any relation state \( r \) of \( R \): If two tuples \( t_1 \) and \( t_2 \) exist in \( r \) such that \( t_1[X] = t_2[X] \), then two tuples \( t_3 \) and \( t_4 \) should also exist in \( r \) with the following properties, where we use \( Z \) to denote \( (R_2 (X \cup Y)) \):
  \[
  \begin{align*}
  & t_3[X] = t_4[X] = t_1[X] = t_2[X]. \\
  & t_3[Y] = t_1[Y] \text{ and } t_4[Y] = t_2[Y]. \\
  & t_3[Z] = t_2[Z] \text{ and } t_4[Z] = t_1[Z].
  \end{align*}
  \]

- An MVD \( X \rightarrow Y \) in \( R \) is called a trivial MVD if (a) \( Y \) is a subset of \( X \), or (b) \( X \cup Y = R \).
Definition:

- A relation schema $R$ is in **4NF** with respect to a set of dependencies $F$ (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \rightarrow Y$ in $F^+$, $X$ is a superkey for $R$.
  
  » Note: $F^+$ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state $r$ of $R$ that satisfies $F$. It is also called the **closure** of $F$. 

Fourth and fifth normal forms. (a) The EMP relation with two MVDs: Ename $\rightarrow\rightarrow$ Pname and Ename $\rightarrow\rightarrow$ Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3). (d) Decomposing the relation SUPPLY into the 5NF relations R1, R2, R3.
Definition:

- A join dependency (JD), denoted by $\text{JD}(R_1, R_2, \ldots, R_n)$, specified on relation schema $R$, specifies a constraint on the states $r$ of $R$.
  - The constraint states that every legal state $r$ of $R$ should have a non-additive join decomposition into $R_1, R_2, \ldots, R_n$; that is, for every such $r$ we have
  - $^\ast (\pi_{R_1}(r), \pi_{R_2}(r), \ldots, \pi_{R_n}(r)) = r$

**Note:** an MVD is a special case of a JD where $n = 2$.

- A join dependency $\text{JD}(R_1, R_2, \ldots, R_n)$, specified on relation schema $R$, is a trivial JD if one of the relation schemas $R_i$ in $\text{JD}(R_1, R_2, \ldots, R_n)$ is equal to $R$. 
Definition:

- A relation schema \( R \) is in **fifth normal form (5NF)** (or **Project-Join Normal Form (PJNF)**) with respect to a set \( F \) of functional, multivalued, and join dependencies if,
  - for every nontrivial join dependency \( JD(R_1, R_2, \ldots, R_n) \) in \( F^+ \) (that is, implied by \( F \)),
    - every \( R_i \) is a superkey of \( R \).

Discovering join dependencies in practical databases with hundreds of relations is next to impossible. Therefore, 5NF is rarely used in practice.
Session Summary

- Informal Design Guidelines for Relational Databases
- Functional Dependencies (FDs)
- Normal Forms (1NF, 2NF, 3NF) Based on Primary Keys
- General Normal Form Definitions of 2NF and 3NF (For Multiple Keys)
- BCNF (Boyce-Codd Normal Form)
- Fourth and Fifth Normal Forms
1. Further topics in Functional Dependencies
   » 1.1 Inference Rules for FDs
   » 1.2 Equivalence of Sets of FDs
   » 1.3 Minimal Sets of FDs

2. Properties of Relational Decompositions

3. Algorithms for Relational Database Schema Design

4. Nulls, Dangling Tuples, Alternative Relational Designs
5. Multivalued Dependencies and Fourth Normal Form – further discussion

6. Other Dependencies and Normal Forms
   » 6.1 Join Dependencies
   » 6.2 Inclusion Dependencies
   » 6.3 Dependencies based on Arithmetic Functions and Procedures
   » 6.2 Domain-Key Normal Form
We discussed functional dependencies in the last chapter.

To recollect:
A set of attributes X \textit{functionally determines} a set of attributes Y if the value of X determines a unique value for Y.

Our goal here is to determine the properties of functional dependencies and to find out the ways of manipulating them.
Defining Functional Dependencies

- X → Y holds if whenever two tuples have the same value for X, they must have the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], then t1[Y]=t2[Y]
- X → Y in R specifies a constraint on all relation instances r(R)
- Written as X → Y; can be displayed graphically on a relation schema as in Figures in the previous section. (denoted by the arrow: ).
- FDs are derived from the real-world constraints on the attributes
Definition: An FD $X \rightarrow Y$ is inferred from or implied by a set of dependencies $F$ specified on $R$ if $X \rightarrow Y$ holds in every legal relation state $r$ of $R$; that is, whenever $r$ satisfies all the dependencies in $F$, $X \rightarrow Y$ also holds in $r$.

Given a set of FDs $F$, we can infer additional FDs that hold whenever the FDs in $F$ hold.
Armstrong's inference rules:

- **IR1. (Reflexive)** If $Y \subset X$, then $X \rightarrow Y$
- **IR2. (Augmentation)** If $X \rightarrow Y$, then $XZ \rightarrow YZ$
  - (Notation: $XZ$ stands for $X \cup Z$)
- **IR3. (Transitive)** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

IR1, IR2, IR3 form a **sound and complete** set of inference rules

- These are rules hold and all other rules that hold can be deduced from these
Some additional inference rules that are useful:

- **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Pseudotransitivity:** If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)
Closure of a set F of FDs is the set $F^+$ of all FDs that can be inferred from F.

Closure of a set of attributes $X$ with respect to F is the set $X^+$ of all attributes that are functionally determined by X.

$X^+$ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F.
Algorithm 15.1. Determining $X^+$, the Closure of $X$ under $F$

**Input:** A set $F$ of FDs on a relation schema $R$, and a set of attributes $X$, which is a subset of $R$.

- $X^+ := X$
- repeat
  - old$X^+ := X^+$;
  - for each functional dependency $Y \rightarrow Z$ in $F$ do
    - if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;
  - until ($X^+ = \text{old}X^+$);
Example of Closure (1/2)

- For example, consider the following relation schema about classes held at a university in a given academic year.

CLASS (Classid, Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity).

- Let $F$, the set of functional dependencies for the above relation include the following f.d.s:

  FD1: Sectionid $\rightarrow$ Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity;
  FD2: Course# $\rightarrow$ Credit_hrs;
  FD3: {Course#, Instr_name} $\rightarrow$ Text, Classroom;
  FD4: Text $\rightarrow$ Publisher
  FD5: Classroom $\rightarrow$ Capacity

These f.d.s above represent the meaning of the individual attributes and the relationship among them and defines certain rules about the classes.
The closures of attributes or sets of attributes for some example sets:

\[
\{ \text{Classid} \}^+ = \{ \text{Classid, Course#, Instr\_name, Credit\_hrs, Text, Publisher, Classroom, Capacity} \} = \text{CLASS}
\]

\[
\{ \text{Course#} \}^+ = \{ \text{Course#, Credit\_hrs} \}
\]

\[
\{ \text{Course#, Instr\_name} \}^+ = \{ \text{Course#, Credit\_hrs, Text, Publisher, Classroom, Capacity} \}
\]

Note that each closure above has an interpretation that is revealing about the attribute(s) on the left-hand-side. The closure of \( \{ \text{Classid} \}^+ \) is the entire relation \text{CLASS} indicating that all attributes of the relation can be determined from Classid and hence it is a key.
Two sets of FDs $F$ and $G$ are equivalent if:

- Every FD in $F$ can be inferred from $G$, and
- Every FD in $G$ can be inferred from $F$
- Hence, $F$ and $G$ are equivalent if $F^+ = G^+$

Definition (Covers):

- $F$ covers $G$ if every FD in $G$ can be inferred from $F$
  - (i.e., if $G^+ \subset F^+$)
- $F$ and $G$ are equivalent if $F$ covers $G$ and $G$ covers $F$
- There is an algorithm for checking equivalence of sets of FDs
1.3 Finding Minimal Cover of F.D.s (1/5)

- Just as we applied inference rules to expand on a set $F$ of FDs to arrive at $F^+$, its closure, it is possible to think in the opposite direction to see if we could shrink or reduce the set $F$ to its minimal form so that the minimal set is still equivalent to the original set $F$.

- **Definition**: An attribute in a functional dependency is considered **extraneous attribute** if we can remove it without changing the closure of the set of dependencies. Formally, given $F$, the set of functional dependencies and a functional dependency $X \rightarrow A$ in $F$, attribute $Y$ is extraneous in $X$ if $Y$ is a subset of $X$, and $F$ logically implies $(F - (X \rightarrow A)) \cup \{ (X - Y) \rightarrow A \}$.
A set of FDs is \textbf{minimal} if it satisfies the following conditions:

1. Every dependency in F has a single attribute for its RHS.

2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

3. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where $Y$ is a proper-subset-of $X$ and still have a set of dependencies that is equivalent to F.
Algorithm 15.2. Finding a Minimal Cover $F$ for a Set of Functional Dependencies $E$

» Input: A set of functional dependencies $E$.

2. Replace each functional dependency $X \rightarrow \{A_1, A_2, ..., A_n\}$ in $F$ by the $n$ functional dependencies $X \rightarrow A_1, X \rightarrow A_2, ..., X \rightarrow A_n$.
3. For each functional dependency $X \rightarrow A$ in $F$
   for each attribute $B$ that is an element of $X$
   if $\{ (F - \{X \rightarrow A\} ) \cup \{ (X - \{B\} ) \rightarrow A\} \}$ is equivalent to $F$
   then replace $X \rightarrow A$ with $\{(X - \{B\}) \rightarrow A\}$ in $F$.

   (* The above constitutes a removal of the extraneous attribute $B$ from $X$ *)

4. For each remaining functional dependency $X \rightarrow A$ in $F$ if $\{F - \{X \rightarrow A\}\}$ is equivalent to $F$, then remove $X \rightarrow A$ from $F$.

   (* The above constitutes a removal of the redundant dependency $X \rightarrow A$ from $F$ *)
We illustrate algorithm 15.2 with the following:
Let the given set of FDs be \( E : \{ B \rightarrow A, D \rightarrow A, AB \rightarrow D \} \). We have to find the minimum cover of \( E \).

- All above dependencies are in canonical form; so we have completed step 1 of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine if \( AB \rightarrow D \) has any redundant attribute on the left-hand side; that is, can it be replaced by \( B \rightarrow D \) or \( A \rightarrow D \)?
- Since \( B \rightarrow A \), by augmenting with \( B \) on both sides (IR2), we have \( BB \rightarrow AB \), or \( B \rightarrow AB \) (i). However, \( AB \rightarrow D \) as given (ii).
- Hence by the transitive rule (IR3), we get from (i) and (ii), \( B \rightarrow D \). Hence \( AB \rightarrow D \) may be replaced by \( B \rightarrow D \).
- We now have a set equivalent to original \( E \), say \( E' : \{ B \rightarrow A, D \rightarrow A, B \rightarrow D \} \). No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3 we look for a redundant FD in \( E' \). By using the transitive rule on \( B \rightarrow D \) and \( D \rightarrow A \), we derive \( B \rightarrow A \). Hence \( B \rightarrow A \) is redundant in \( E' \) and can be eliminated.
- Hence the minimum cover of \( E \) is \( \{ B \rightarrow D, D \rightarrow A \} \).
Every set of FDs has an equivalent minimal set

There can be several equivalent minimal sets

There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set $F$ of FDs. The process of Algorithm 15.2 is used until no further reduction is possible.

To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set

» E.g., see algorithm 15.4
The Approach of Relational Synthesis (Bottom-up Design):

- Assumes that all possible functional dependencies are known.
- First constructs a minimal set of FDs.
- Then applies algorithms that construct a target set of 3NF or BCNF relations.
- Additional criteria may be needed to ensure the set of relations in a relational database are satisfactory (see Algorithm 15.3).
Goals:

» Lossless join property (a must)
  • Algorithm 15.3 tests for general losslessness.

» Dependency preservation property
  • Observe as much as possible
  • Algorithm 15.5 decomposes a relation into BCNF components by sacrificing the dependency preservation.

» Additional normal forms
  • 4NF (based on multi-valued dependencies)
  • 5NF (based on join dependencies)
Algorithm 15.2a Finding a Key K for R, given a set F of Functional Dependencies

» Input: A universal relation R and a set of functional dependencies F on the attributes of R.

1. Set K := R;
2. For each attribute A in K {
   Compute (K - A)+ with respect to F;
   If (K - A)+ contains all the attributes in R,
   then set K := K - {A};
}
Relation Decomposition and Insufficiency of Normal Forms:

» Universal Relation Schema:
  • A relation schema $R = \{A_1, A_2, \ldots, A_n\}$ that includes all the attributes of the database.

» Universal relation assumption:
  • Every attribute name is unique.
2.1 Relation Decomposition and Insufficiency of Normal Forms (cont.):

» Decomposition:
  • The process of decomposing the universal relation schema $R$ into a set of relation schemas $D = \{R_1, R_2, \ldots, R_m\}$ that will become the relational database schema by using the functional dependencies.

» Attribute preservation condition:
  • Each attribute in $R$ will appear in at least one relation schema $R_i$ in the decomposition so that no attributes are “lost”.
Another goal of decomposition is to have each individual relation \( R_i \) in the decomposition \( D \) be in BCNF or 3NF.

Additional properties of decomposition are needed to prevent from generating spurious tuples.
2.2 Dependency Preservation Property of a Decomposition:

- **Definition**: Given a set of dependencies $F$ on $R$, the projection of $F$ on $R_i$, denoted by $p_{R_i}(F)$ where $R_i$ is a subset of $R$, is the set of dependencies $X \rightarrow^+ Y$ in $F^+$ such that the attributes in $X \cup Y$ are all contained in $R_i$.

- Hence, the projection of $F$ on each relation schema $R_i$ in the decomposition $D$ is the set of functional dependencies in $F^+$, the closure of $F$, such that all their left- and right-hand-side attributes are in $R_i$. 
 Dependency Preservation Property of a Decomposition (cont.):

 › Dependency Preservation Property:

   » A decomposition \( D = \{R_1, R_2, ..., R_m\} \) of \( R \) is dependency-preservation with respect to \( F \) if the union of the projections of \( F \) on each \( R_i \) in \( D \) is equivalent to \( F \); that is

   \[
   ((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+
   \]

   » (See examples in Fig 14.13a and Fig 14.12)

 Claim 1:

 › It is always possible to find a dependency-preserving decomposition \( D \) with respect to \( F \) such that each relation \( R_i \) in \( D \) is in 3nf.
2.3 Non-additive (Lossless) Join Property of a Decomposition:

» Definition: Lossless join property: a decomposition $D = \{R_1, R_2, ..., R_m\}$ of $R$ has the **lossless (nonadditive) join property** with respect to the set of dependencies $F$ on $R$ if, for every relation state $r$ of $R$ that satisfies $F$, the following holds, where $\ast$ is the natural join of all the relations in $D$:

$$\ast (\pi_{R_1}(r), ..., \pi_{R_m}(r)) = r$$

» Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for “loss of information” a better term is “addition of spurious information”
Properties of Relational Decompositions (7/12)

Lossless (Non-additive) Join Property of a Decomposition:

- **Algorithm 15.3: Testing for Lossless Join Property**
  - **Input**: A universal relation $R$, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of $R$, and a set $F$ of functional dependencies.
  1. Create an initial matrix $S$ with one row $i$ for each relation schema $R_i$ in $D$, and one column $j$ for each attribute $A_j$ in $R$.
  2. Set $S(i,j):=b_{ij}$ for all matrix entries. (* each $b_{ij}$ is a distinct symbol associated with indices $(i,j)$ *).
  3. For each row $i$ representing relation schema $R_i$
     {for each column $j$ representing attribute $A_j$
      {if (relation $R_i$ includes attribute $A_j$) then set $S(i,j):= a_j$};};
  » (* each $a_j$ is a distinct symbol associated with index $(j)$ *)
  » CONTINUED on NEXT SLIDE
Lossless (Non-additive) Join Property of a Decomposition (cont.):

Algorithm 15.3: Testing for Lossless Join Property (continued)

4. Repeat the following loop until a complete loop execution results in no changes to S
   {for each functional dependency X → Y in F
      {for all rows in S which have the same symbols in the columns corresponding to attributes in X
         {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:
            If any of the rows has an “a” symbol for the column, set the other rows to that same “a” symbol in the column.
            If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appear in one of the rows for the attribute and set the other rows to that same “b” symbol in the column ;};
         };}
      }
   }
5. If a row is made up entirely of “a” symbols, then the decomposition has the lossless join property; otherwise it does not.
Properties of Relational Decompositions (9/12)

Figure 15.1 Nonadditive join test for n-ary decompositions.
(a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test.
(b) A decomposition of EMP_PROJ that has the lossless join property.

(a) \[ R = \{ \text{Ssn, Ename, Pnumber, Pname, Plocation, Hours} \} \]
\[ R_1 = \text{EMP_LOCS} = \{ \text{Ename, Plocation} \} \]
\[ R_2 = \text{EMP_PROJ1} = \{ \text{Ssn, Pnumber, Hours, Pname, Plocation} \} \]

\[ D = \{ R_1, R_2 \} \]

\[ F = \{ \text{Ssn }\rightarrow\text{Ename; Pnumber }\rightarrow\{\text{Pname, Plocation}\}; \{\text{Ssn, Pnumber}\} \rightarrow\text{Hours} \} \]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Ssn} & \text{Ename} & \text{Pnumber} & \text{Pname} & \text{Plocation} & \text{Hours} \\
\hline
b_{11} & a_2 & b_{13} & b_{14} & a_5 & b_{16} \\
\hline
a_1 & b_{22} & a_3 & a_4 & a_5 & a_6 \\
\hline
\end{array}
\]

(No changes to matrix after applying functional dependencies)

(b) EMP
\[
\begin{array}{|c|c|}
\hline
\text{Ssn} & \text{Ename} \\
\hline
\end{array}
\]

PROJECT
\[
\begin{array}{|c|c|c|}
\hline
\text{Pnumber} & \text{Pname} & \text{Plocation} \\
\hline
\end{array}
\]

WORKS_ON
\[
\begin{array}{|c|c|c|}
\hline
\text{Ssn} & \text{Pnumber} & \text{Hours} \\
\hline
\end{array}
\]
Properties of Relational Decompositions (10/12)

(c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(c) Nonadditive join test for n-ary decompositions. *(Figure 15.1)*

\[ R = \{Ssn, Ename, Pnumber, Pname, Plocation, Hours\} \]
\[ R_1 = EMP = \{Ssn, Ename\} \]
\[ R_2 = PROJ = \{Pnumber, Pname, Plocation\} \]
\[ R_3 = WORKS_ON = \{Ssn, Pnumber, Hours\} \]

\[ D = \{R_1, R_2, R_3\} \]

\[ F = \{Ssn \rightarrow Ename; Pnumber \rightarrow \{Pname, Plocation\}; \{Ssn, Pnumber\} \rightarrow \text{Hours}\} \]

<table>
<thead>
<tr>
<th></th>
<th>Ssn</th>
<th>Ename</th>
<th>Pnumber</th>
<th>Pname</th>
<th>Plocation</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>(a₁)</td>
<td>(a₂)</td>
<td>(b_{13})</td>
<td>(b_{14})</td>
<td>(b_{15})</td>
<td>(b_{16})</td>
</tr>
<tr>
<td>R₂</td>
<td>(b_{21})</td>
<td>(b_{22})</td>
<td>(a₃)</td>
<td>(a₄)</td>
<td>(a₅)</td>
<td>(b_{26})</td>
</tr>
<tr>
<td>R₃</td>
<td>(a₁)</td>
<td>(b_{32})</td>
<td>(a₃)</td>
<td>(b_{34})</td>
<td>(b_{35})</td>
<td>(a₆)</td>
</tr>
</tbody>
</table>

(Original matrix S at start of algorithm)

<table>
<thead>
<tr>
<th></th>
<th>Ssn</th>
<th>Ename</th>
<th>Pnumber</th>
<th>Pname</th>
<th>Plocation</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>(a₁)</td>
<td>(a₂)</td>
<td>(b_{13})</td>
<td>(b_{14})</td>
<td>(b_{15})</td>
<td>(b_{16})</td>
</tr>
<tr>
<td>R₂</td>
<td>(b_{21})</td>
<td>(b_{22})</td>
<td>(a₃)</td>
<td>(a₄)</td>
<td>(a₅)</td>
<td>(b_{26})</td>
</tr>
<tr>
<td>R₃</td>
<td>(a₁)</td>
<td>(b_{32}a₂)</td>
<td>(a₃)</td>
<td>(b_{34}a₄)</td>
<td>(b_{36}a₅)</td>
<td>(a₆)</td>
</tr>
</tbody>
</table>

(Matrix S after applying the first two functional dependencies; last row is all “a” symbols so we stop)
2.4 Testing Binary Decompositions for Non-additive Join (Lossless Join) Property

» **Binary Decomposition:** Decomposition of a relation R into two relations.

» **PROPERTY NJB (non-additive join test for binary decompositions):** A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property with respect to a set of functional dependencies $F$ on R if and only if either
  - The f.d. $((R_1 \cap R_2) \rightarrow (R_1- R_2))$ is in $F^+$, or
  - The f.d. $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in $F^+$. 
2.5 Successive Non-additive Join Decomposition:

» Claim 2 (Preservation of non-additivity in successive decompositions):

• If a decomposition \( D = \{R_1, R_2, \ldots, R_m\} \) of \( R \) has the lossless (non-additive) join property with respect to a set of functional dependencies \( F \) on \( R \),
• and if a decomposition \( D_i = \{Q_1, Q_2, \ldots, Q_k\} \) of \( R_i \) has the lossless (non-additive) join property with respect to the projection of \( F \) on \( R_i \),
  – then the decomposition \( D_2 = \{R_1, R_2, \ldots, R_{i-1}, Q_1, Q_2, \ldots, Q_k, R_{i+1}, \ldots, R_m\} \) of \( R \) has the non-additive join property with respect to \( F \).
3. Algorithms for Relational Database Schema Design (1/2)

- Design of 3NF Schemas:
  
  Algorithm 15.4 Relational Synthesis into 3NF with Dependency Preservation and Non-Additive (Lossless) Join Property

  » Input: A universal relation R and a set of functional dependencies F on the attributes of R.

  1. Find a minimal cover G for F (use Algorithm 15.0).

  2. For each left-hand-side X of a functional dependency that appears in G,

     create a relation schema in D with attributes \( \{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\} \),

     where \( X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k \) are the only dependencies in G with X as left-hand-side (X is the key of this relation).

  3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R. (*Use Algorithm 15.4a to find the key of R*)
Design of BCNF Schemas

Algorithm 15.5: Relational Decomposition into BCNF with Lossless (non-additive) join property

- Input: A universal relation $R$ and a set of functional dependencies $F$ on the attributes of $R$.

1. Set $D := \{R\}$;

2. While there is a relation schema $Q$ in $D$ that is not in BCNF do {
   
   choose a relation schema $Q$ in $D$ that is not in BCNF;
   
   find a functional dependency $X \rightarrow Y$ in $Q$ that violates BCNF;
   
   replace $Q$ in $D$ by two relation schemas $(Q - Y)$ and $(X \cup Y)$;

};

Assumption: No null values are allowed for the join attributes.
4. Problems with Null Values and Dangling Tuples (1/5)

4.1 Problems with NULL values

- when some tuples have NULL values for attributes that will be used to join individual relations in the decomposition that may lead to incomplete results.

- E.g., see Figure 15.2(a), where two relations EMPLOYEE and DEPARTMENT are shown. The last two employee tuples—‘Berger’ and ‘Benitez’—represent newly hired employees who have not yet been assigned to a department (assume that this does not violate any integrity constraints).

- If we want to retrieve a list of (Ename, Dname) values for all the employees. If we apply the NATURAL JOIN operation on EMPLOYEE and DEPARTMENT (Figure 15.2(b)), the two aforementioned tuples will not appear in the result.

- In such cases, LEFT OUTER JOIN may be used. The result is shown in Figure 15.2 (c).
(a) 

**EMPLOYEE**

<table>
<thead>
<tr>
<th>Ename</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
<th>Dnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John B.</td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>5</td>
</tr>
<tr>
<td>Wong, Franklin T.</td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Voss, Houston, TX</td>
<td>5</td>
</tr>
<tr>
<td>Zelaya, Alicia J.</td>
<td>999887777</td>
<td>1968-07-19</td>
<td>3321 Castle, Spring, TX</td>
<td>4</td>
</tr>
<tr>
<td>Wallace, Jennifer S.</td>
<td>987654321</td>
<td>1941-06-20</td>
<td>291 Berry, Bellaire, TX</td>
<td>4</td>
</tr>
<tr>
<td>Narayan, Ramesh K.</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
<td>5</td>
</tr>
<tr>
<td>English, Joyce A.</td>
<td>453453453</td>
<td>1972-07-31</td>
<td>5631 Rice, Houston, TX</td>
<td>5</td>
</tr>
<tr>
<td>Jabbar, Ahmad V.</td>
<td>987987987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>4</td>
</tr>
<tr>
<td>Borg, James E.</td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>1</td>
</tr>
<tr>
<td>Berger, Anders C.</td>
<td>999775555</td>
<td>1965-04-26</td>
<td>6530 Braes, Bellaire, TX</td>
<td>NULL</td>
</tr>
<tr>
<td>Benitez, Carlos M.</td>
<td>888664444</td>
<td>1963-01-09</td>
<td>7654 Beech, Houston, TX</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**DEPARTMENT**

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnum</th>
<th>Dmgr_ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
</tr>
</tbody>
</table>
Problems with Null Values and Dangling Tuples (3/5)

(b) Result of applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT relations.
(c) Result of applying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT relations.

<table>
<thead>
<tr>
<th>Ename</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
<th>Dnum</th>
<th>Dname</th>
<th>Dmgr_ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John B.</td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>Wong, Franklin T.</td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Voss, Houston, TX</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>Zelaya, Alicia J.</td>
<td>999887777</td>
<td>1968-07-19</td>
<td>3321 Castle, Spring, TX</td>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
</tr>
<tr>
<td>Wallace, Jennifer S.</td>
<td>987654321</td>
<td>1941-06-20</td>
<td>291 Berry, Bellaire, TX</td>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
</tr>
<tr>
<td>Narayan, Ramesh K.</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>English, Joyce A.</td>
<td>453453453</td>
<td>1972-07-31</td>
<td>5631 Rice, Houston, TX</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>Jabbar, Ahmad V.</td>
<td>987987987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
</tr>
<tr>
<td>Borg, James E.</td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
</tr>
<tr>
<td>Berger, Anders C.</td>
<td>999775555</td>
<td>1965-04-26</td>
<td>6530 Braes, Bellaire, TX</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Benitez, Carlos M.</td>
<td>888665555</td>
<td>1983-01-09</td>
<td>7654 Beech, Houston, TX</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Problems with Dangling Tuples

- Consider the decomposition of EMPLOYEE into EMPLOYEE_1 and EMPLOYEE_2 as shown in Figure 15.3 (a) and 15.3 (b).
- Their NATURAL JOIN yields the original relation EMPLOYEE in Figure 15.2(a).
- We may use the alternative representation, shown in Figure 15.3(c), where we do not include a tuple in EMPLOYEE_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE_2).
- If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result; these are called dangling tuples in EMPLOYEE.
Problems with Null Values and Dangling Tuples (5/5)

(a) **EMPLOYEE_1**

<table>
<thead>
<tr>
<th>Ename</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John B.</td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
</tr>
<tr>
<td>Wong, Franklin T.</td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Voss, Houston, TX</td>
</tr>
<tr>
<td>Zelaya, Alicia J.</td>
<td>999887777</td>
<td>1968-07-19</td>
<td>3321 Castle, Spring, TX</td>
</tr>
<tr>
<td>Wallace, Jennifer S.</td>
<td>987654321</td>
<td>1941-06-20</td>
<td>291 Berry, Bellaire, TX</td>
</tr>
<tr>
<td>Narayan, Ramesh K.</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
</tr>
<tr>
<td>English, Joyce A.</td>
<td>453453453</td>
<td>1972-07-31</td>
<td>5631 Rice, Houston, TX</td>
</tr>
<tr>
<td>Jabbar, Ahmad V.</td>
<td>987987987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
</tr>
<tr>
<td>Borg, James E.</td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
</tr>
<tr>
<td>Berger, Anders C.</td>
<td>999775555</td>
<td>1965-04-26</td>
<td>6530 Braes, Bellaire, TX</td>
</tr>
<tr>
<td>Benitez, Carlos M.</td>
<td>888665555</td>
<td>1963-01-09</td>
<td>7654 Beech, Houston, TX</td>
</tr>
</tbody>
</table>

(b) **EMPLOYEE_2**

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Dnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>5</td>
</tr>
<tr>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>999887777</td>
<td>4</td>
</tr>
<tr>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>666884444</td>
<td>5</td>
</tr>
<tr>
<td>453453453</td>
<td>5</td>
</tr>
<tr>
<td>987987987</td>
<td>4</td>
</tr>
<tr>
<td>888665555</td>
<td>NULL</td>
</tr>
<tr>
<td>999775555</td>
<td>NULL</td>
</tr>
<tr>
<td>888664444</td>
<td>NULL</td>
</tr>
</tbody>
</table>

(c) **EMPLOYEE_3**

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Dnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>5</td>
</tr>
<tr>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>999887777</td>
<td>4</td>
</tr>
<tr>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>666884444</td>
<td>5</td>
</tr>
<tr>
<td>453453453</td>
<td>5</td>
</tr>
<tr>
<td>987987987</td>
<td>4</td>
</tr>
<tr>
<td>888665555</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 15.3**
The dangling tuple problem. (a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 15.2(a) except Dnum). (b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values). (c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).
4.2 Discussion of Normalization Algorithms:

- Problems:
  - The database designer must first specify all the relevant functional dependencies among the database attributes.
  - These algorithms are not deterministic in general.
  - It is not always possible to find a decomposition into relation schemas that preserves dependencies and allows each relation schema in the decomposition to be in BCNF (instead of 3NF as in Algorithm 15.5).
### Summary of the Algorithms Discussed in This Chapter

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Input</th>
<th>Output</th>
<th>Properties/Purpose</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1</td>
<td>An attribute or a set of attributes $X$, and a set of FDs $F$</td>
<td>A set of attributes in the closure of $X$ with respect to $F$</td>
<td>Determine all the attributes that can be functionally determined from $X$</td>
<td>The closure of a key is the entire relation</td>
</tr>
<tr>
<td>15.2</td>
<td>A set of functional dependencies $F$</td>
<td>The minimal cover of functional dependencies</td>
<td>To determine the minimal cover of a set of dependencies $F$</td>
<td>Multiple minimal covers may exist—depends on the order of selecting functional dependencies</td>
</tr>
<tr>
<td>15.2a</td>
<td>Relation schema $R$ with a set of functional dependencies $F$</td>
<td>Key $K$ of $R$</td>
<td>To find a key $K$ (that is a subset of $R$)</td>
<td>The entire relation $R$ is always a default superkey</td>
</tr>
<tr>
<td>15.3</td>
<td>A decomposition $D$ of $R$ and a set $F$ of functional dependencies</td>
<td>Boolean result: yes or no for nonadditive join property</td>
<td>Testing for nonadditive join decomposition</td>
<td>See a simpler test NJB in Section 14.5 for binary decompositions</td>
</tr>
</tbody>
</table>
## Table 15.1 Summary of the Algorithms Discussed in This Chapter

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Input</th>
<th>Output</th>
<th>Properties/Purpose</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.4</td>
<td>A relation $R$ and a set of functional dependencies $F$</td>
<td>A set of relations in 3NF</td>
<td>Nonadditive join and dependency-preserving decom-</td>
<td>May not achieve BCNF, but achieves all desirable properties and 3NF</td>
</tr>
<tr>
<td>15.5</td>
<td>A relation $R$ and a set of functional dependencies $F$</td>
<td>A set of relations in BCNF</td>
<td>Nonadditive join decomposition</td>
<td>No guarantee of dependency preservation</td>
</tr>
<tr>
<td>15.6</td>
<td>A relation $R$ and a set of functional and multivalued dependencies</td>
<td>A set of relations in 4NF</td>
<td>Nonadditive join decomposition</td>
<td>No guarantee of dependency preservation</td>
</tr>
</tbody>
</table>
Definition:

- A **multivalued dependency (MVD)** $X \longrightarrow Y$ specified on relation schema $R$, where $X$ and $Y$ are both subsets of $R$, specifies the following constraint on any relation state $r$ of $R$: If two tuples $t_1$ and $t_2$ exist in $r$ such that $t_1[X] = t_2[X]$, then two tuples $t_3$ and $t_4$ should also exist in $r$ with the following properties, where we use $Z$ to denote $(R 2 (X \cup Y))$:
  - $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
  - $t_3[Y] = t_4[Y]$ and $t_4[Y] = t_2[Y]$.
  - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

- An MVD $X \longrightarrow Y$ in $R$ is called a **trivial MVD** if (a) $Y$ is a subset of $X$, or (b) $X \cup Y = R$. 
Inference Rules for Functional and Multivalued Dependencies:

- **IR1** (reflexive rule for FDs): If \( X \supseteq Y \), then \( X \rightarrow Y \).
- **IR2** (augmentation rule for FDs): \( \{X \rightarrow Y\} \cup = XZ \rightarrow YZ \).
- **IR3** (transitive rule for FDs): \( \{X \rightarrow Y, Y \rightarrow Z\} \cup = X \rightarrow Z \).
- **IR4** (complementation rule for MVDs): \( \{X \longrightarrow Y\} \cup = X \longrightarrow (R - (X \cup Y)) \).
- **IR5** (augmentation rule for MVDs): If \( X \longrightarrow Y \) and \( W \supseteq Z \) then \( WX \longrightarrow YZ \).
- **IR6** (transitive rule for MVDs): \( \{X \longrightarrow Y, Y \longrightarrow Z\} \cup = X \longrightarrow (Z - Y) \).
- **IR7** (replication rule for FD to MVD): \( \{X \rightarrow Y\} \cup = X \longrightarrow Y \).
- **IR8** (coalescence rule for FDs and MVDs): If \( X \longrightarrow Y \) and there exists \( W \) with the properties that
  - (a) \( W \cap Y \) is empty, (b) \( W \rightarrow Z \), and (c) \( Y \supseteq Z \), then \( X \rightarrow Z \).
Definition:
- A relation schema $R$ is in 4NF with respect to a set of dependencies $F$ (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency $X \longrightarrow Y$ in $F^+$, $X$ is a superkey for $R$.

  » Note: $F^+$ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state $r$ of $R$ that satisfies $F$. It is also called the closure of $F$. 
Fig. 15.4 Decomposing a relation state of EMP that is not in 4NF.
(a) EMP relation with additional tuples.
(b) Two corresponding 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.

<table>
<thead>
<tr>
<th>(a) EMP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ename</td>
<td>Phame</td>
<td>Dname</td>
</tr>
<tr>
<td>Smith</td>
<td>X</td>
<td>John</td>
</tr>
<tr>
<td>Smith</td>
<td>Y</td>
<td>Anna</td>
</tr>
<tr>
<td>Smith</td>
<td>X</td>
<td>Anna</td>
</tr>
<tr>
<td>Smith</td>
<td>Y</td>
<td>John</td>
</tr>
<tr>
<td>Brown</td>
<td>W</td>
<td>Jim</td>
</tr>
<tr>
<td>Brown</td>
<td>X</td>
<td>Jim</td>
</tr>
<tr>
<td>Brown</td>
<td>Y</td>
<td>Jim</td>
</tr>
<tr>
<td>Brown</td>
<td>Z</td>
<td>Jim</td>
</tr>
<tr>
<td>Brown</td>
<td>W</td>
<td>Joan</td>
</tr>
<tr>
<td>Brown</td>
<td>X</td>
<td>Joan</td>
</tr>
<tr>
<td>Brown</td>
<td>Y</td>
<td>Joan</td>
</tr>
<tr>
<td>Brown</td>
<td>Z</td>
<td>Joan</td>
</tr>
<tr>
<td>Brown</td>
<td>W</td>
<td>Bob</td>
</tr>
<tr>
<td>Brown</td>
<td>X</td>
<td>Bob</td>
</tr>
<tr>
<td>Brown</td>
<td>Y</td>
<td>Bob</td>
</tr>
<tr>
<td>Brown</td>
<td>Z</td>
<td>Bob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) EMP_PROJECTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ename</td>
<td>Phame</td>
</tr>
<tr>
<td>Smith</td>
<td>X</td>
</tr>
<tr>
<td>Smith</td>
<td>Y</td>
</tr>
<tr>
<td>Brown</td>
<td>W</td>
</tr>
<tr>
<td>Brown</td>
<td>X</td>
</tr>
<tr>
<td>Brown</td>
<td>Y</td>
</tr>
<tr>
<td>Brown</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMP_DEPENDENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ename</td>
<td>Dname</td>
</tr>
<tr>
<td>Smith</td>
<td>Anna</td>
</tr>
<tr>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>Brown</td>
<td>Jim</td>
</tr>
<tr>
<td>Brown</td>
<td>Joan</td>
</tr>
<tr>
<td>Brown</td>
<td>Bob</td>
</tr>
</tbody>
</table>
5.3 Non-additive (Lossless) Join Decomposition into 4NF Relations:

- **PROPERTY NJB’**
  
  » The relation schemas $R_1$ and $R_2$ form a lossless (non-additive) join decomposition of $R$ with respect to a set $F$ of functional and multivalued dependencies if and only if
  
  • $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$
  
  » or by symmetry, if and only if
  
  • $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$.
Algorithm 15.7: Relational decomposition into 4NF relations with non-additive join property

- **Input:** A universal relation R and a set of functional and multivalued dependencies F.

1. Set \( D := \{ R \} \);
2. While there is a relation schema \( Q \) in \( D \) that is not in 4NF do {
   choose a relation schema \( Q \) in \( D \) that is not in 4NF;
   find a nontrivial MVD \( X \rightarrow>> Y \) in \( Q \) that violates 4NF;
   replace \( Q \) in \( D \) by two relation schemas \( (Q - Y) \) and \( (X \cup Y) \);
};
Join Dependency was defined in the previous section:

Definition:

- A **join dependency** (JD), denoted by \( JD(R_1, R_2, \ldots, R_n) \), specified on relation schema \( R \), specifies a constraint on the states \( r \) of \( R \).
  
  » The constraint states that every legal state \( r \) of \( R \) should have a non-additive join decomposition into \( R_1, R_2, \ldots, R_n \); that is, for every such \( r \) we have
    \[
    * (\pi_{R_1}(r), \pi_{R_2}(r), \ldots, \pi_{R_n}(r)) = r
    \]

  **Note:** an MVD is a special case of a JD where \( n = 2 \).

- A join dependency \( JD(R_1, R_2, \ldots, R_n) \), specified on relation schema \( R \), is a **trivial JD** if one of the relation schemas \( R_i \) in \( JD(R_1, R_2, \ldots, R_n) \) is equal to \( R \).
Definition of 5NF:

- A relation schema \( R \) is in **fifth normal form (5NF)** (or **Project-Join Normal Form (PJNF)**) with respect to a set \( F \) of functional, multivalued, and join dependencies if,
  
  - for every nontrivial join dependency \( JD(R_1, R_2, ..., R_n) \) in \( F^+ \) (that is, implied by \( F \)),
    
    - every \( R_i \) is a superkey of \( R \).

- Discovering join dependencies in practical databases with hundreds of relations is next to impossible. Therefore, 5NF is rarely used in practice.
Inclusion Dependencies (1/2)

Definition:

- An inclusion dependency \( R.X < S.Y \) between two sets of attributes—\( X \) of relation schema \( R \), and \( Y \) of relation schema \( S \)—specifies the constraint that, at any specific time when \( r \) is a relation state of \( R \) and \( s \) a relation state of \( S \), we must have

\[
\pi_X(r(R)) \subseteq \pi_Y(s(S))
\]

- **Note:**
  - The \( \subseteq \) (subset) relationship does not necessarily have to be a proper subset.
  - The sets of attributes on which the inclusion dependency is specified—\( X \) of \( R \) and \( Y \) of \( S \)—must have the same number of attributes.
  - In addition, the domains for each pair of corresponding attributes should be compatible.
Objective of Inclusion Dependencies:

To formalize two types of interrelational constraints which cannot be expressed using F.D.s or MVDs:

- Referential integrity constraints
- Class/subclass relationships

Inclusion dependency inference rules

- **IDIR1 (reflexivity):** \( R.X < R.X \)
- **IDIR2 (attribute correspondence):** If \( R.X < S.Y \)
  - where \( X = \{A_1, A_2, \ldots, A_n\} \) and \( Y = \{B_1, B_2, \ldots, B_n\} \) and \( A_i \text{ Corresponds-to } B_i \), then \( R.A_i < S.B_i \)
  - for \( 1 \leq i \leq n \).
- **IDIR3 (transitivity):** If \( R.X < S.Y \) and \( S.Y < T.Z \), then \( R.X < T.Z \).
Arithmetic Functions:

- As long as a unique value of \( Y \) is associated with every \( X \), we can still consider that the FD \( X \rightarrow Y \) exists.

For example, consider the relation:

\[ \text{ORDER\_LINE} \ (\text{Order}\#, \text{Item}\#, \text{Quantity}, \text{Unit\_price}, \text{Extended\_price}, \text{Discounted\_price}) \]

- Each tuple represents an item from an order with a particular quantity, and the price per unit for that item. In this relation, 
  
  \((\text{Quantity}, \text{Unit\_price}) \rightarrow \text{Extended\_price}\) by the formula
  
  \[ \text{Extended\_price} = \text{Quantity} \times \text{Unit\_price}. \]

- Hence, there is a unique value for \( \text{Extended\_price} \) for every pair \((\text{Quantity}, \text{Unit\_price})\), and thus it conforms to the definition of functional dependency.
Procedures:

- There may be a procedure that takes into account the quantity discounts, the type of item, and so on and computes a discounted price for the total quantity ordered for that item. Therefore, we can say

- \((\text{Item#}, \text{Quantity}, \text{Unit\_price}) \rightarrow \text{Discounted\_price}, \text{or} \)
- \((\text{Item#}, \text{Quantity}, \text{Extended\_price}) \rightarrow \text{Discounted\_price}.\)
6.4 Domain-Key Normal Form (DKNF):

- **Definition:**
  - A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.

- The idea is to specify (theoretically, at least) the “ultimate normal form” that takes into account all possible types of dependencies and constraints.

- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.

- The practical utility of DKNF is limited.
Recap

- Functional Dependencies Revisited
- Designing a Set of Relations by Synthesis
- Properties of Relational Decompositions
- Algorithms for Relational Database Schema Design in 3Nf and BCNF
- Multivalued Dependencies and Fourth Normal Form
- Other Dependencies and Normal Forms
Agenda

1. Session Overview
2. Functional Dependencies & Normalization for RDBs
3. RDB Design Algorithms and Further Dependencies
4. Summary and Conclusion
Summary

- Functional Dependencies and Normalization for RDBs
- RDB Design Algorithms and Further Dependencies
- Summary & Conclusion
Assignments & Readings

- **Readings**
  - Slides and Handouts posted on the course web site
  - Textbook: Chapters 14 and 15

- **Assignment #7 – due on 5/06/16**
  - Optional exercises (Extra Credit): 15.21, 15.28, 15.31
  - Note: MVDs and Design Algorithms will be covered in class on 4/29/16

- **Database Project Part II – Logical Schema Optimization**
  - due on 4/29/16
Physical design of the database using various file organization and indexing techniques for efficient query processing
Any Questions?