Chapter 14: Basics of Functional Dependencies and Normalization for Relational Databases

Introduction

- So far we have assumed:
  - **Attributes** are grouped to form a **relation schema** by using the common sense of the **database designer**

- Need some **formal way** of analyzing why one grouping of attributes into a relation schema may be better than another

- In this chapter we discuss some of the **theory** to evaluate relational schemas for design quality

What Are Some Database Schema Better Than Others

- Consider the following a very simple way to design a **database schema**:
  - Put **every attribute** that you need to store into one single (huge) relation.

- Example: the University database schema will become:

Company (SSN, fname, . . . , DNo, DName, . . . , PNo, PName, . . . , DepName, . .)

  - This relation is called the **Universal Relation**

- So what is bad about this relation?

What Makes a Relation "Good" or "Bad"?

- **Four informal guidelines:** measures to determine the quality of relation schema design:
  - Making sure that the semantics of the attributes is clear in the schema
  - Reducing the redundant information in tuples
  - Reducing the NULL values in tuples
  - Disallowing the possibility of generating spurious tuples

Imparting Clear Semantics to Attributes in Relations

- Making sure that the semantics of the attributes is clear in the schema

  - Design a relation schema so it is easy to explain its meaning

  - Do not combine attributes from **multiple entity types** and **relationship types** into a **single relation**.

  - If the relation corresponds to a **mixture of multiple entities**, semantic ambiguities will result, and the relation cannot be easily explained

Imparting Clear Semantics to Attributes in Relations (Example)

EMPLOYEE

F.K.

Ename	Ssn	Bdate	Address	Dnumber

P.K.

PROJECT

F.K.

Pname	Pnumber	Plocation	Dnum

P.K.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation

Redundant Information in Tuples

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				

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

Update Anomalies

- Storing such joins of base relations leads to an additional problem referred to as **update anomalies**

- Database researchers has found a number of bad properties that they call **anomalies**
  - When a relation exhibits one or more of these "anomalies", it is deemed to be "bad"

- There are three types of database anomalies:
  - Insert anomaly
  - Delete anomaly
  - Update anomaly

Insert Anomaly

- See example:

SSN	FName	LName	BDate	DNum	DName	MgrSSN
111-11-1111	John	Smith	Jan-1-78	5	Research	123-45-6789
222-22-2222	Jane	Doe	Apr-1-76	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	May-4-79	1	Payroll	777-77-7777

- If we insert a new tuple for an **employee** who works in department number 5
  - Must enter all the attribute values of department 5 correctly so that they are *consistent* with other tuples

Insert Anomaly (cont.)

- See example:

SSN	FName	LName	B
111-11-1111	John	Smith	Ja
222-22-2222	Jane	Doe	Ap
333-33-3333	Jack	Rabbit	Ma… 79

SSN	FName	LName	BDate	DNum	DName	MgrSSN
111-11-1111	John	Smith	Jan-1-78	5	Research	123-45-6789
222-22-2222	Jane	Doe	Apr-1-76	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	May-4-79	1	Payroll	777-77-7777
NULL	NULL	NULL	NULL	6	HR	NULL

- If we insert a department, say (dnumber=6, dname='Human Resources'), that **does not have any employees**, then we need to use NULL values

Delete Anomaly

- See example:

SSN	FName	LName	BDate	DNum	DName	MgrSSN
111-11-1111	John	Smith	Jan-1-78	5	Research	123-45-6789
222-22-2222	Jane	Doe	Apr-1-76	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	May-4-79	1	**Payroll**	777-77-7777

- If we delete the **last employee** "Jack Rabbit", we will lose information on the department

DELETE employee
WHERE fname = 'Jack' AND lname = '

SSN	FName	LName	BDate	DNum	DName	MgrSSN
111-11-1111	John	Smith	Jan-1-78	5	Research	123-45-6789
222-22-2222	Jane	Doe	Apr-1-76	5	Research	123-45-6789

Delete Anomaly (cont.)

- This delete operation has deleted additional information !!!
    - Information on the **"Payroll" department** is also deleted !!! (its dnumber, its name and its manager) from the database !!!
    - Clearly, that should not be a consequence of a delete command that is routinely used to remove employees....

Update Anomaly

- See example:

SSN	FName	LName	BDate	DNum	DName	MgrSSN
111-11-1111	John	Smith	Jan-1-78	5	Research	888-88-8888
222-22-2222	Jane	Doe	Apr-1-76	5	Research	888-88-8888
333-33-3333	Jack	Rabbit	May-4-79	1	Payroll	777-77-7777

SSN	FName					
111-11-1111	John					
222-22-2222	Jane	Doe	Apr-1-76	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	May-4-79	1	Payroll	777-77-7777

- If we change the manager of the Research department to 888-88-8888
- The update **logically** involved changing ONE item of information (namely change 123-45-6789 to 888-88-8888), but the update operation has modified **multiple tuples** in Employee

Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Today: develop theory to understand why this design may be better **and** how to find this *decomposition*…

Design Guideline 1

- Design guideline 1:
  - Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.
  - **If** any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly

Design Guideline 2

- Design guideline 2:
  - As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL.
  - **If** NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation

Design Guideline 3

- Design guideline 3:
  - Avoid relations that contain **matching attributes** that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Summary and Discussion of Design Guidelines

- We have informally discussed situations that lead to problematic relation schemas
- We proposed **informal guidelines** for a good relational design
- In the rest: Present formal concepts that may be used to define the *goodness* and *badness* of relation schemas more precisely.
  - Discuss Functional dependency as a tool for analysis
  - Specify the three **normal forms**

Introduction to Functional Dependency

- A **formal tool** for analysis of relational schemas
  - Enables us to *detect* and *describe* some of the above-mentioned **problems** in precise terms

Functional Dependencies

- Let *X* and *Y* be 2 **set of attributes** in a relation ***R***

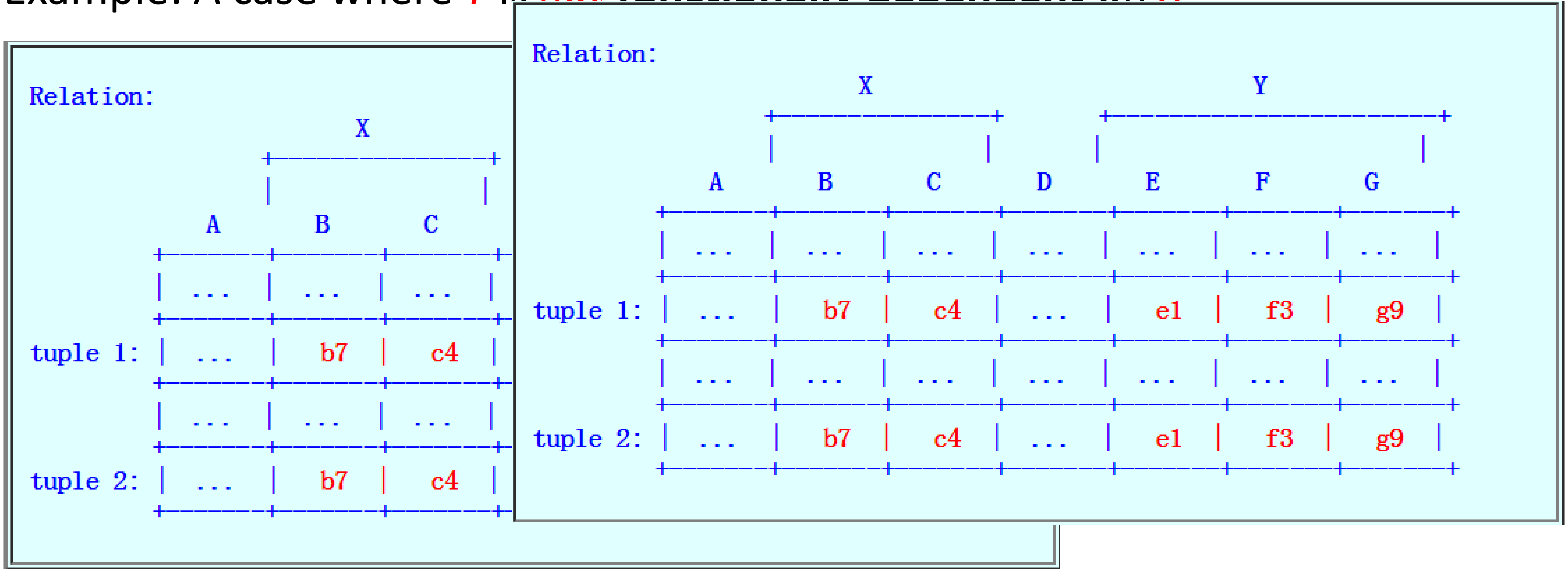
- We say that *Y* is functionally dependent on *X* (or *X* functionally determines *Y* , notation: *X* → *Y*) iff:

X → Y <=> for any two tuples t1 and t2 of the relation R,
if t1[X] = t2[X] (i.e., the attribute values for the X attributes are same in both tuples)
then t1[Y] = t2[Y]

- See example:

Functional Dependencies (cont.)

- Example: A case where *Y* is *not* **functionally dependent** on *X*



- What you must see if *Y* is **functionally dependent** on *X*

Example of Functional Dependencies

- Consider the following relation:
  - That represent information about the **employees** AND the **projects** that they work on

Employee1(SSN, FName, LName, PNumber, PName, Hours)

The key of this relation is: (SSN, PNumber)

- Sample content of the Employee1 relation:

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

Example of Functional Dependencies (cont.)

- Some **functional dependencies** in the Employee1 relation:
  - SSN → fname, lname
  - PNumber → PName
  - SSN, PNumber → Hours
  - But also: SSN, PNumber → fname, lname !!!
  - And also: SSN, PNumber → PName !!!

Normal Forms

- How to use **functional dependencies** to develop a formal methodology for testing and improving relation schemas
- Takes a relation schema through a series of tests to ***certify*** whether it satisfies a certain **normal form**
  - 1NF, 2NF and 3NF, which are based on primary keys

Normal Forms

- 1st Normal Form (1NF) = All tables are flat
- 2nd Normal Form (2NF)
- **Boyce-Codd Normal Form (BCNF)**
- **3rd Normal Form (3NF)**

DB designs based on *functional dependencies*, intended to prevent data ***anomalies***

- *4th and 5th Normal Forms = see text books*

The First Normal Form (1NF)

- The 1NF is the simplest and the **only one** that does not depend on the notion of "**functional dependency**“

- A relation is in 1NF (first normal form) if:
  - Every attribute of the relation has atomic (single, not multi) values

1^{st} Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1^{st} NF

1NF Constraint: Types must be atomic!

Full Functional Dependency

- $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the **dependency** does not hold anymore
- That is, for any attribute **A ε X**, **(X – {A})** does not functionally determine Y.

- Example: {Ssn, Pnumber} → Hours is a full dependency

  {Ssn, Pnumber} → Ename is partial dependency

- 2NF is based on the concept of **full functional dependency**

The Second Normal Form (2NF)

- A relation is in **2NF** iff:
  - **Relation** is in 1NF (i.e., every attribute is atomic), and
  - **Every non-key attribute** is fully functionally determined by **every key** of the relation

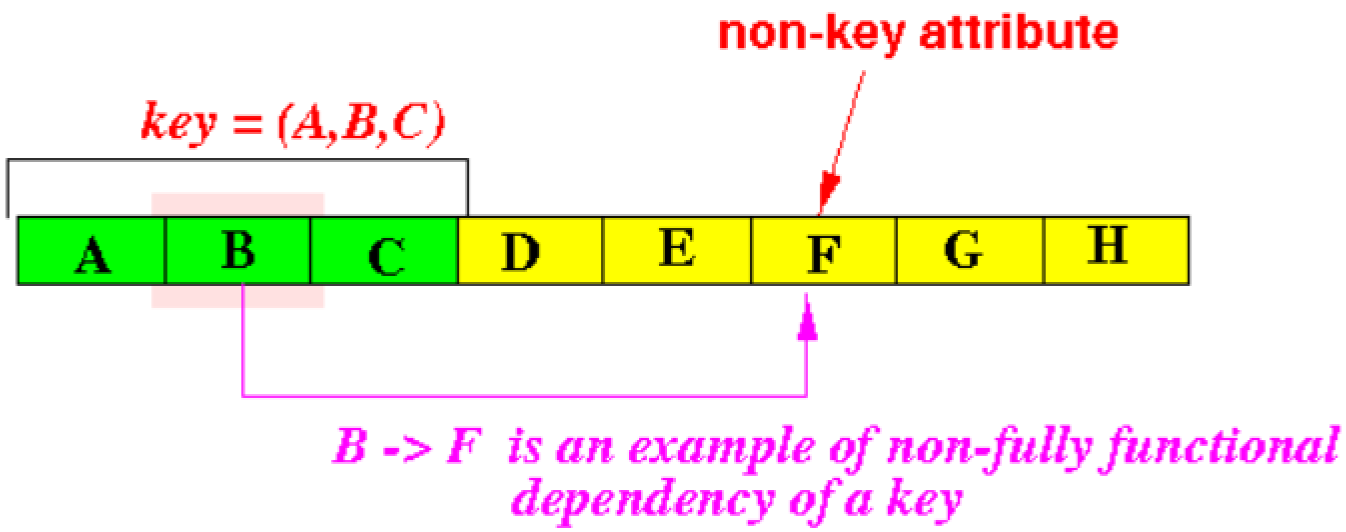
$$\forall\ N \notin KEY:\ K \rightarrow N \quad \text{and} \quad K\text{-}A \text{ does not} \rightarrow N \quad (K \text{ is a key})$$

  - In other words

    - You must **not** have a **non-key** attribute that is **functionally determined** by ***only*** **a (proper) subset** of a key

The Second Normal Form (2NF) (cont.)

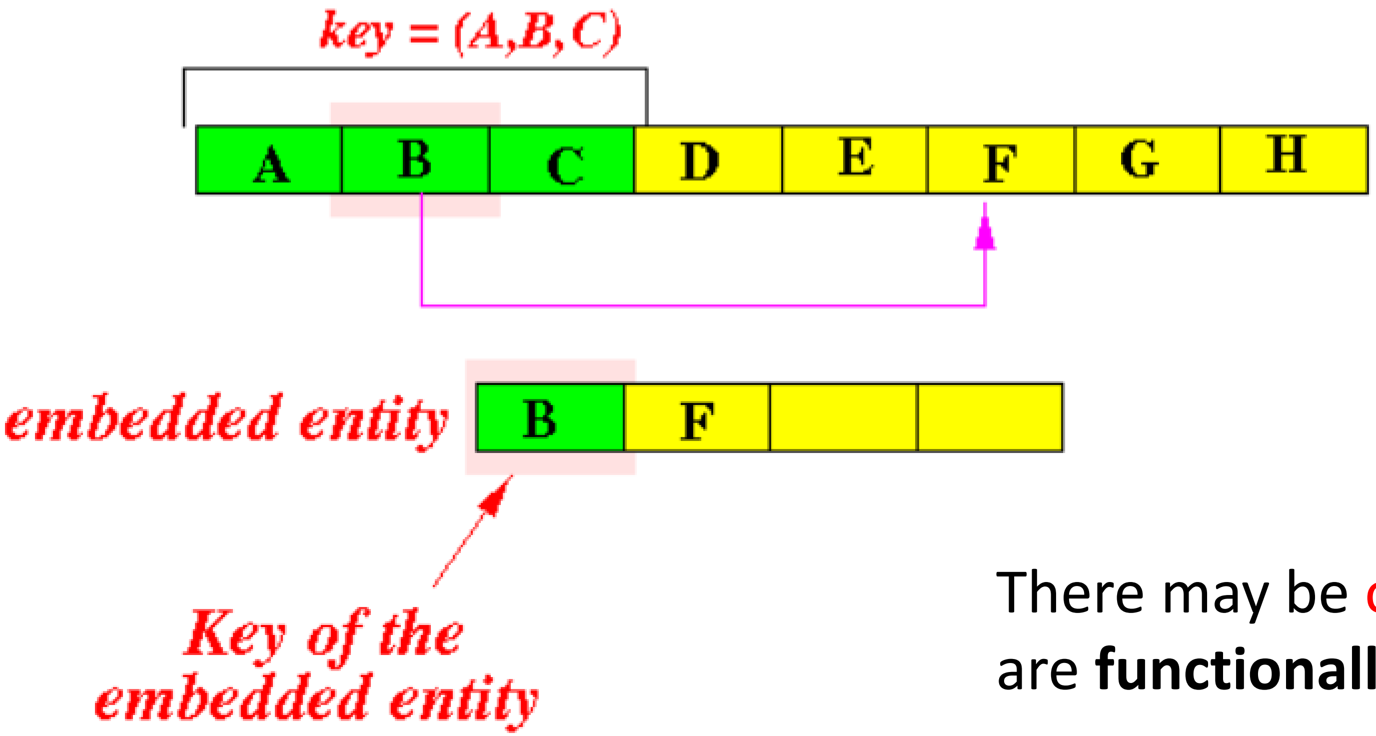
- How to spot the fact that a **relation** violates the **2NF**
  - Find a **non-key attribute** that is functionally determined by a subset of some key
  - Graphically:



This will make the relation violates the 2NF criteria.

Meaning of the 2NF

- The 2NF deals with embedded entities:
  - A **relation** that violates the **2NF** contains another embedded autonomous entity



There may be other attributes (besides F) that are **functionally dependent** on **B** !!!

Example of a Violation of the 2NF

- Consider the following relation:

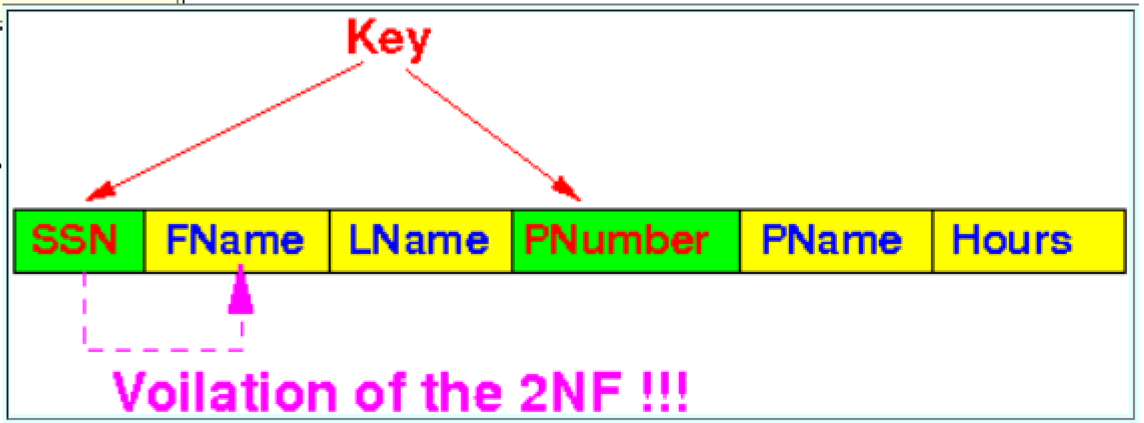
Employee1(SSN, FName, LName, PNumber, PName, Hours)

Functional Dependencies:

SSN → FName, LName
PNumber → PName
SSN, PNumber → Hours

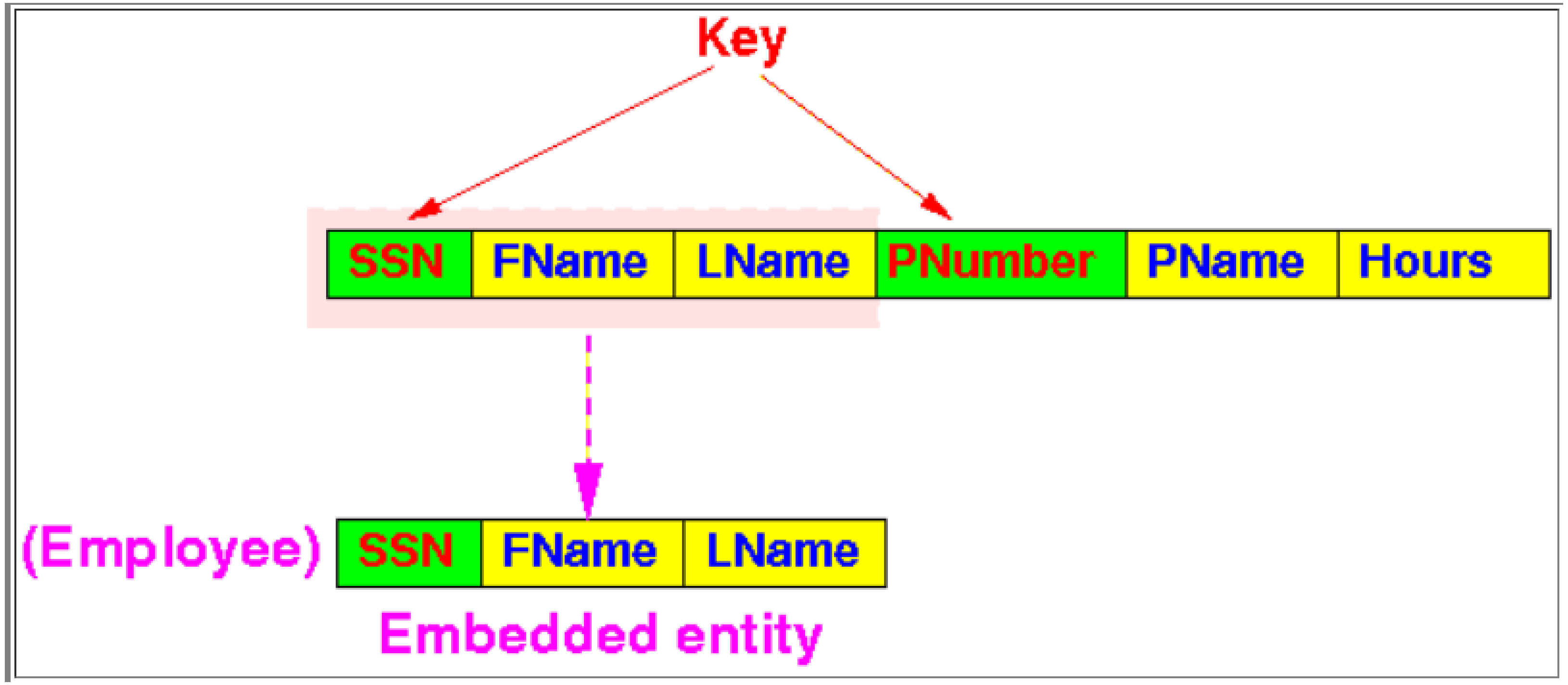
Keys: (SSN, PNumber)

- Employee1 **is not** in **2NF** because of



Example of a Violation of the 2NF (cont.)

- The **entity** that is embedded inside this **relation** is Employee:



How to Solve Normal Form Violations : Decomposition

- How can we remove the **violation** of a **normal form criteria**
- **Answer**: Decompose (= break up) a **relation** into two or more relations
- Example:

Employee1(SSN, FName, LName, PNumber, PName, Hours)

SSN → FName causes 2NF violation

A decomposition:

R1(PNumber, PName, Hours)

R2(SSN, LName, FName)

Good and Bad Decomposition

- We need to learn more about the effect of **breaking a relation** into **2 or more relations**
- Fact:
  - There are good and bad **decomposition**
- Note
  - The **decomposition** suggested by the previous example was a bad decomposition

Decomposition and Its Effect on a Relation

- A **decomposition** of a relation **R** is :
  - There are good and bad **decomposition**

- **Decomposition** = a **collection** of **relations** ***R1, R2, ..., Rn***, such that:

1. ***Each*** **attribute** in ***R1, R2, ..., Rn*** is **found** in ***R*** and

2. ***Every*** **attribute** of ***R*** appears in ***R1, R2, ..., Rn at least*** **once**

- In other words:
  - A **decomposition** will not lose any attributes from the original relation

Decomposition and Its Effect on a Relation (cont.)

- There are "good" and "bad" **decompositions**

- We will soon see that

  - A **"bad" decompositions** will **introduce *extraneous* information** into the **database** !!!

  - A **"good" decompositions** will ***preserve*** the **content** of the ***original* relation**

The Effect of a Decomposition

- Consider the following content of the **relation** Employee1:

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

- The content of the relation Employee1 convey the following **facts**:
  - There are 2 employees
    - **(111-11-1111, John, Smith)**
    - **(111-22-3333, Jane, Doe)**
  - There are 2 projects
    - **(pj1, DBApplet)**
    - **(pj2, WebServer)**
  - There are 3 "works_on" information items:
    - **(111-11-1111, pj1, 20)**,
    - **(111-11-1111, pj2, 10)**
    - **(111-22-3333, pj1, 5)**

The Effect of a Decomposition (cont.)

- Go back to our previous example:

> A decomposition for Employee1(SSN, FName, LName, PNumber, PName, Hours)
> R1(SSN, LName, FName)
> R2(PNumber, PName, Hours)

- Now populate **R1** and **R2** using the data in **Employee1:**

$R1 = \pi_{SSN,\ FName,\ LName}(Employee1)$

$R2 = \pi$ [illegible] $(Employee1)$

- Therefor[illegible] R2 will be:

R1

SSN	FName	LName
111-11-1111	John	Smith
111-22-3333	Jane	Doe

R2

PNumber	PName	Hours
pj1	DBApplet	20
pj2	WebServer	10
pj1	DBApplet	5

The Effect of a Decomposition (cont.)

- Note:
  - After the decomposition of **Employee1** into **R1** and **R2**, the relation Employee1 is deleted !!!
- An important question to ask is:
  - Can we obtain the **same information** stored in **Employee1** from the relations **R1** and **R2** ?
  - (If not, we will be in deep trouble)...

Reconstructing the Original Content of a Decomposed Relation

- The **reconstruction algorithm** used is as follows:

```
if ( R1 ∩ R2 != Ø )
  {
    reconstruction = R1 ⋈ R2   // Join
  }
  else
  {
    reconstruction = R1 × R2   // Cartesian product
  }
```

Reconstructing the Original Content of a Decomposed Relation (Example)

R1

SSN	FName	LName
111-11-1111	John	Smith
111-22-3333	Jane	Doe

×

R2

PNumber	PName	Hours
pj1	DBApplet	20
pj2	WebServer	10
pj1	DBApplet	5

=

Reconstruction:

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-11-1111	John	Smith	pj1	DBApplet	5
111-22-3333	Jane	Doe	pj1	DBApplet	20
111-22-3333	Jane	Doe	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

- We performed a **cartesian product R1 × R2**) on **R1** and **R2** because they do **not** have any attributes **in common** !

Reconstructing the Original Content of a Decomposed Relation (Example) (cont.)

- Compare the content of the **reconstruction** to the content of the original Employee1 relation:

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

Reconstruction:

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-11-1111	John	Smith	pj1	DBApplet	5
111-22-3333	Jane	Doe	pj1	DBApplet	20
111-22-3333	Jane	Doe	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

- The **good:**

  - We are able to **obtain** ***every* tuple** that was in the **original relation** :

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

- The **bad:**

  - There are ***extraneous* tuples** that wer **not present** in the ***original* relation**

Reconstruction: (**extraneous** tuples in **red**)

SSN	FName	LName	PNumber	PName	Hours
111-11-1111	John	Smith	pj1	DBApplet	20
111-11-1111	John	Smith	pj2	WebServer	10
111-11-1111	John	Smith	pj1	DBApplet	5
111-22-3333	Jane	Doe	pj1	DBApplet	20
111-22-3333	Jane	Doe	pj2	WebServer	10
111-22-3333	Jane	Doe	pj1	DBApplet	5

Requirements of a decomposition

- After decomposing a **relation** R:
  - We must be able to obtain **all tuples** in the **original relation R** using the reconstruction algorithm
  - We must not obtain **extraneous tuples** that were not present in the **original relation R** using the reconstruction algorithm

Requirements of a decomposition (cont.)

- If we miss some **tuples** in the reconstruction, it means that:
  - We have lose information
  - **Clearly** that is unacceptable!!!
- If we gain some (extraneous) tuple in the reconstruction, it means that:
  - We have some invalid information in the relation (= database) !!!
  - That is **also** inacceptable !!!