CSIE30600/CSIEB0290
Database Systems

Lecture 10:
Relational Database Design I

Outline

- Informal Design Guidelines for Relational Databases
  - Semantics of the Relation Attributes
  - Redundant Information in Tuples and Update Anomalies
  - Null Values in Tuples
  - Spurious Tuples
- Functional Dependencies (FDs)
  - Definition of FD
  - Inference Rules for FDs
  - Equivalence of Sets of FDs
  - Minimal Sets of FDs

Outline

- Normal Forms Based on Primary Keys
  - Normalization of Relations
  - Practical Use of Normal Forms
  - Definitions of Keys and Attributes Participating in Keys
  - First Normal Form
  - Second Normal Form
  - Third Normal Form
- General Normal Form Definitions (For Multiple Keys)
- BCNF (Boyce-Codd Normal Form)

Relational Database Design

- Relational database design requires that we find a "good" collection of relation schemas
- A bad design may lead to
  - Repetition of Information
  - Inability to represent certain information
- **Design Goals**:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates for violation of database integrity constraints

Informal Design Guidelines (1)

- 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?

Informal Design Guidelines (2)

- 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 next lecture.

Measures of Quality

- Making sure attribute semantics are clear
- Reducing redundant information in tuples
- Reducing NULL values in tuples
- Disallowing possibility of generating spurious tuples

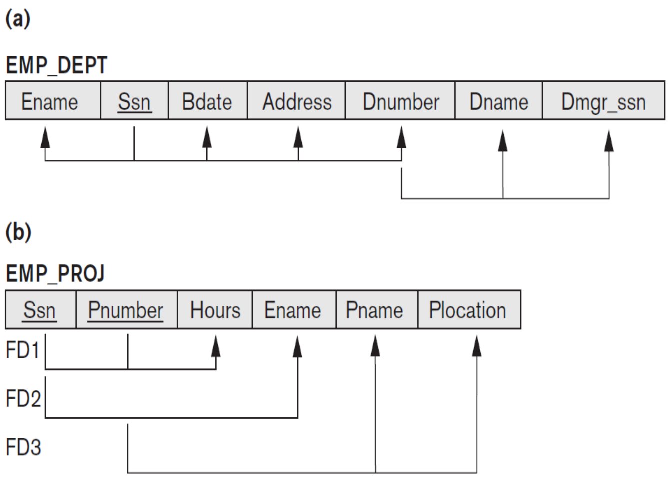
Guideline 1

- Design relation schema so that it is easy to explain its meaning
- Each tuple in a relation should represent one entity or relationship instance.
- Do not combine attributes from multiple entity types and relationship types into a single 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*.

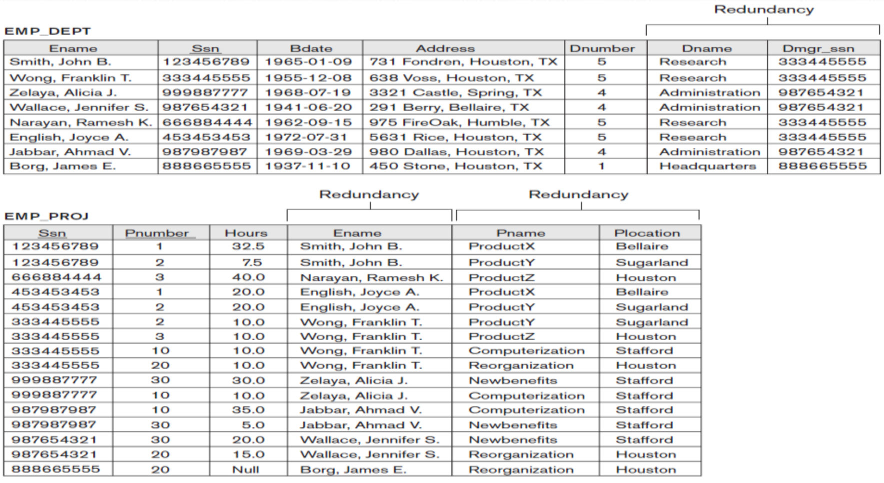
Guideline 1 (cont’d.)

- Example of violating Guideline 1: Figure 14.3

Figure 14.3
Two relation schemas suffering from update anomalies.
(a) EMP_DEPT and
(b) EMP_PROJ.



Example States for EMP_DEPT and EMP_PROJ



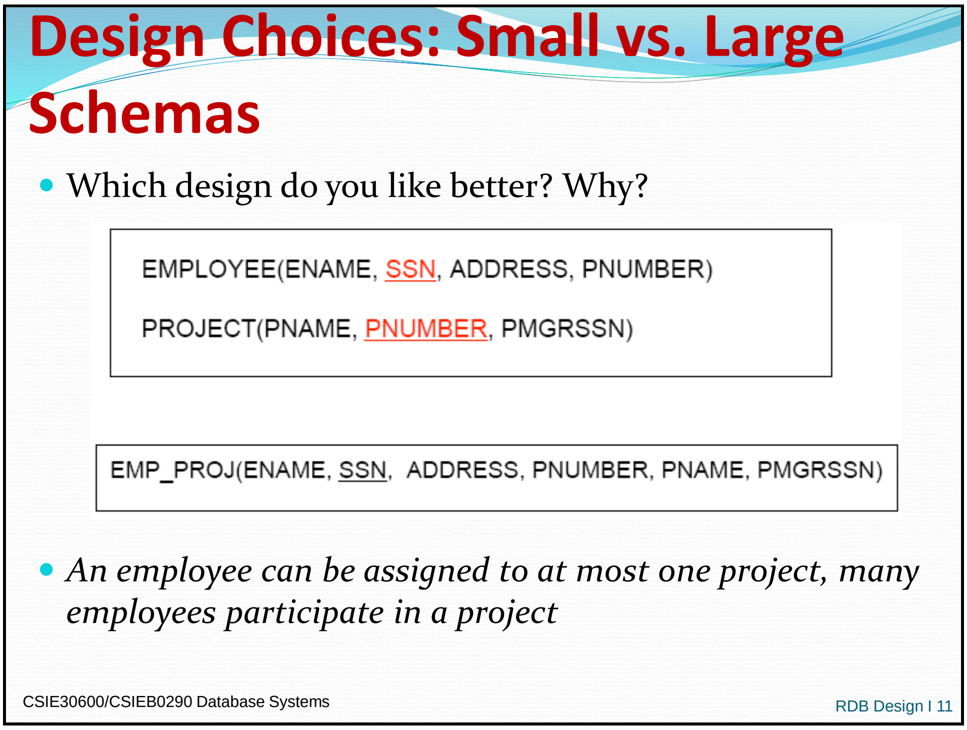
EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

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.



Design Choices: Small vs. Large Schemas

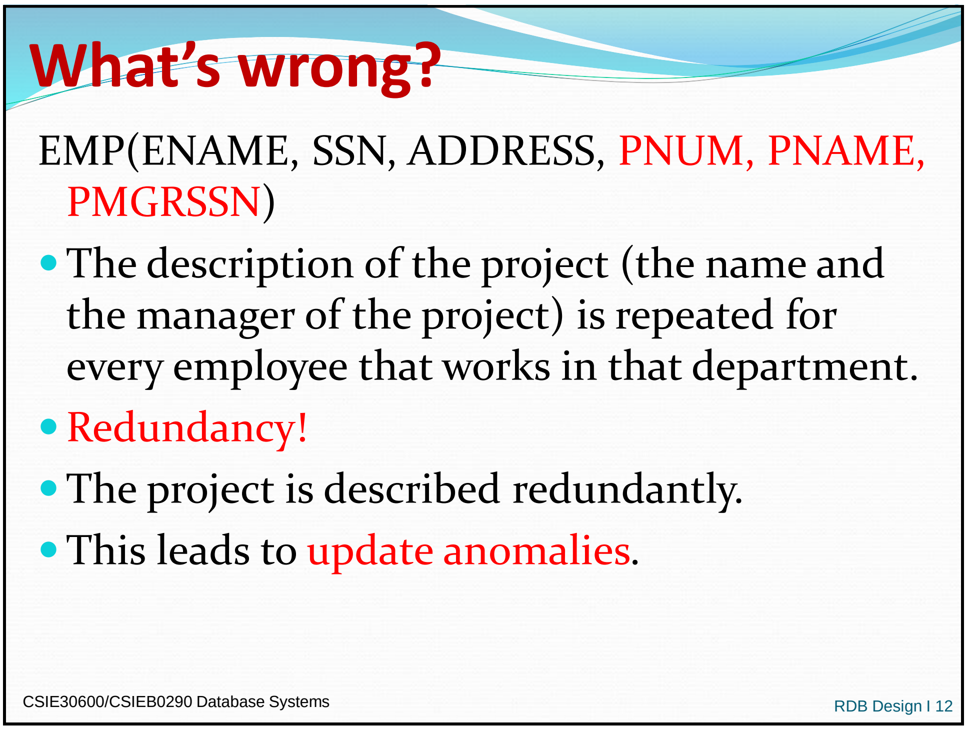
- Which design do you like better? Why?

EMPLOYEE(ENAME, SSN, ADDRESS, PNUMBER)

PROJECT(PNAME, PNUMBER, PMGRSSN)

EMP_PROJ(ENAME, SSN, ADDRESS, PNUMBER, PNAME, PMGRSSN)

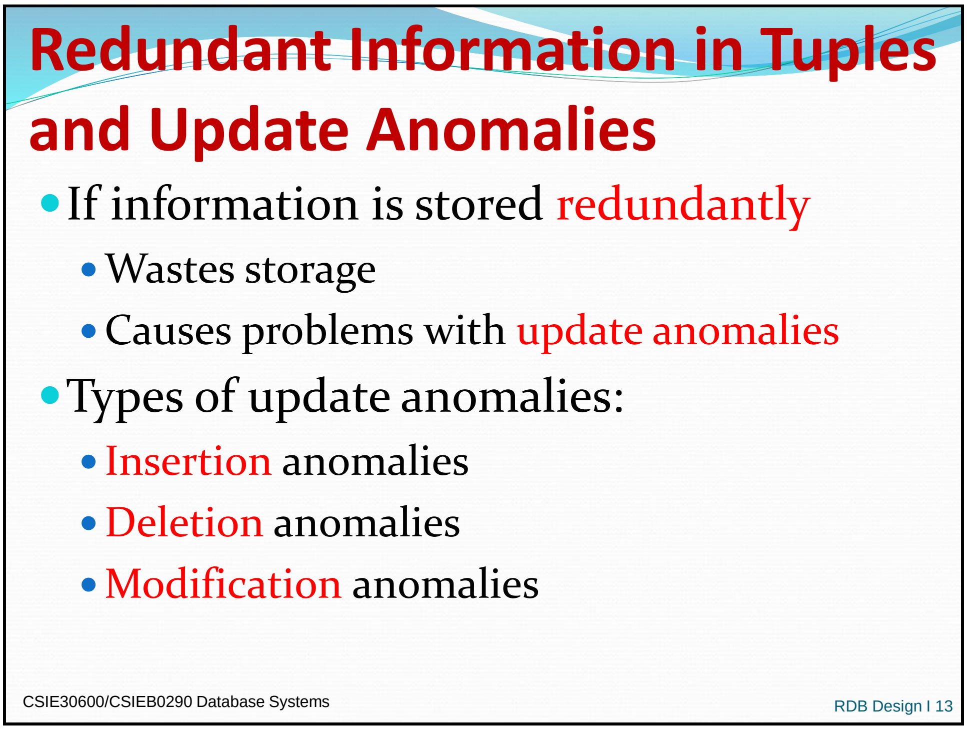
- *An employee can be assigned to at most one project, many employees participate in a project*



What's wrong?

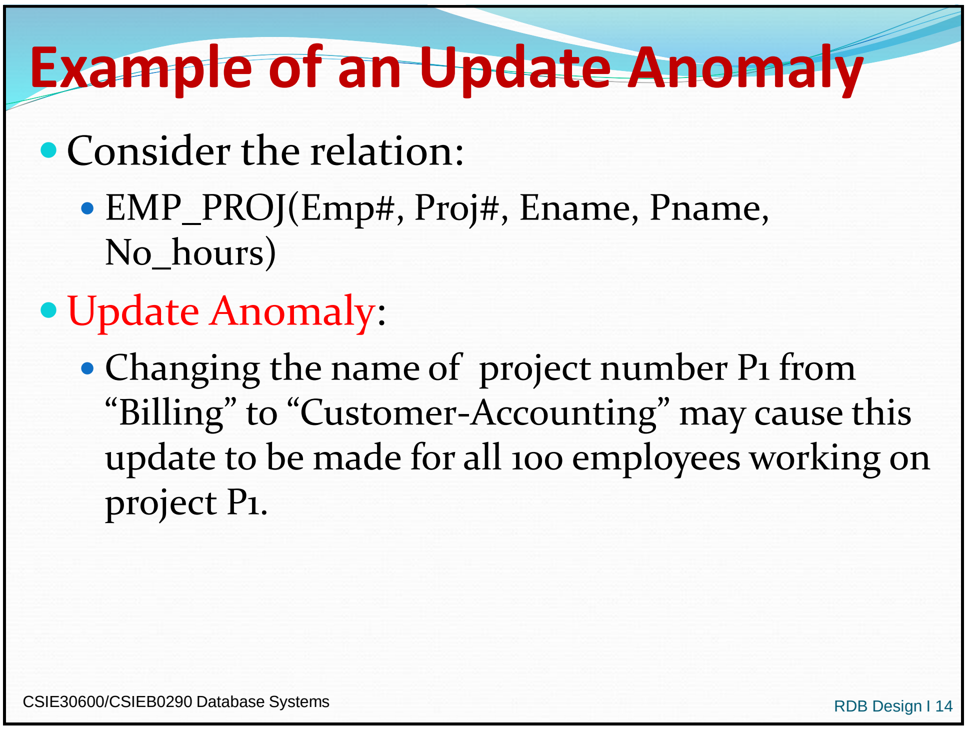
EMP(ENAME, SSN, ADDRESS, PNUM, PNAME, PMGRSSN)

- The description of the project (the name and the manager of the project) is repeated for every employee that works in that department.
- Redundancy!
- The project is described redundantly.
- This leads to update anomalies.



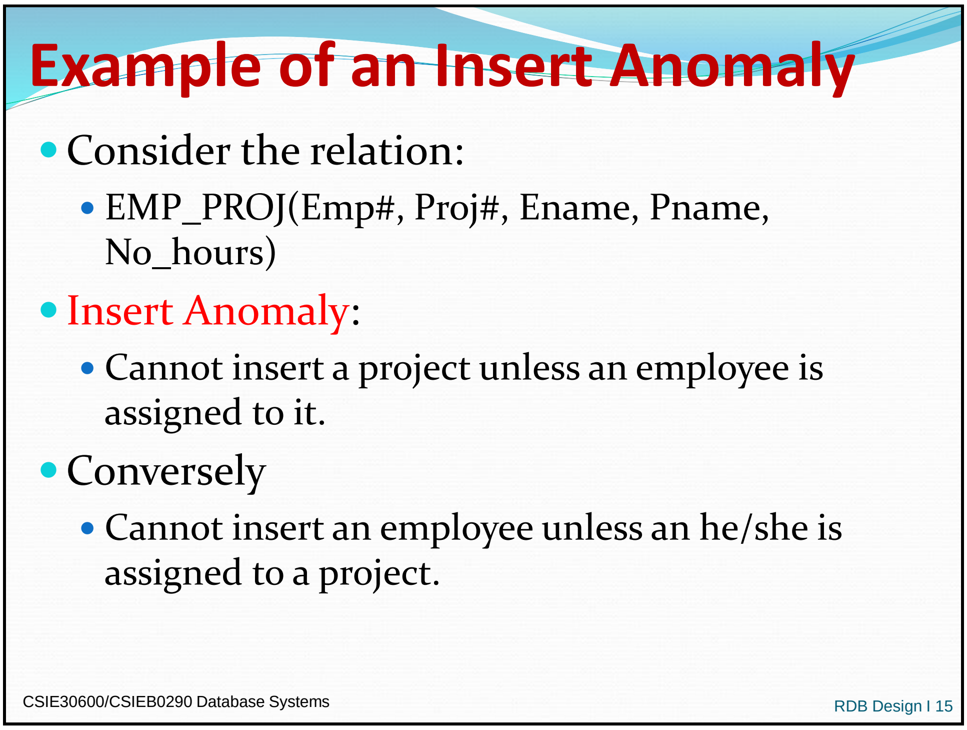
Redundant Information in Tuples and Update Anomalies

- If information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
- Types of update anomalies:
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies



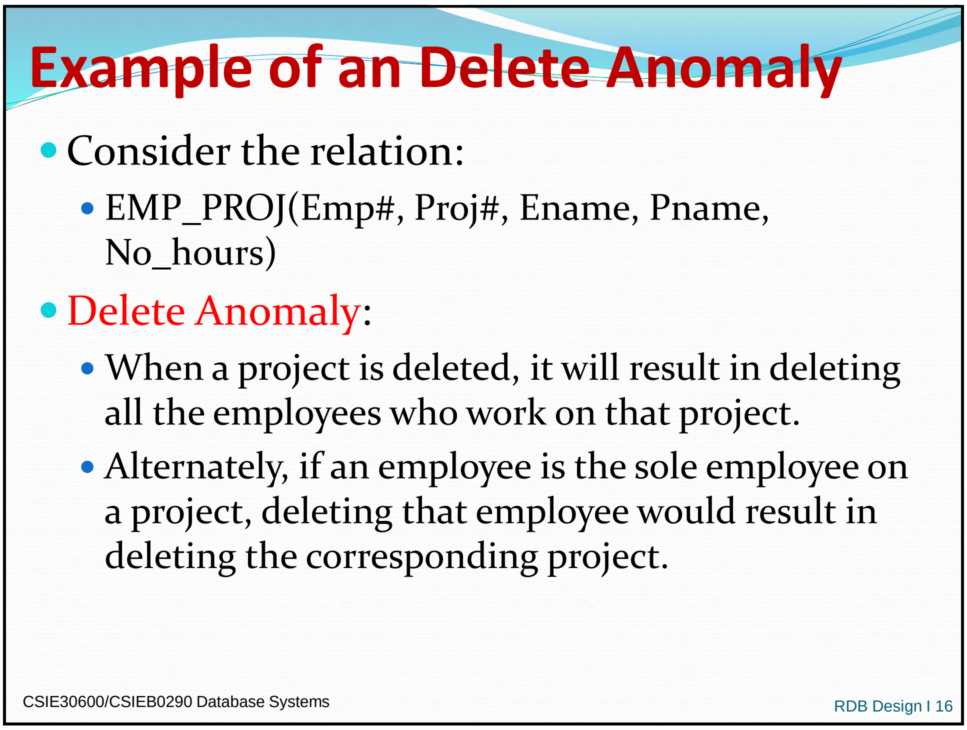
Example of an Update Anomaly

- 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 an 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.

Guideline 2

- Design base relation schemas so that no update anomalies are present in the relations
- If any anomalies are present:
  - Note them clearly
  - Make sure that the programs that update the database will operate correctly

NULL Values in Tuples

- Some designers may group many attributes together into a “fat” relation
  - Can end up with many NULLs
- Problems with NULLs
  - Wasted storage space
  - Problems understanding meaning

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
- If NULLs are unavoidable:
  - Make sure that they apply in exceptional cases only, not to a majority of tuples
- Reasons for NULL s:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

Spurious(偽、假) Tuples

- Bad schema designs may result in erroneous results for certain JOIN operations
- Figure 14.5(a)
  - Relation schemas EMP_LOCS and EMP_PROJ1
- NATURAL JOIN
  - Result produces many more tuples than the original set of tuples in EMP_PROJ
  - Called **spurious tuples**
  - Represent spurious information that is not valid

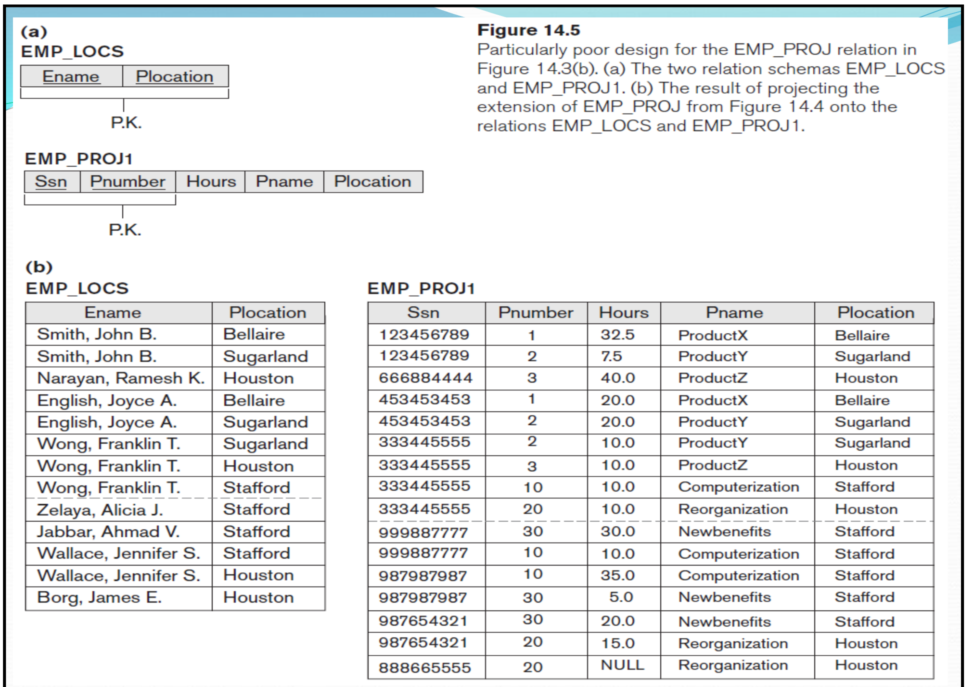


Figure 14.5
Particularly poor design for the EMP_PROJ relation in Figure 14.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 14.4 onto the relations EMP_LOCS and EMP_PROJ1.

(b)

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

Guideline 4

- Design relation schemas to be joined with equality conditions on attributes that are appropriately related
  - Guarantees that no spurious tuples are generated
- The "lossless join" property is used to guarantee meaningful results for join operations (more about this later)
- **Avoid** relations that contain matching attributes that are **NOT** (foreign key, primary key) combinations

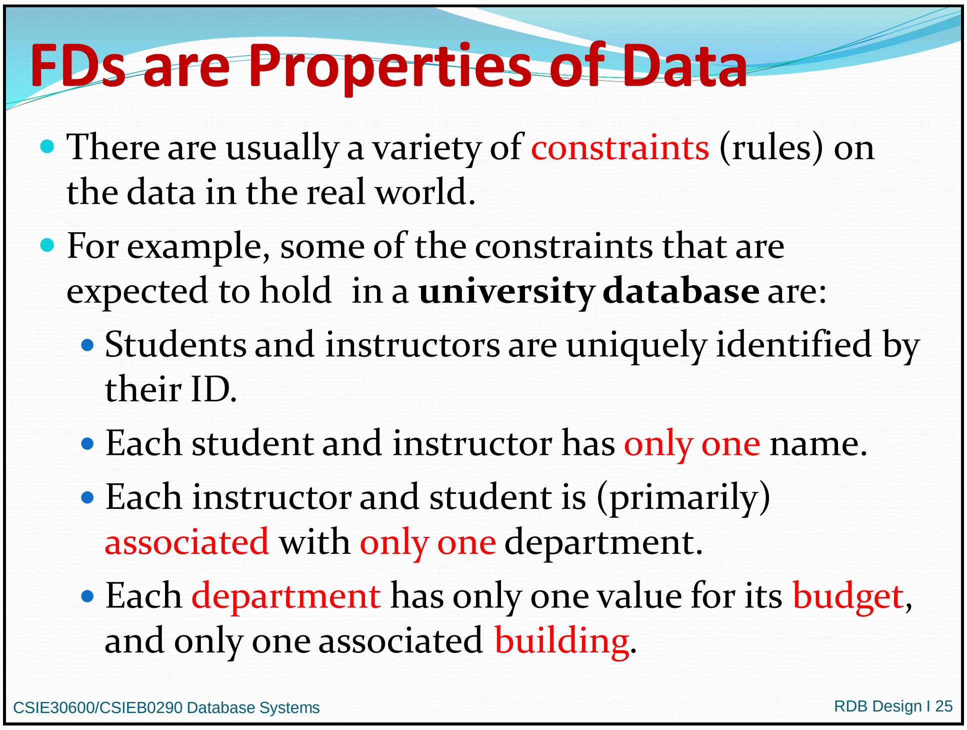
Summary of Design Guidelines

- Anomalies cause redundant work to be done
- Waste of storage space due to NULLs
- Difficulty of performing operations and joins due to NULL values
- Generation of invalid and spurious data during joins

- A good design should avoid all problems above.

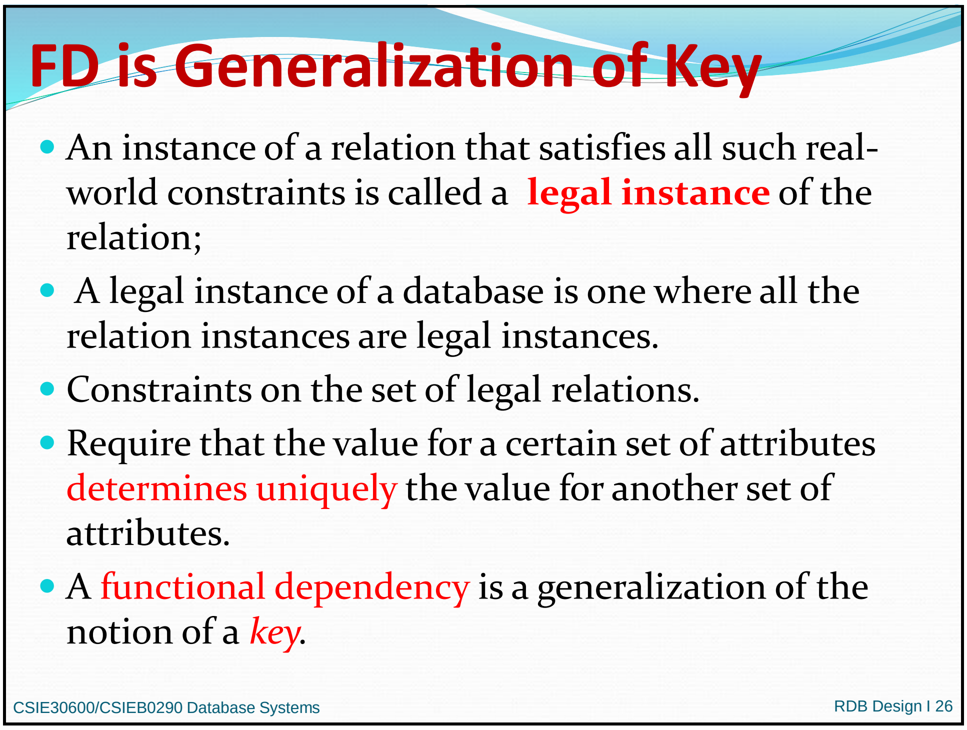
Functional Dependencies (FDs)

- Formal tool for analysis of relational schemas
- Enables us to detect and describe some of the above-mentioned problems in precise terms
- Theory of functional dependency



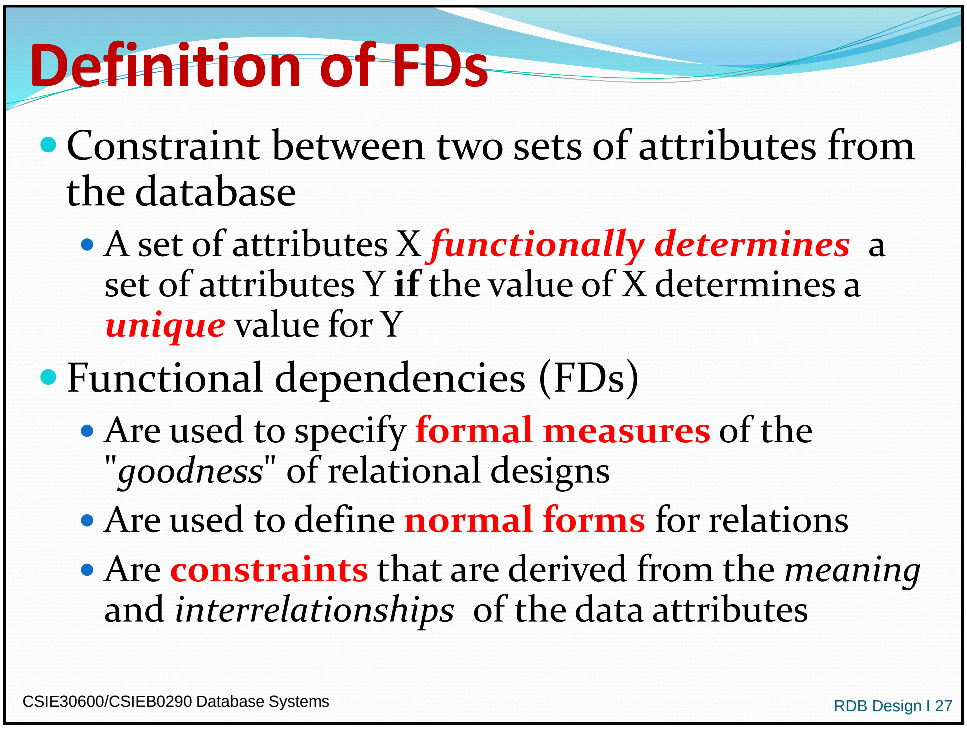
FDs are Properties of Data

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a **university database** are:
  - Students and instructors are uniquely identified by their ID.
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.



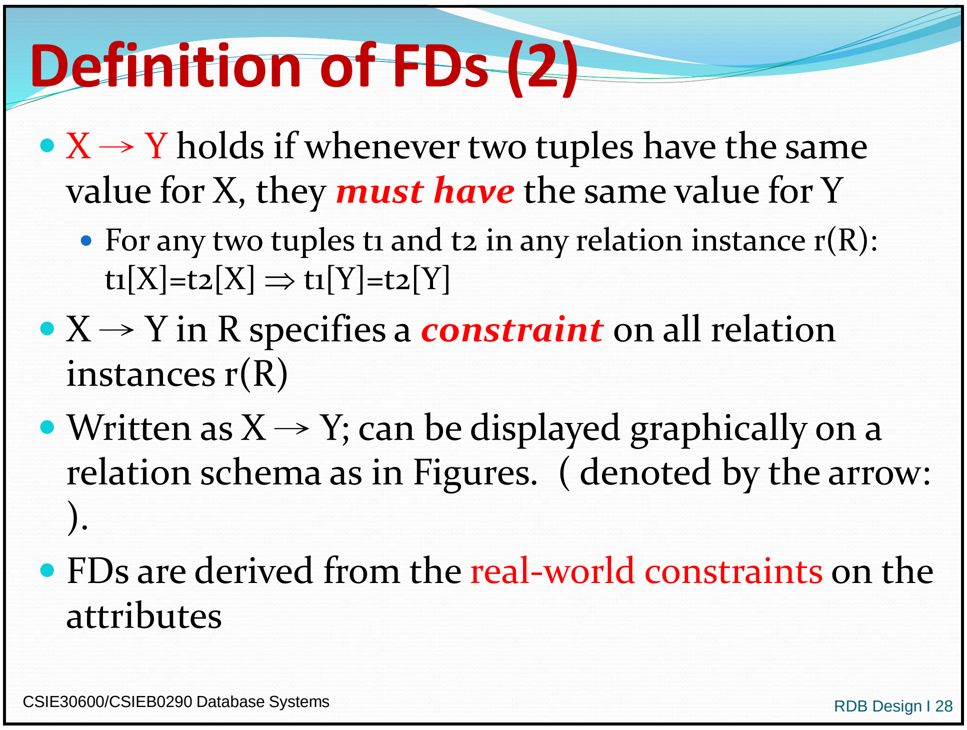
FD is Generalization of Key

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation;
- A legal instance of a database is one where all the relation instances are legal instances.
- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.



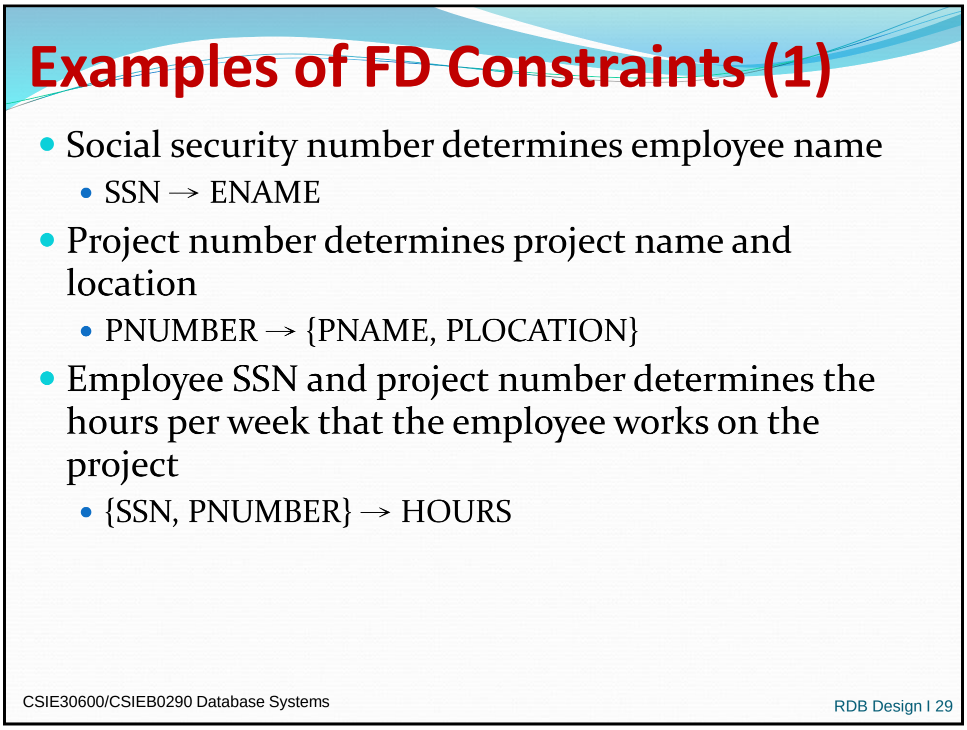
Definition of FDs

- Constraint between two sets of attributes from the database
  - A set of attributes X ***functionally determines*** a set of attributes Y **if** the value of X determines a ***unique*** value for Y
- Functional dependencies (FDs)
  - Are used to specify **formal measures** of the "*goodness*" of relational designs
  - Are used to define **normal forms** for relations
  - Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes



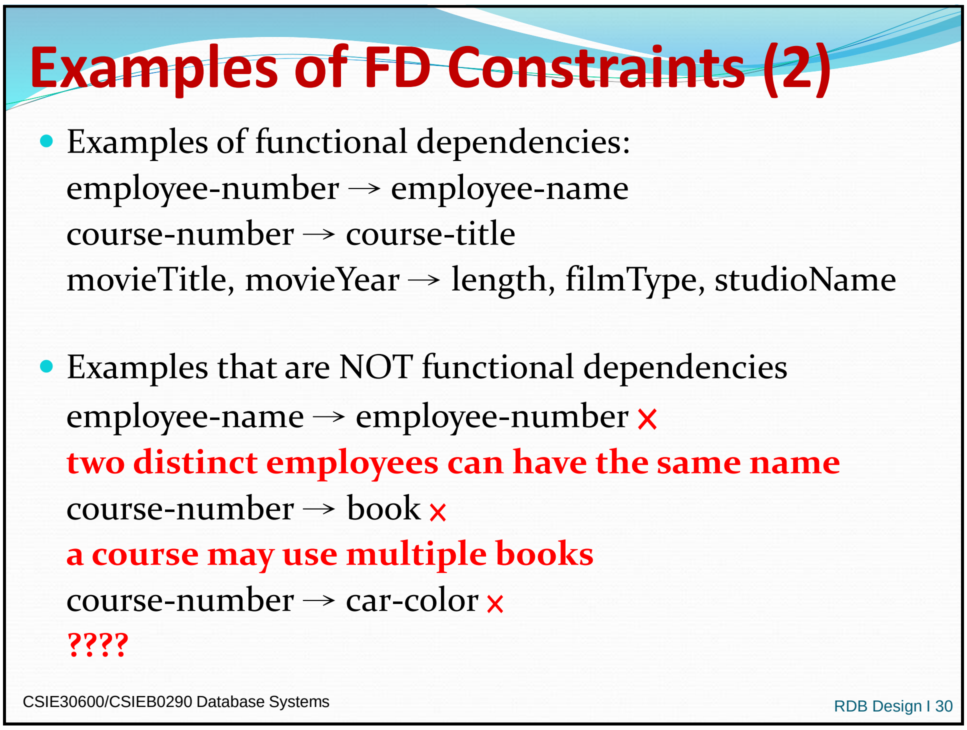
Definition of FDs (2)

- 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): $t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$
- 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)

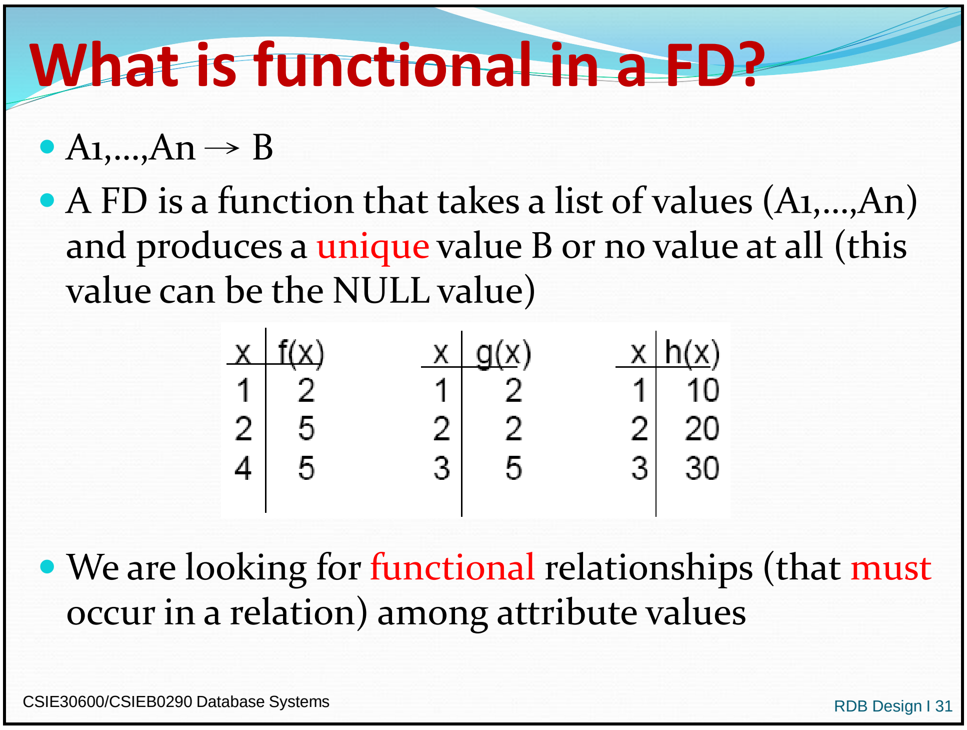
- Social security number determines employee name
  - SSN → ENAME
- Project number determines project name and location
  - PNUMBER → {PNAME, PLOCATION}
- Employee SSN and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} → HOURS



Examples of FD Constraints (2)

- Examples of functional dependencies:
  employee-number → employee-name
  course-number → course-title
  movieTitle, movieYear → length, filmType, studioName

- Examples that are NOT functional dependencies
  employee-name → employee-number ×
  two distinct employees can have the same name
  course-number → book ×
  a course may use multiple books
  course-number → car-color ×
  ????



What is functional in a FD?

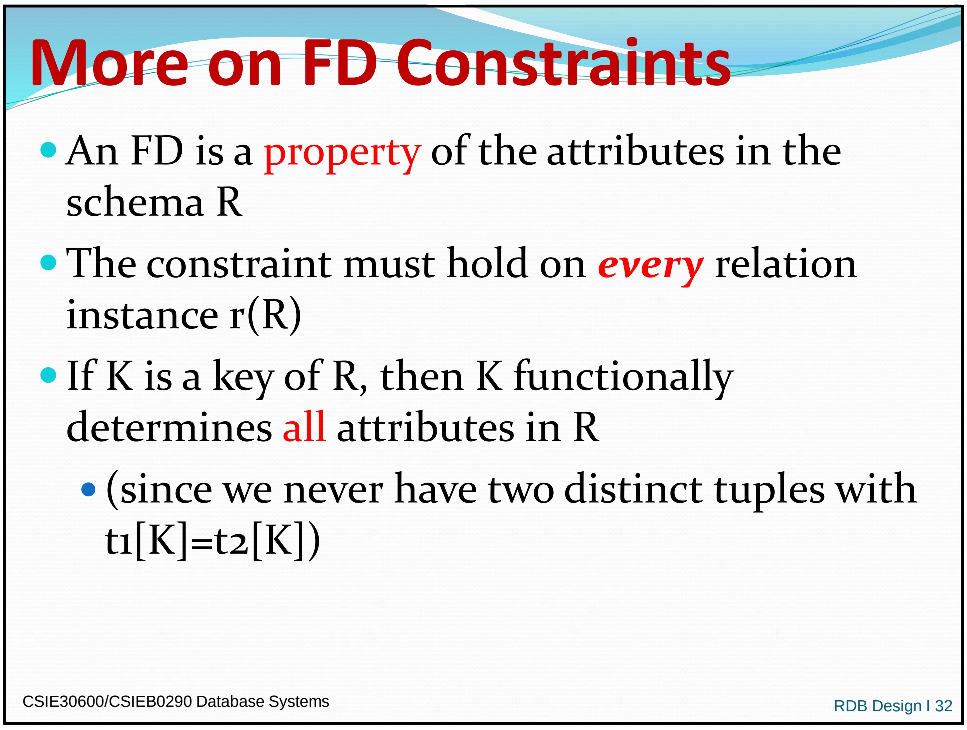
- A1,…,An → B
- A FD is a function that takes a list of values (A1,…,An) and produces a unique value B or no value at all (this value can be the NULL value)

x	f(x)
1	2
2	5
4	5

x	g(x)
1	2
2	2
3	5

x	h(x)
1	10
2	20
3	30

- We are looking for functional relationships (that must occur in a relation) among attribute values



More on FD Constraints

- 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])

Keys and FDs

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  $in\_dep\ (\underline{ID,}\ name,\ salary\underline{,\ dept\_name,}\ building,\ budget\ ).$

  We expect these functional dependencies to hold:

  $dept\_name \rightarrow building$

  $ID \rightarrow building$

  but would not expect the following to hold:

  $dept\_name \rightarrow salary$

Use of FDs

- We use functional dependencies to:
  - To test relations to see if they are legal under a given set of FDs.
    - If a relation r is legal under a set F of FDs, we say that r **satisfies** F.
  - To specify constraints on the set of legal relations
    - We say that F **holds on** R if all legal relations on R satisfy the set of functional dependencies F.
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy the FD $name \rightarrow ID$.

Inference Rules for FDs (1)

- 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-of* X, then X → Y
  - IR2. (**Augmentation**) If X → Y, then XZ → YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X → Y and Y → Z, then X → Z
- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold
  - All other rules that hold can be deduced from these

Inference Rules for FDs (2)

- Some additional inference rules that are useful:
  - **Decomposition**: If X → YZ, then X → Y and X → Z
  - **Union**: If X → Y and X → Z, then X → YZ
  - **Psuedotransitivity**: If X → Y and WY → Z, then WX → Z
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

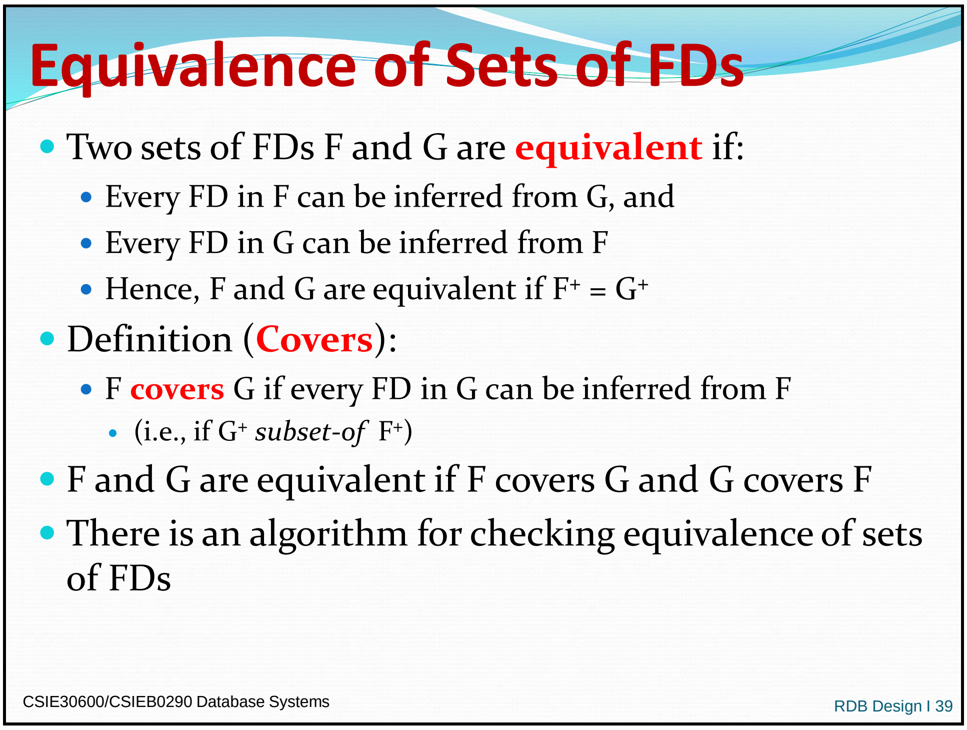
Example: Using Inference Rules

- Prove that if $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

1. $X \rightarrow Y$ (given)
2. $XZ \rightarrow YZ$ (1 and Augmentation)
3. $Z \rightarrow W$ (given)
4. $YZ \rightarrow YW$ (3 and Augmentation)
5. $XZ \rightarrow YW$ (2, 4, and Transitivity)

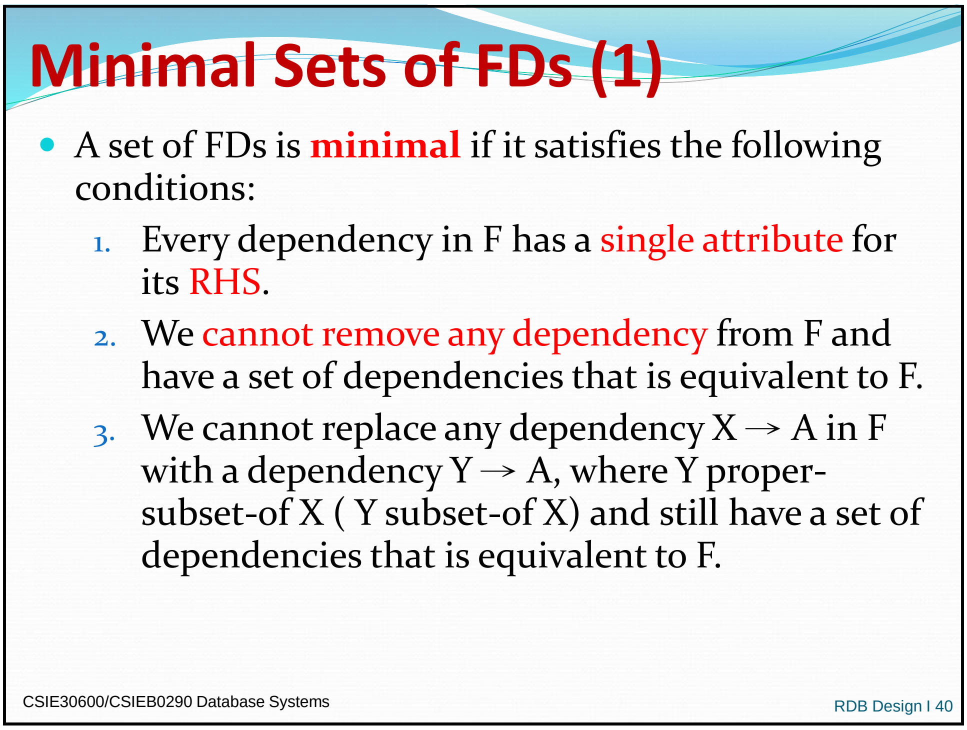
Closure

- **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
- *If we know how to compute the closure of any set of attributes, we can test if any given FD $A_1,\ldots,A_n \rightarrow B$ follows from a set of FDs F*
  - *Compute $\{A_1,\ldots,A_n\}^+$*
  - *If $B \in \{A_1,\ldots,A_n\}^+$, then $A_1,\ldots,A_n \rightarrow B$*



Equivalence of Sets of FDs

- 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-of* 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



Minimal Sets of FDs (1)

- A set of FDs is **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 → A in F with a dependency Y → A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs (2)

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

Normal Forms Based on Primary Keys

- Normalization of relations
- Approaches for relational schema design
  - Perform a conceptual schema design using a conceptual model then map conceptual design into a set of relations
  - Design relations based on external knowledge derived from existing implementation of files or forms or reports

Normalization of Relations (1)

- **Normalization**:
  - Takes a schema through a series of tests
  - Certify whether it satisfies a certain normal form
  - Decompose unsatisfactory "bad" relations into smaller "good" relations
- **Normal form**:
  - Conditions that must be satisfied for a relation schema to be in a particular "good" form

Normalization of Relations (2)

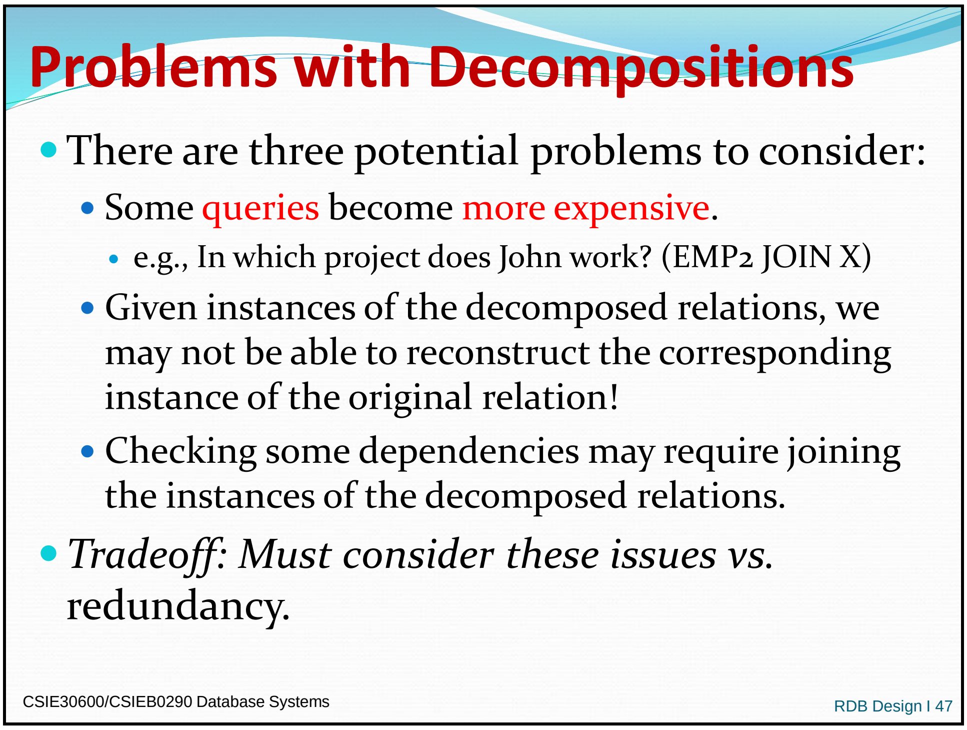
- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- 4NF
  - based on keys, multi-valued dependencies : MVDs;
- 5NF
  - based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design

Desirable Properties of Relational Schemas

- **Nonadditive join property (lossless join)**
  - Extremely critical
- **Dependency preservation property**
  - Desirable but sometimes sacrificed for other factors

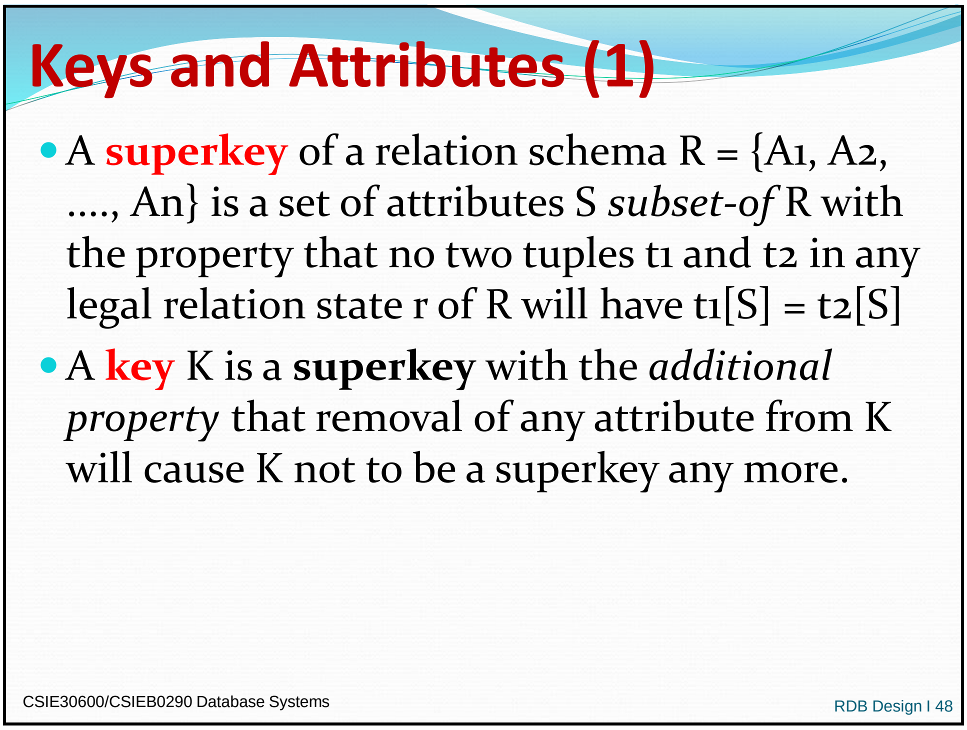
Practical Use of Normal Forms

- **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, BCNF. 4NF and further are rarely used)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form



Problems with Decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., In which project does John work? (EMP2 JOIN X)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  - Checking some dependencies may require joining the instances of the decomposed relations.
- *Tradeoff: Must consider these issues vs.* redundancy.



Keys and Attributes (1)

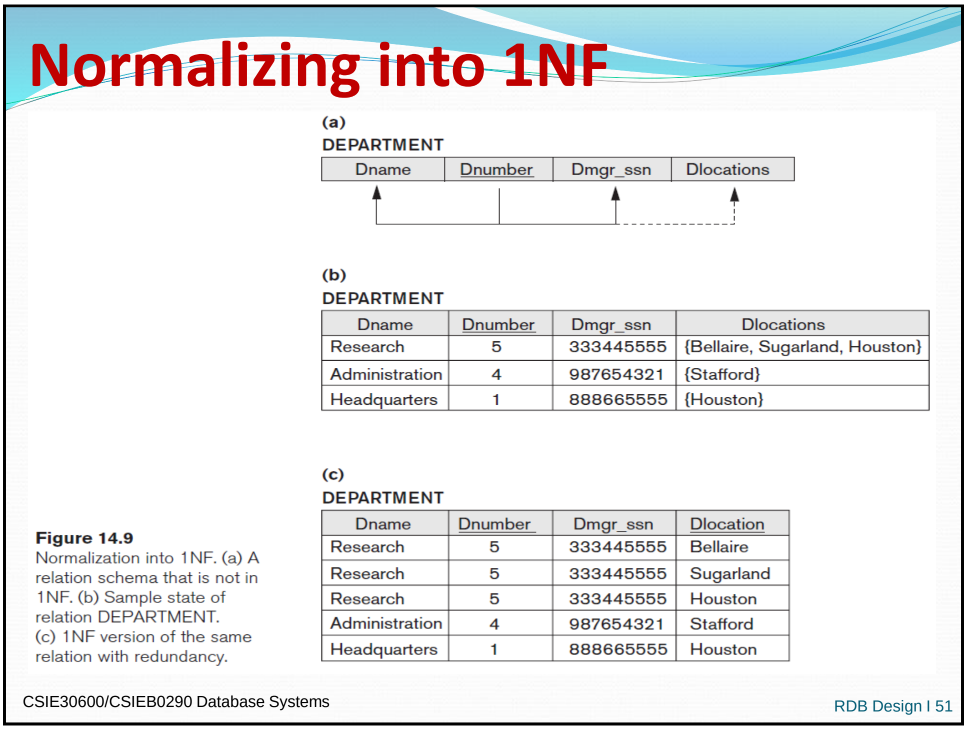
- A **superkey** of a relation schema R = {A1, A2, …., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]
- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

Keys and Attributes(2)

- 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.

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 the basic (flat) relational model
- Most RDBMSs allow only those relations to be defined that are in First Normal Form



Normalizing into 1NF

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

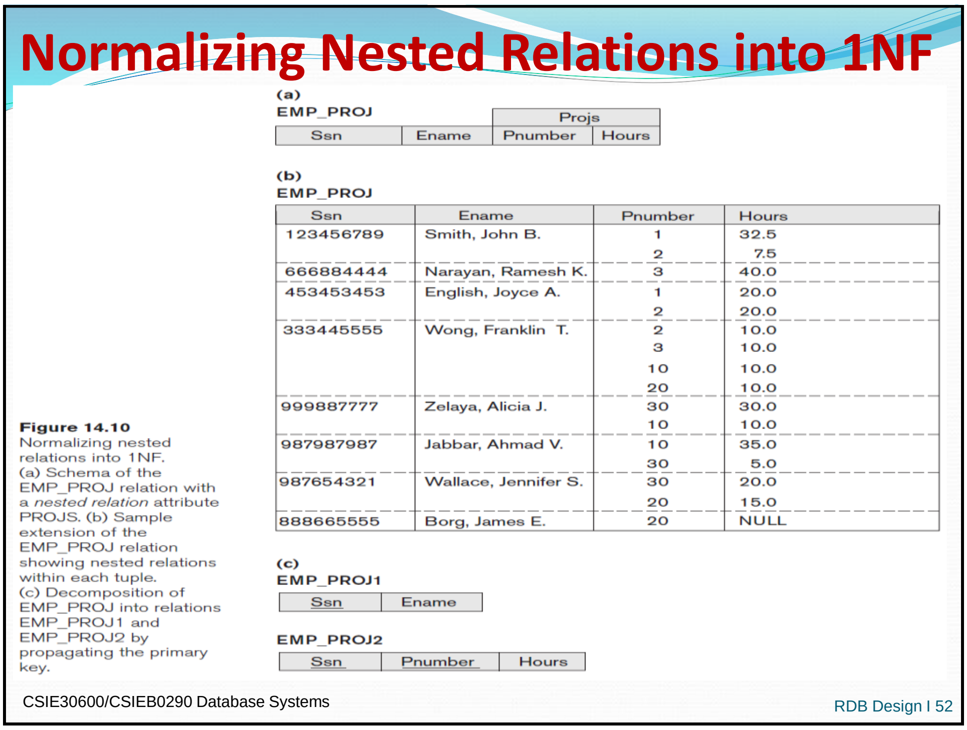
Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 14.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.



Normalizing Nested Relations into 1NF

(a)

EMP_PROJ

Ssn	Ename	Projs	
		Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

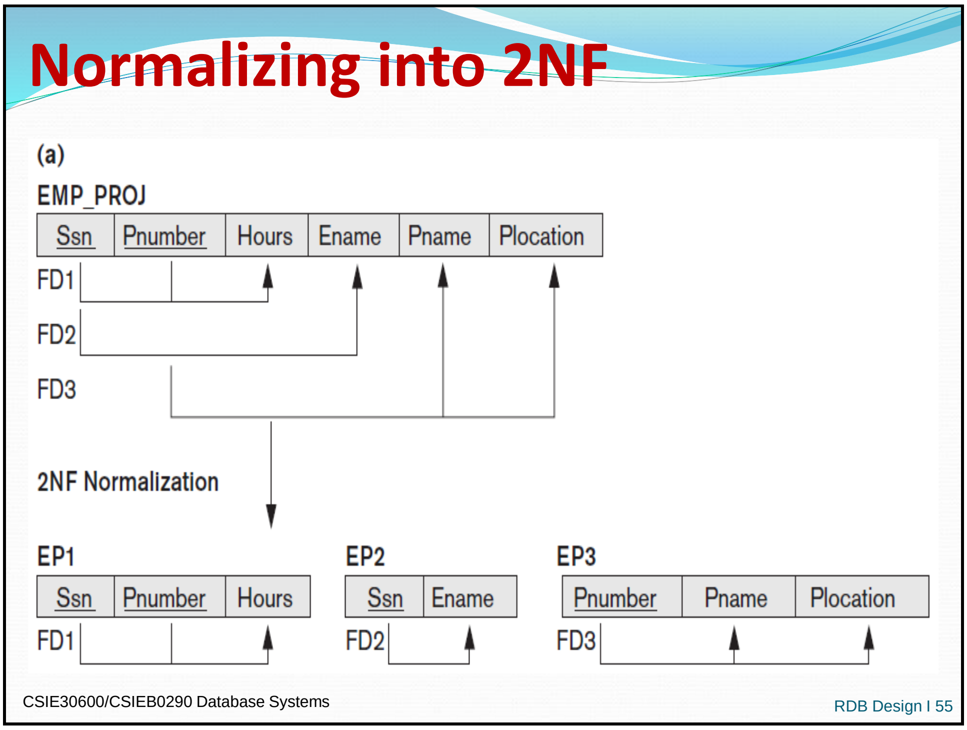
Figure 14.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Second Normal Form (1)

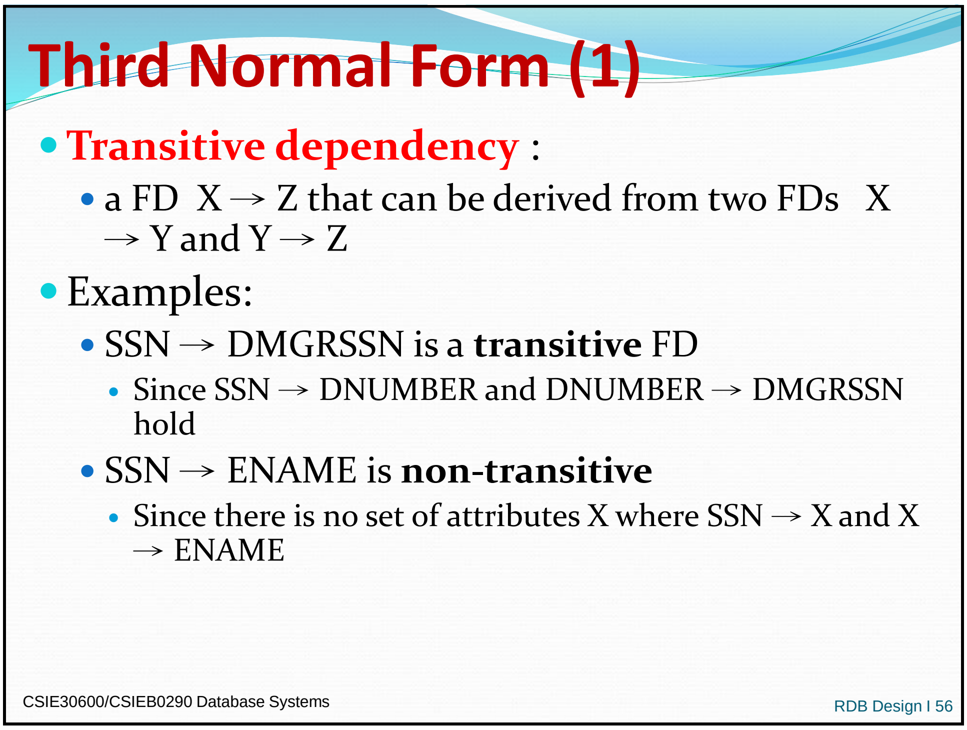
- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute**: An attribute that is member of any candidate key K
  - **Full functional dependency**: a FD Y → Z where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {SSN, PNUMBER} → HOURS is a full FD since neither SSN → HOURS nor PNUMBER → HOURS hold
  - {SSN, PNUMBER} → ENAME is not a full FD (it is called a partial dependency) since SSN → ENAME also holds

Second Normal Form (2)

- 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



Normalizing into 2NF



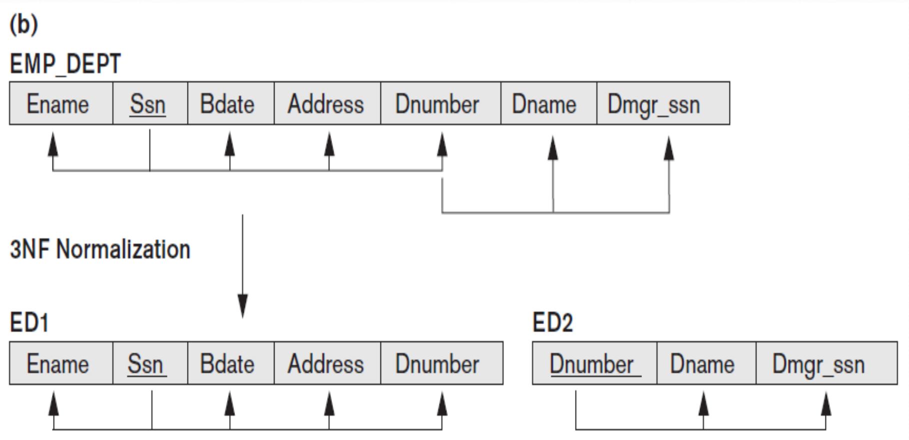
Third Normal Form (1)

- **Transitive dependency** :
  - a FD X → Z that can be derived from two FDs X → Y and Y → Z
- Examples:
  - SSN → DMGRSSN is a **transitive** FD
    - Since SSN → DNUMBER and DNUMBER → DMGRSSN hold
  - SSN → ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN → X and X → ENAME

Third Normal Form (2)

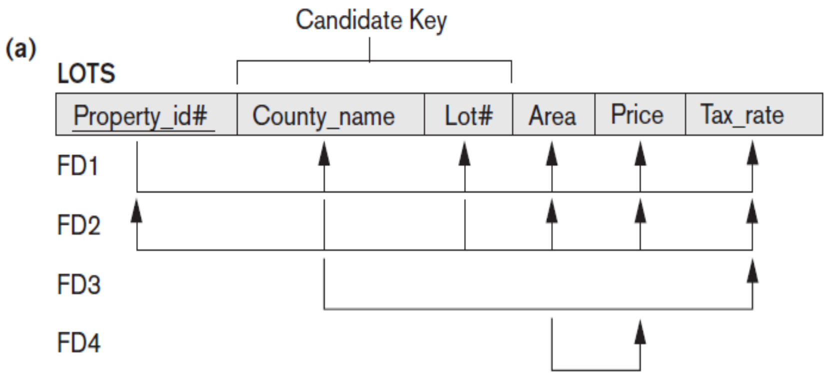
- A relation schema R is in **third normal form** (3NF) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key.
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In X → Y and Y → Z, with X as the primary key, we consider 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 → Emp# → Salary and Emp# is a candidate key.

Normalizing into 3NF

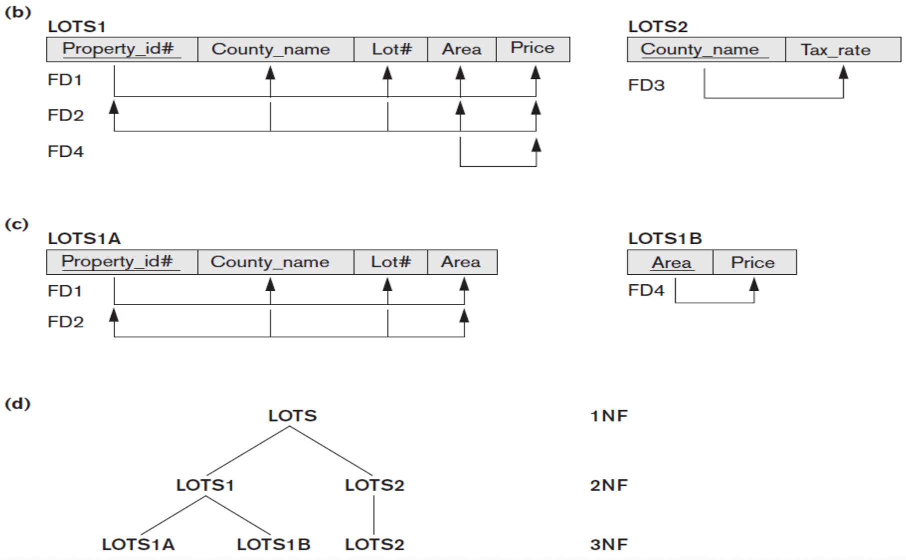


Normalization into 2NF and 3NF

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.



Normalization into 2NF and 3NF



Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

General Definition of Second Normal Form

- The above definitions consider the primary key only.
- The following more general definitions take into account relations with multiple candidate keys.
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is not partially dependent on *any* key of R.

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 (the main point)
    - (b) A is a prime attribute of R (not a problem)
- NOTE: Boyce-Codd normal form disallows condition (b) above (slide 65)

Alternative Definition of 3NF

- A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:
  - It is fully functionally dependent on every key of R.
  - It is nontransitively dependent on every key of R.

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an **FD X → 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
- The goal is to have each relation in BCNF (or 3NF)

Boyce-Codd Normal Form

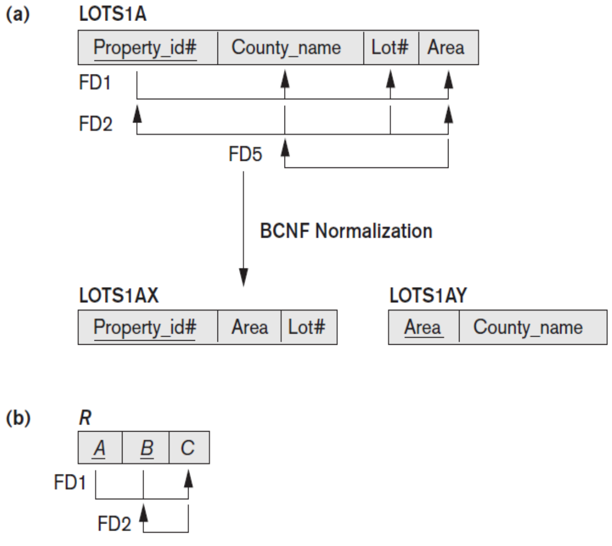


Figure 14.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. $C \rightarrow B$.

A relation in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 14.14
A relation TEACH that is in 3NF but not BCNF.

- A student can take several courses. But cannot take the same course twice.
- A course can be taught by several instructors.
- An instructor teaches only one course.

Achieving BCNF by Decomposition(1)

- 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

BCNF by Decomposition (2)

- 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.
- Three possible decompositions for relation TEACH
  - {student, instructor} and {student, course}
  - {course, instructor } and {course, student}
  - **{instructor, course }** and **{instructor, student}**

Achieving BCNF by Decomposition (3)

- 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.
- Only the 3rd decomposition will not generate spurious tuples after join (and hence has the non-additive property).
- A test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless) will be discussed in the next lecture.

Modeling Temporal Data

- **Temporal data** have an association time interval during which the data are *valid*.
- A **snapshot** is the value of the data at a particular point in time.
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard.
- Adding a temporal component results in FDs like

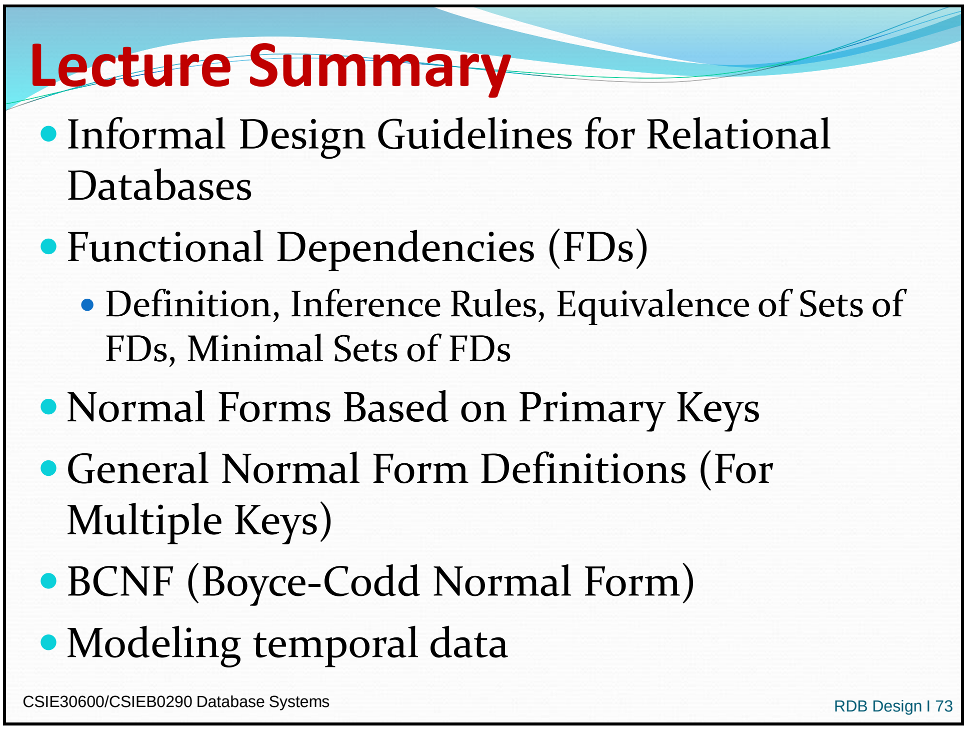
  $ID \rightarrow street, city$

  not holding, because the address varies over time
- A **temporal functional dependency** $X \rightarrow Y$ holds on schema *R* if the functional dependency X → Y holds on all snapshots for all legal instances r(*R*).

Modeling Temporal Data (Cont.)

- In practice, database designers may add start and end time attributes to relations
  - E.g., *course*(*course_id, course_title*) is replaced by

    course(*course_id, course_title, start, end*)
    - **Constraint**: no two tuples can have overlapping valid times
      - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time.
  - E.g., student transcript should refer to course information at the time the course was taken



Lecture Summary

- Informal Design Guidelines for Relational Databases
- Functional Dependencies (FDs)
  - Definition, Inference Rules, Equivalence of Sets of FDs, Minimal Sets of FDs
- Normal Forms Based on Primary Keys
- General Normal Form Definitions (For Multiple Keys)
- BCNF (Boyce-Codd Normal Form)
- Modeling temporal data