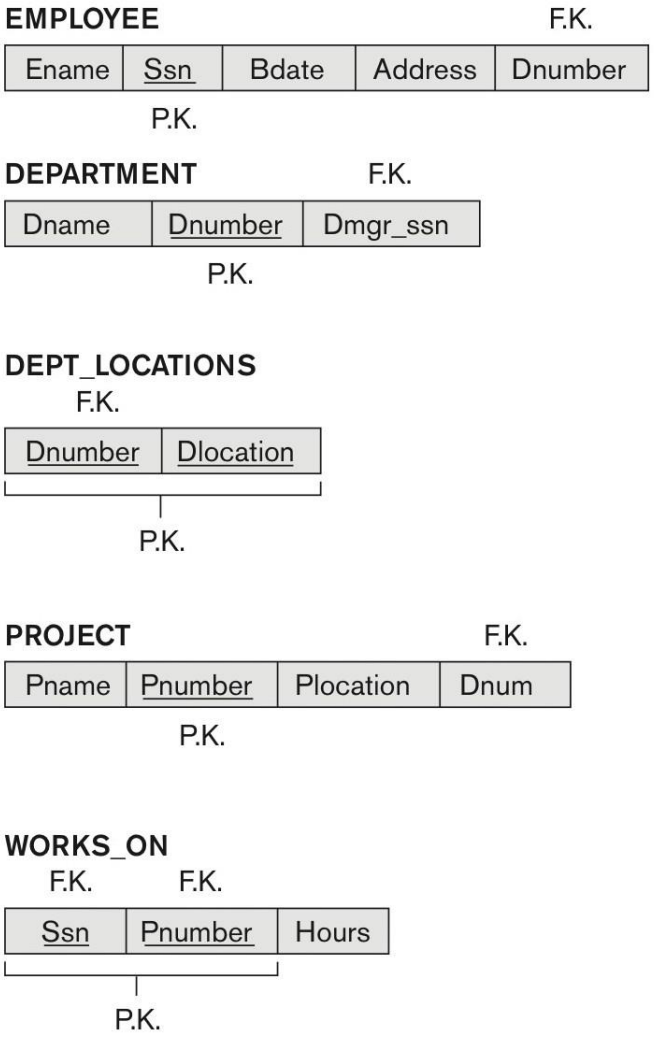
Part 5: Relational Database Design and Normalization

Database System Concepts, 7th Ed.

A Well-Designed COMPANY relational database schema



Other Designs: Update Anomaly

- Consider the following relation involving employees and projects
  - *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 the employees working on project P1

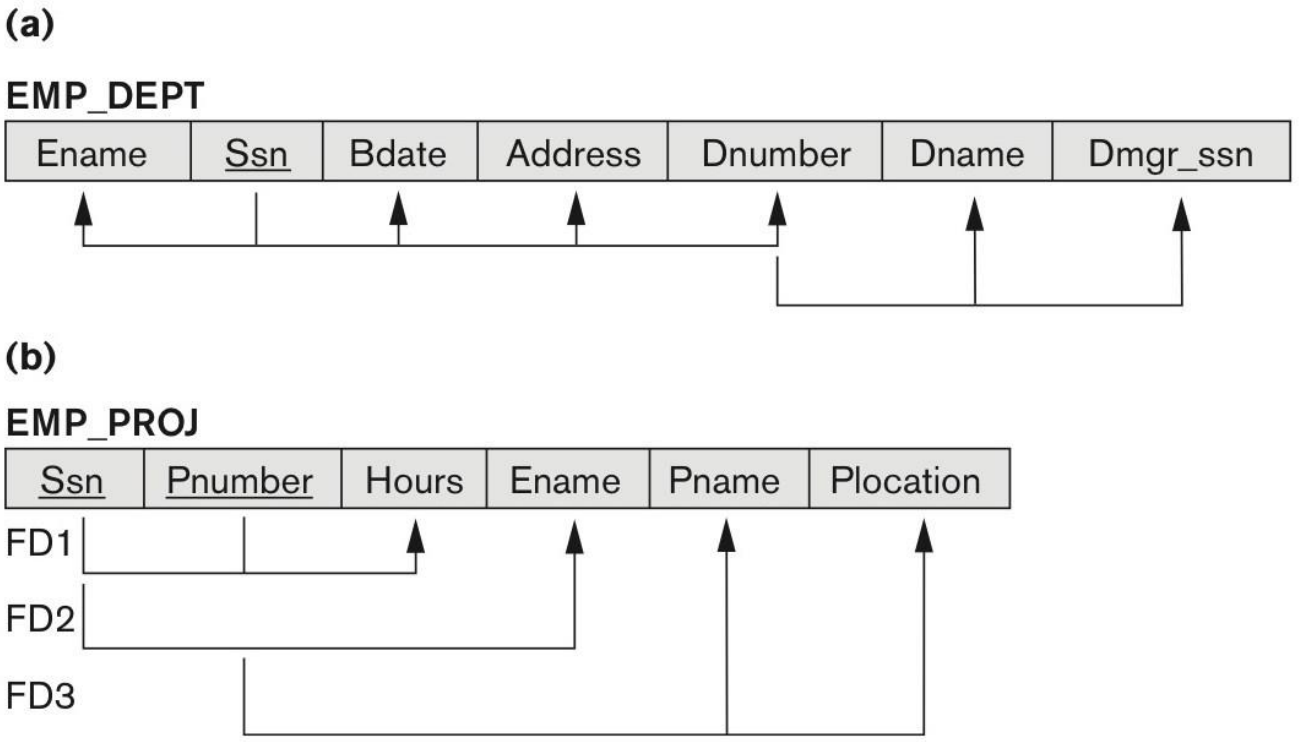
Other Designs: Insertion Anomaly

- *EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)*
- Insert Anomaly:
  - Cannot naturally insert a project unless an employee is assigned to it
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project

Other Designs: Deletion Anomaly

- *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

Two relation schemas suffering from operation anomalies



Sample states for EMP_DEPT and EMP_PROJ

Redundancy

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

Redundancy Redundancy

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

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Example of non-atomic domains:

    Variable number of banking transactions of an account (sometimes called a repeating group)
  - Variable number of locations of a department

- A relational schema *R* is in **first normal form** if the domains of all attributes of *R* are atomic

Normalization 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}

Assume that a Department can have (a variable number of) multiple locations

(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

Normalizing into 1NF

(a)

EMP_PROJ

		Projs	
Ssn	Ename	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

Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key

Another Example: University Database

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

Decomposition

- To avoid the repetition-of-information problem in the *in_dep* schema, decompose it into two schemas – instructor and *department* schemas
- Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary)

  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

  The problem arises when we have two employees with the same name
- We can also lose information -- we cannot reconstruct the original *employee* relation, resulting in a **lossy decomposition**

A Lossy Decomposition

ID	name	street	city	salary
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000

employee

ID	name
57766	Kim
98776	Kim

name	street	city	salary
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000

natural join

ID	name	street	city	salary
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000

ID 57766 has 2 addresses

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R That is $R = R_1 \cup R_2$
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \cup R_2$
- Formally,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$:

  $R_1 = (A, B)$ $\quad$ $R_2 = (B, C)$

A	*B*	*C*
α	1	A
β	2	B

r

A	*B*
α	1
β	2

$\Pi_{R_1}(r)$

B	*C*
1	A
2	B

$\Pi_{R_2}(r)$

A	*B*	*C*
α	1	A
β	2	B

$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

Normalization Theory

- Decide whether a particular relation *R* is in "good" form
- In the case that a relation *R* is not in "good" form, decompose it into set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Functional Dependencies

- 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 associated with only one department
  - Each department has only one value for its budget, and only one associated building

Functional Dependencies

- 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
- 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*

Functional Dependencies Definition

- Let R be a relation schema

$$\alpha \subseteq R \quad and \quad \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A,B)$ with the following instance of r.

A	B
1	4
1	5
3	7

On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*
- We denote the *closure* of *F* by F^+

Keys and Functional Dependencies

- 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 $\alpha \subset K$, $\alpha \rightarrow R$
  - One of the candidate keys is designated to be the **primary key,** and the others can be called **secondary keys**
- In general, any data field other than the primary key can be called a **secondary key** (particularly for search and indexing purposes)
  - A secondary key may or may not uniquely identify a tuple (i.e., may or may not be a superkey)

Keys and Functional Dependencies

- A **prime attribute** is a member of *some* candidate key
  - If there is only one candidate key – the primary key – then a **prime attribute** is an attribute that is member of the primary key
- A **nonprime attribute** is not a prime attribute — that is, it is not a member of any candidate key
  - If there is only one candidate key – the primary key – then a **nonprime attribute** is an attribute that is not a member of the primary key

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.
- Consider the schema:

  in_dep (*ID*, *name*, *salary*, *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:

  $$dept\_name \rightarrow salary$$

  thus *dept_name* is not a superkey

Use of Functional Dependencies

- We use functional dependencies:
  - To test relations to see if they are legal under a given set of functional dependencies
    - If a relation *r* is legal under a set *F* of functional dependencies, 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
    name $\rightarrow$ *ID*

Trivial Functional Dependencies

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- *Note:*
  - $B \rightarrow BC$

    is a shorthand notation for
  - $B \rightarrow \{B, C\}$

Second Normal Form

- Definitions
  - **Prime attribute:** An attribute that is a member of the primary key K (assuming it is the only candidate key)
  - **Full functional dependency:** a FD $Y \rightarrow Z$ where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {Emp#, Proj#} $\rightarrow$ No_hours is a full FD since neither Emp# $\rightarrow$ No_hours nor Proj# $\rightarrow$ No_hours hold
  - {Emp#, Proj#} $\rightarrow$ Ename is not a full FD (it is called a partial dependency) since Emp# $\rightarrow$ Ename also holds

Second Normal Form

- A relation schema R is in **second normal form (2NF)** if it is in first normal form, and every nonprime attribute A in R is fully functionally dependent on the primary key (assuming it is the only candidate key)
- Consider

  EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

  where *Ename* does not fully depend on the primary key, and

  Pname does not fully depend on the primary key
- Decompose it into

  EMP(Emp#, Ename)

  PROJ(Proj#, Pname)

  EP(Emp#, Proj#, No_hours)

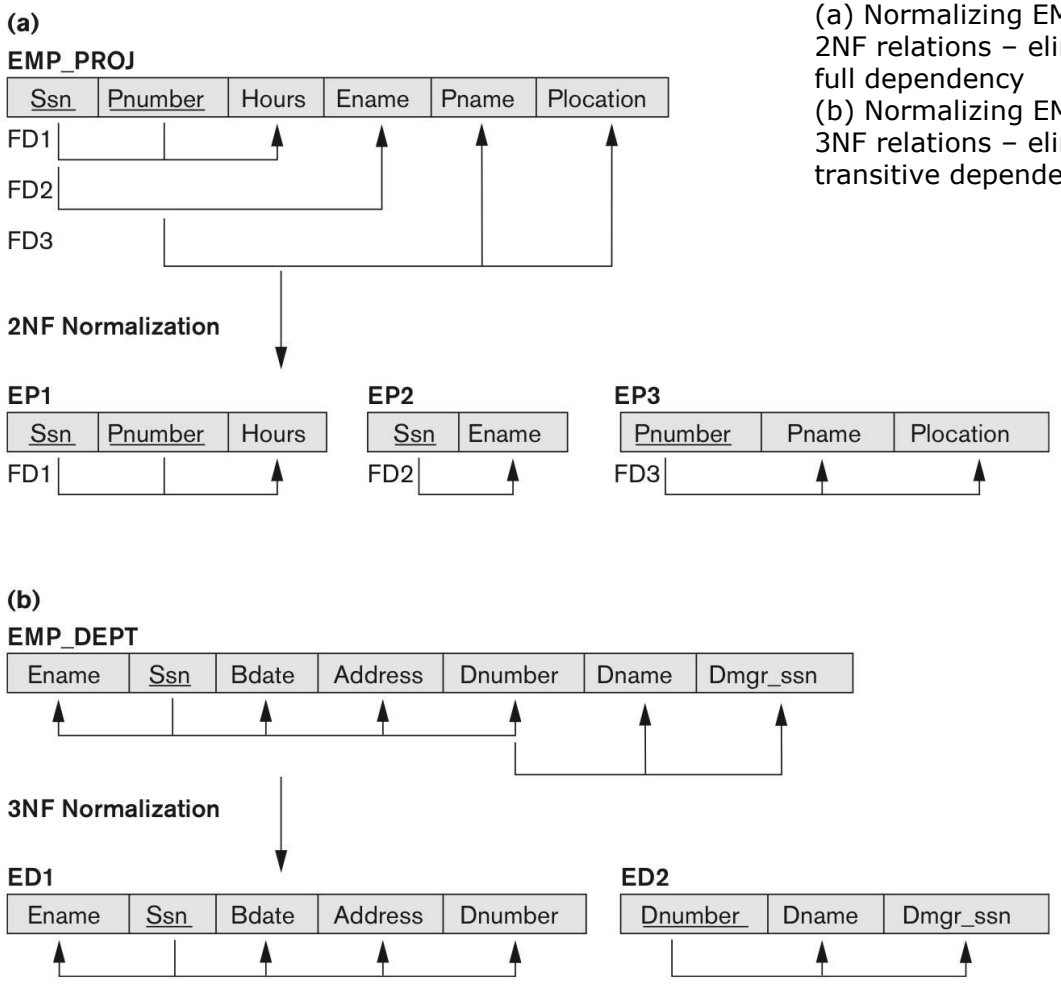
Third Normal Form

- Definition:
  - **Transitive functional dependency:** a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$, where Y is a nonprime attribute
- Examples:
  - Ssn $\rightarrow$ Dmgr_ssn is a **transitive** FD
    - Since Ssn $\rightarrow$ Dnumber and Dnumber $\rightarrow$ Dmgr_ssn hold
    - Transitive dependence is undesirable because consider the relation (Ssn, Dnumber, Dmgr_ssn)
      - if the manager of a department changes, then this information needs to be updated for everyone in that department involving many tuples
  - Ssn $\rightarrow$ Ename is **non-transitive**
    - Since there is no set of attributes X where Ssn $\rightarrow$ X and X $\rightarrow$ Ename

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no nonprime attribute A in R is transitively dependent on the primary key (assuming it is the only candidate 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
    - When the Salary of Emp# changes, then only one tuple needs to be changed

Normalizing into 2NF and 3NF



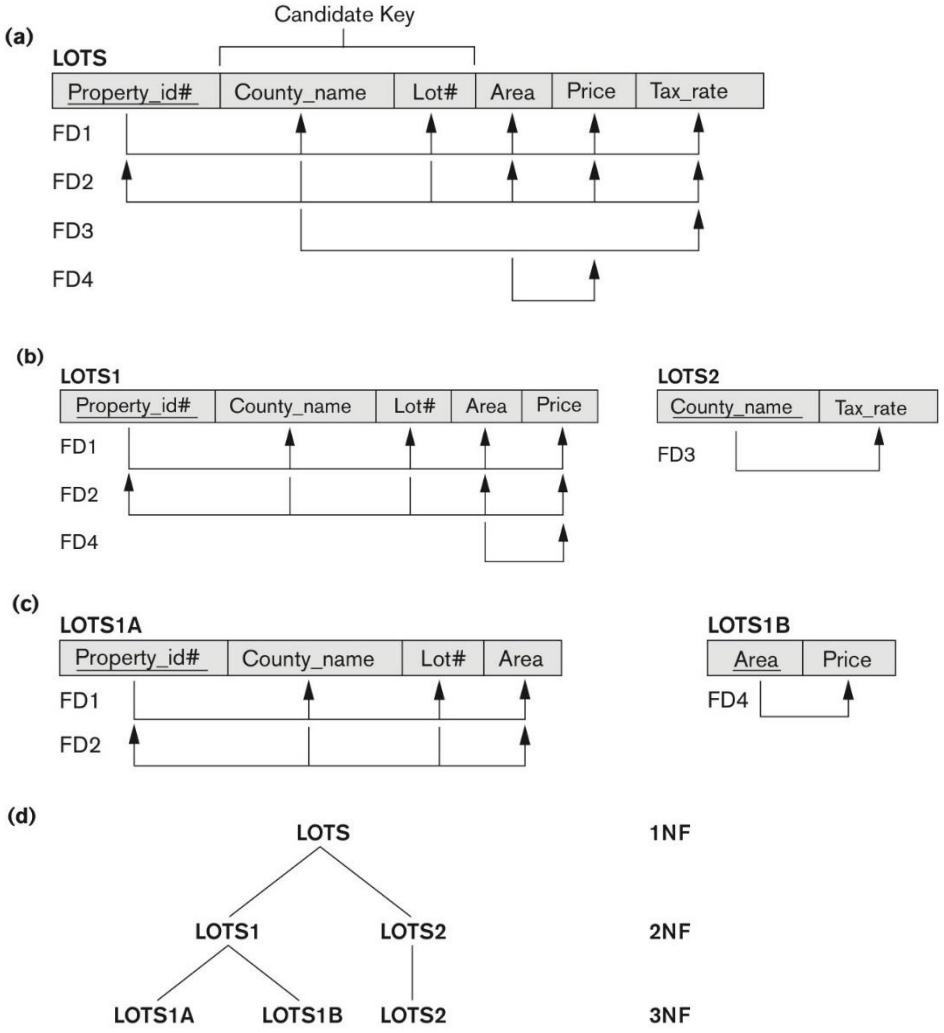
Normalizing into 2NF and 3NF.
(a) Normalizing EMP_PROJ into 2NF relations – eliminate non-full dependency
(b) Normalizing EMP_DEPT into 3NF relations – eliminate transitive dependency

Normalizing into 2NF and 3NF

(a) The LOTS relation with its functional dependencies FD1 through FD4.

(b) Since Tax_rate depends only partially on the candidate key (i.e. only on County_name), we decompose LOTS into the 2NF relations LOTS1 and LOTS2

(c) Since Price is a transitively dependent on the primary key through Area in LOTS1, we decompose LOTS1 into the 3NF relations LOTS1A and LOTS1B.



Normal Forms Defined Informally for Single Candidate Key

- 1st normal form
  - All attributes depend on **the key (i.e., primary key)**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

General Normal Form Definitions for Multiple Candidate Keys

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- Recall: any attribute that is a subset of a candidate key is a *prime attribute*
  - All other attributes are called *nonprime attributes*

General Normal Form Definitions for Multiple Candidate Keys

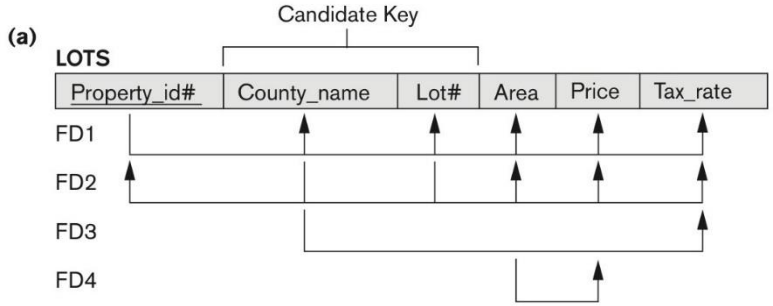
- A relation schema R is in **second normal form (2NF)** if it is in 1NF, and every nonprime attribute A in R is fully functionally dependent on *every* candidate key of R
- In the Property Database the FD

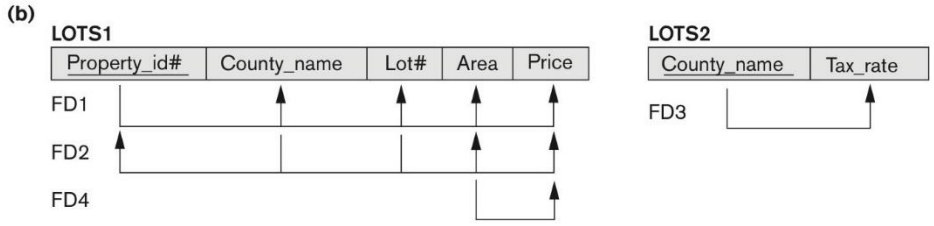
  County_name → Tax_rate violates 2NF, because it is a partial dependence on the candidate key County_name+Lot#

- So second normalization converts LOTS into

  LOTS1 (Property_id#, County_name, Lot#, Area, Price)

  LOTS2 (County_name, Tax_rate)





General Definition of Third Normal Form

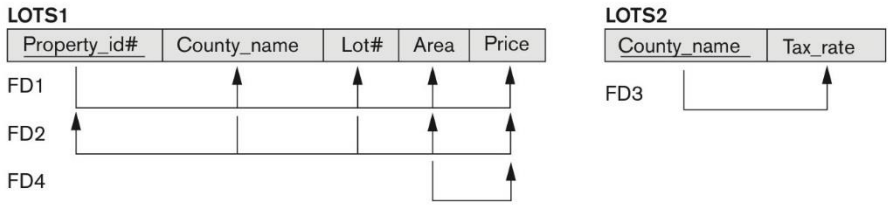
- Definition:
  - **Superkey** of relation schema R - a set of attributes S of R that contains a candidate key of R
  - A relation schema R is in **third normal form (3NF)** if whenever a nontrivial functional dependency X → A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R (more precisely, A-X is a prime attribute of R)
- LOTS1 relation violates 3NF because in

  Area → Price

  Area is not a superkey in LOTS1

  and Price is not a prime attribute in LOTS1

(b)



Interpreting the General Definition of Third Normal Form

- Consider the 2 conditions in the definition of 3NF:

  A relation schema R is in **third normal form (3NF)** if whenever a *nontrivial* functional dependency X → A holds in R, then either:

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

- A relation schema that violates 3NF has to violate **both** conditions (a) and (b)
- Violating (b) means A is a nonprime attribute
- Violating (a) means that X is not a superset of any candidate key of R; thus X can be either prime or nonprime:
  - X prime ⇒ X is a proper subset of a candidate key of R (it cannot be an improper subset because it is not a superkey)

    ⇒ X → A is a partial dependency on the candidate key
  - X nonprime (i.e. not a proper subset of any candidate key), in which case we have a nonprime attribute X determining another nonprime attribute A (violation of (b))

    ⇒ X → A is part of a transitive dependency

    i.e., candidate key determining X (since candidate key determines everything), and X determining A

Interpreting the General Definition of Third Normal Form

- Thus, violating 3NF from the above definition means either
  - having a transitive dependency (TD), or
  - having partial dependence on a candidate key (PD)
- That is

$$\sim \text{3NF} \Rightarrow \text{TD} \vee \text{PD}$$

which is logically equivalent to $\sim(\text{TD} \vee \text{PD}) \Rightarrow \text{3NF}$ (contrapositive)

which is logically equivalent to $\sim \text{TD} \wedge \sim \text{PD} \Rightarrow \text{3NF}$ (De Morgan's Law)

- This leads to an alternative definition of 3NF

Alternative Definition of Third Normal Form

- **We can restate the definition as:**

  A relation schema R is in **third normal form (3NF)** if every nonprime attribute in R meets both of these conditions:

  - It is fully functionally dependent on every candidate key of R
  - It is non-transitively dependent on every candidate key of R

  Note that stated this way, a relation in 3NF also meets the requirements for 2NF

- 3NF will be revisited in relation to the Boyce-Codd normal form (BCNF)

Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
- It is useful to design the database in a way that constraints can be tested efficiently
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low
- When decomposing a relation, it is possible that the testing cannot be done without having to perform a Cartesian Product or Join
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**

Dependency Preservation Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With functional dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*
- However, any decomposition that does not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$

  will not be dependency preserving, since this functional dependency can only be checked by computing a join

Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

  $$\alpha \rightarrow \beta$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
- If we call the left hand side of an FD a determinant, BCNF roughly says that every determinant is a superkey

Boyce-Codd Normal Form

- Example schema that is ***not*** in BCNF:

  in_dep (*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)

  because :

  - *dept_name* $\rightarrow$ *building*, *budget*
    - holds in *in_dep*
    - but *dept_name* is not a superkey
- When decompose *in_dep* into *instructor* and *department*:

  instructor (*ID*, *name*, *salary*, *dept_name*)

  department (*dept_name*, *building*, *budget*)

  - *instructor* is in BCNF
  - *department* is in BCNF

Decomposing a Schema into BCNF

- Let *R* be a schema *R* that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF, i.e. α is not a superkey of *R*.
- We decompose *R* into:
  - (α U β)
  - (*R* - (β - α))
- In our example of *in_dep* (*ID, name, salary, dept_name, building, budget*)
  - *dept_name* $\rightarrow$ *building, budget*
  - α = *dept_name*
  - β = *building, budget*

  and *in_dep* is replaced by (note that β - α = β here)
  - (α U β) = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{B\} \quad \text{and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider the schema:

  dept_advisor(s_ID, i_ID, dept_name)
- With functional dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$

  A student has only one advisor in a given department
- *dept_advisor* is not in BCNF
  - *i_ID* is not a superkey
- Any decomposition of *dept_advisor* will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$

  (if it includes all the attributes, then it is not a decomposition)
- Using the first FD to decompose, we have

  (*i_ID*, *dept_name*)

  (*s_ID, i_ID*)

  which is NOT dependency preserving

Third Normal Form Revisited

- A relation schema R is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

    (**NOTE**: the third condition does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; each attribute A in $\beta - \alpha$ may be contained in a *different* candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (as will be seen later)
- For example, a relation that is in 3NF but not in BCNF requires that for any nontrivial $\alpha \rightarrow \beta$, α is not a superkey but β is a prime attribute

3NF Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)
- With functional dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF (because *i_ID* is not a superkey)
- *R,* however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$ *and* *i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* } and
    - *dept_name* is contained in a candidate key

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- What is wrong with the table?
  - Repetition of information (the association between L and K is repeated several times)
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- Advantages of 3NF over BCNF
  - It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation
- Disadvantages of 3NF
  - May need to use null values to represent some of the possible meaningful relationships among data items
  - Repetition of information

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in "good" form
- In the case that a relation scheme R is not in "good" form, need to decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that:
  - Each relation scheme is in good form
  - The decomposition is a lossless decomposition
  - Preferably, the decomposition should be dependency preserving

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

  inst_info (ID, child_name, phone)

  - where an instructor may have more than one phone and can have multiple children
  - different instructors can have children with the same name, and a phone number can be shared by multiple instructors
  - Instance of *inst_info*

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

How good is BCNF?

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

Higher Normal Forms

- It is better to decompose *inst_info* into:
  - *inst_child*:

ID	*child_name*
99999	David
99999	William

  - *inst_phone:*

ID	*phone*
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall consider later