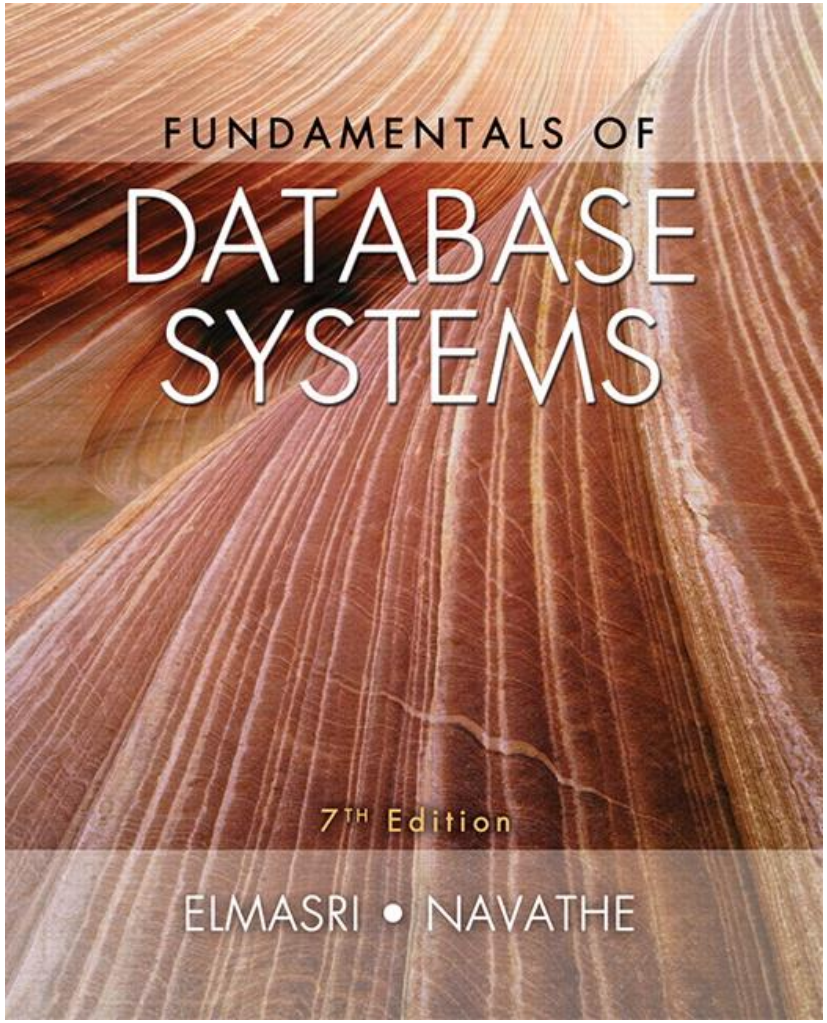


CHAPTER 14

Basics of Functional Dependencies and Normalization for Relational Databases

Chapter Outline

- 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

Chapter Outline

- 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

3,4 & 5 covered some other time

- 4 General Normal Form Definitions for 2NF and 3NF (For Multiple Candidate Keys)
- 5 BCNF (Boyce-Codd Normal Form)
- 6 Multivalued Dependency and Fourth Normal Form
- 7 Join Dependencies and Fifth Normal Form

6 & 7 not covered in this course

PLAN FOR LECTURES #10,11,12

GOALS:

- Introduce Relational Database Design as a practical activity followed in large organizations worldwide. Relational model dominates the commercial market.
- Initially, provide informal guidelines that can point put problems with relational design
- Give the theoretical basis for analyzing what are good vs. poor designs.
- Introduce the tool for analyzing designs called functional dependencies
- Introduce the process of normalization as it was proposed to “improve” or “purify” poor designs
- Introduce formal basis for synthesizing good relations strictly based on the knowledge of dependencies among attributes

PLAN FOR LECTURE #10

- Cover Sections 14.1 and 14.2 from Chapter 14
  - 14.1 Informal Design Guidelines for Relational Schemas
  - 14.2 Functional Dependencies
- Cover Section 15.1 from Chapter 15
  - 15.1 Further Topics in Functional Dependencies

1. Informal Design Guidelines for Relational Databases (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?

1. Informal Design Guidelines for Relational Databases (2)

- CRITERIA FOR "GOOD" DESIGN:
  - minimality : should express the information with minimum number of distinct relations
  - lack of redundancy: should minimize amount of redundancy among relations
  - Information preservation: should preserve all information captured by the conceptual design (in terms of entity types, relationship types, attributes, and constraints)
  - consistency: among the relations
  - efficiency: BEYOND our scope. Typically addressed in the physical design.

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.*

ER DIAGRAM – Company Database

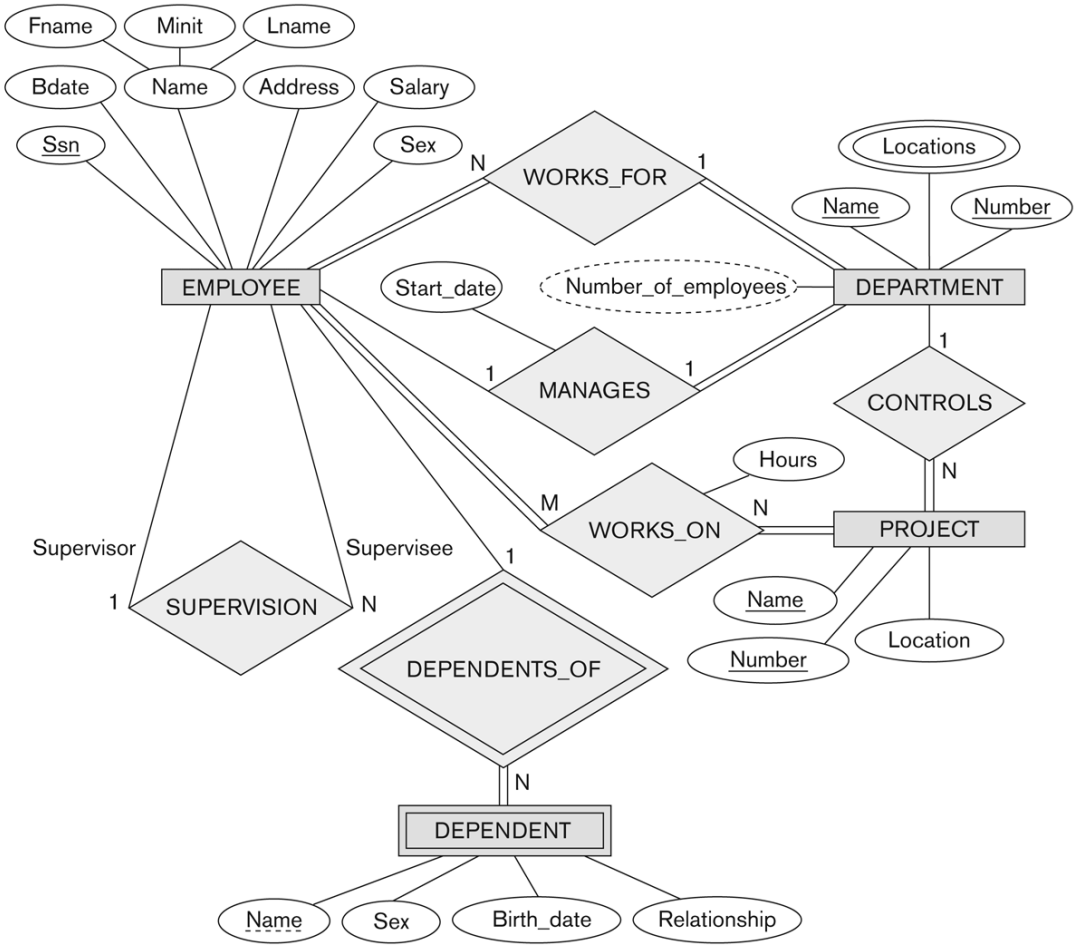


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Figure 14.1 A simplified COMPANY relational database schema

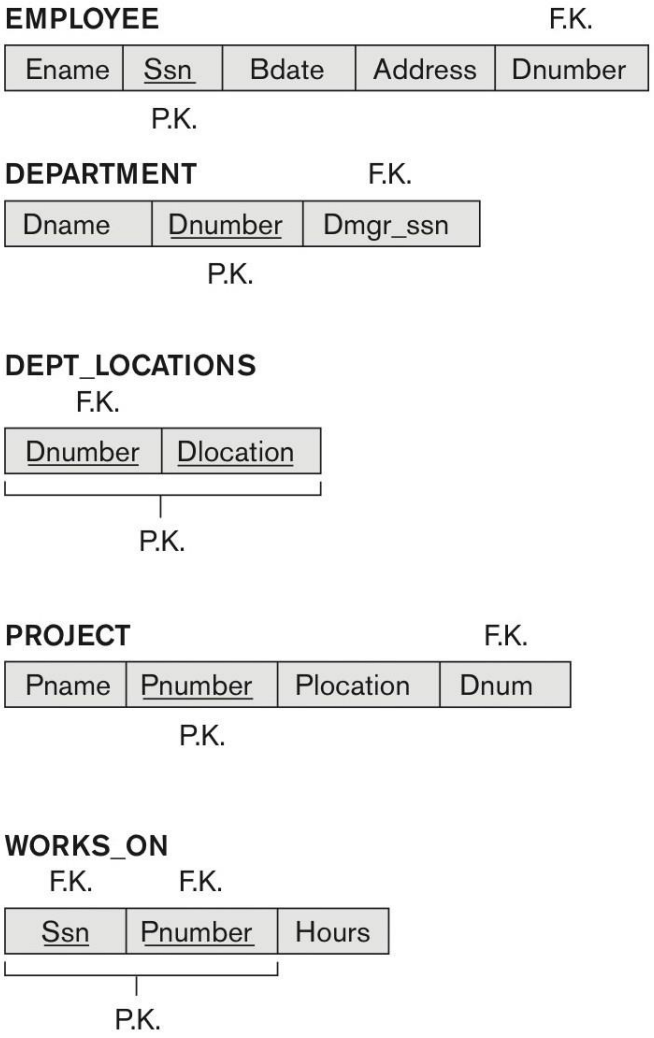


Figure 14.1 A simplified COMPANY relational database schema.

EXAMPLE OF AN UPDATE ANOMALY

- Consider the following relational design:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Assume the designer proposed this based on the Employee works-on Project relationship type.

CREATES A PROBLEM:

- 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.

1.2 Redundant Information in Tuples and Update Anomalies

- When information in ER schema is stored redundantly in relations- what happens?
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification (or update) anomalies

EXAMPLE OF AN INSERT ANOMALY

- Consider the relation again:
  - 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 same relation again:
  - 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.

REITERATING:

- The design of relation :
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Leads to Insert Anomaly, Delete anomaly and Update/Modification anomaly.
- Conclusion - it is a bad design and such designs must be avoided.
- Question- how to tell whether a given design is good or bad?
- Answer – we need some tool for analysis. That tool is “functional dependency” and the methodology for “fixing” the design is specified by the process of “Normalization”.

The Correct Design

- If we followed the mapping algorithm (and informal guidelines) correctly and applied to the ER schema, we would have the correct design:

EMPLOYEE(Ssn, Fname, .., Lname,.., Dno)

FK

PROJECT (Pnumber, Pname, Plocation, Dnum)

FK

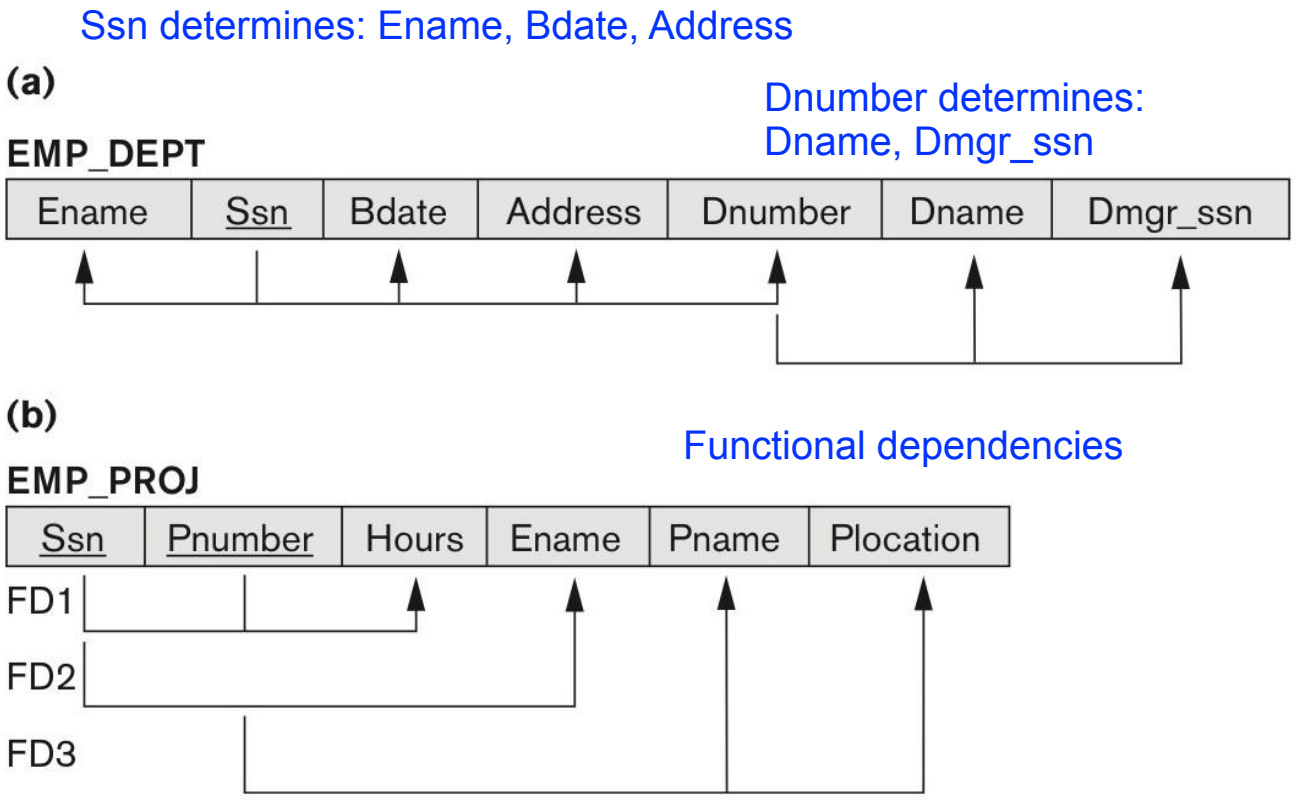
WORKS_ON (Ssn, Pnumber, No-hours)

FK FK

- What did we achieve?
- - Top 2 relations strictly stand for an entity type
- - Third relation strictly stands for a relationship type
- - There is no mixing of entity and relationship information as done in previous example
- NET RESULT: This design DOES NOT suffer from any of the anomalies.

Figure 14.3 Two relation schemas suffering from update anomalies

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



What did the previous slide show?

- It showed “bad designs” which mixed multiple entity types and/or relationship types.
- It showed the dependencies among attributes which we will define as “functional dependencies”
- In relation (a), we mixed information from 2 entity types: EMPLOYEE and DEPARTMENT. That leads to Insert Anomaly, Delete anomaly and Update anomaly. Here, Dname and Dmgr_Ssn are determined from Department number and have nothing to do with SSN. If the department# D5 gets a new manager, the update must be applied to all employees, say 50 – hence 50 updates instead of one!
- In relation (b) we already saw the anomalies and they arose because Employee attributes (Ename) depended on SSN while project attributes like Pname and Plocation depended only on Pnumber. This was a mixing of two entity types and a relationship type.

Figure 14.4 Sample states for EMP_DEPT and 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.

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

Guideline for Redundant Information in Tuples and Update Anomalies

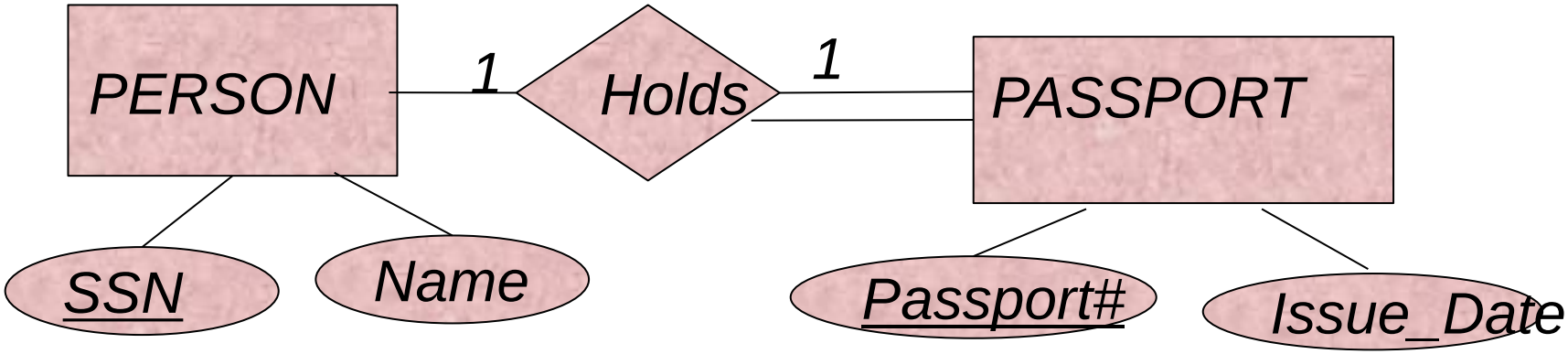
- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

This is like a corollary of Guideline1. It basically states that when you are forced to mix descriptor attributes of an entity type into the table for another entity type or a relationship type (for performance reasons or reporting requirements etc.), you should document them and take care of the consistency preservation via the application.

1.3 Null Values in Tuples

- 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

A one-to-one relationship type



- *Suppose only 60% of persons in the database of 2 million persons hold a Passport*

The design:

- *PERSON (SSN, Name, Passport#, Issue_date) – 2 million tuples*

Will contain 40% of tuples with NULL values for Passport# and Issue_date. Hence, the correct design is:

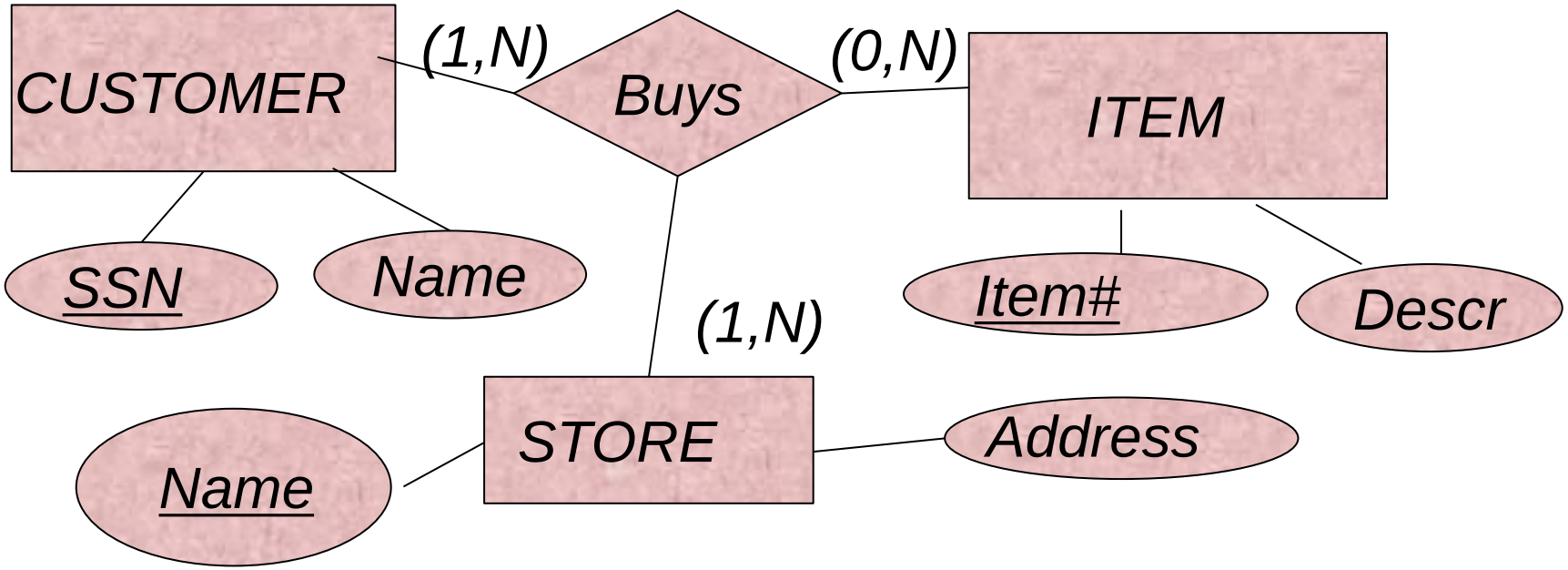
PERSON (SSN, Name) – 2 million tuples

PASSPORT (Passport#, Issue_date, SSN) – 1.2 million tuples. *FK*

1.4 Generation of Spurious Tuples – avoid at any cost

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

1.4 Generation of Spurious Tuples: example of a Ternary Relationship



- *Here, each instance of Buys relationship type relates one Customer, one Item and one Store. Hence it is CORRECT.*
- *A Customer buys at least one but may have bought any number of items: (1,N)*
- *An Item may not have been sold but can be sold in any number of BUYS transactions: (0,N)*

1.4 Generation of Spurious Tuples – contd.

- The right way to map this ternary relationship is to create one relation for “BUYS”:
- BUYS(SSN, Item#, Store_name, ……..)
- Suppose:

<s1 bought x1 from t1>
<s1 bought x2 from t2 >
<s2 bought x1 from t2>
<s2 bought x2 from t1>

- Suppose we mapped this data into two tables:

Cust_Item (Ssn, Item#)

Cust_Store(Ssn, Store_name)

1.4 Generation of Spurious Tuples – contd.

- The two resulting tables:

Cust_Item

SSN	ITEM#
s1	x1
s1	x2
s2	x1
s2	x2

Cust_Store

SSN	STORE_name
s1	t1
s1	t2
s2	t2
s2	t1

Generation of Spurious Tuples – contd.

- Now, suppose we Join the two tables Cust_Item and Cust_Store

*Select **

From Cust_Item x Inner Join Cust_Store t on x.ssn = t.ssn

Ssn	Item#	Store_name	
s1	x1	t1	
s1	x1	t2	BAD
s1	x2	t1	BAD
s1	x2	t2	
s2	x1	t2	
s2	x1	t1	BAD
s2	x2	t2	BAD
s2	x2	t1	

The BAD tuples are spurious (incorrect/invalid) data that resulted from the BAD design

Rules about decomposing relations

- What happened in the last case?
- The ternary relation BUYS (Cust_ssn, Item#, Store#) was wrongly decomposed into 2 relations Cust_Item(Cust_ssn, Item#) and Cust_store(Cust_ssn, Store#)

THIS WAS A BAD DECOMPOSITION

The correct design would :be

BUYS(Cust_ssn, Item#, Store_name, ……..)

- There are two important properties of decompositions:
  a) Non-additive or losslessness of the corresponding join
  b) Preservation of the functional dependencies.

WE VIOLATED THE LOSSLESSNESS (NON-ADDITIVENESS) PROPERTY

FINAL (INFORMAL) DESIGN GUIDELINE

- 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. This should apply to a natural join among any pairs or collections of relations.
- How to know whether a decomposition will be lossless?
- We take the help of functional dependencies and formulate an algorithm (see Algorithm 15.3 in chapter 15 that tests the losslessness property of an n-way decomposition).

2. Functional Dependencies

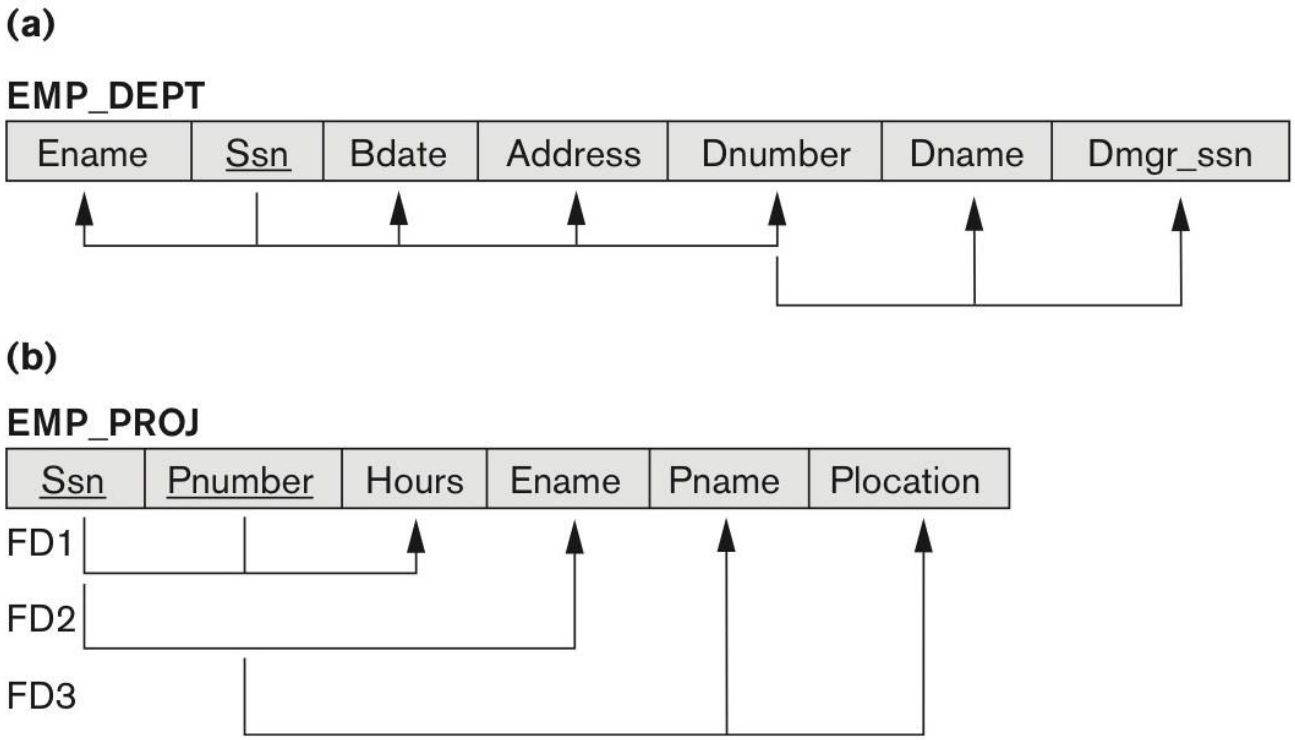
- Functional dependencies (FDs)
  - Are used to specify *formal measures* of the "goodness" of relational designs
  - And keys (derived from FDs) are used to define **normal forms** for relations
  - Are **constraints** that are derived from the *meaning* of and *interrelationships* among the data attributes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

Figure 14.3 – FDs among attributes

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



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)

- 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
  - we never have two distinct tuples with t1[K]=t2[K]) , i.e., the projection of each tuple in a relation on the K column(s) must yield distinct values.

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 (semantic) 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.

Figure 14.7 Ruling Out FDs

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 Couse → Text are ruled out.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.8 What FDs may exist?

- A relation *R*(A, B, C, D) with its extension.
- Which FDs *may exist* in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

NO: A->B? , B-> A , C-> D, D -> C, A -> {B,C}, C-> {B,D}, {B,C} -> A.

YES: {A,B} -> C , {A,C} -> D, {A,B} -> {C, D}

3.1 Normalization of Relations (1)

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations by the process of decomposition

- **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)

- 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 Chapter 15)