Chapter 7: Functional Dependences and Normalization

Outline:

7.1 Problems of relational design

What are the criteria for "good" relational schemas?

- 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)

Informal Guidelines Design for Relational Databases

- There are four informal measure of quality for relation schema design:
    1. Semantics of the relation attributes.
    2. Redundant values in tuples and update anomalies.
    3. Null values in tuples.
    4. The possibility of generating spurious tuples.
- These measures are not always independent of one another.

1. Semantics of the relation attributes

- o Design a relation schema so that it is easy to explain its meaning.
- o Do not combine attributes from multiple entity sets and relationship sets into a single relation.

Guideline1:

Map each entity set to a relation & each relationship set to a relation.

- o Don't mix these two
- o Only use foreign keys to refer to other entities as opposed to duplicating attributes in other relations.

Example: Consider the following "EMP_DEPT" relation.

EMP_DEPT

EName	SSN	BDate	Address	DNumber	DName	DMGRSSN

It is good design?

- o The attributes semantic of the EMP_DEPT relation are not clear.
- o It is poor design because they violate guideline 1 by mixing attributes from distinct real-world entities.
- o Solution: decompose the "EMP_DEPT" relation into two relations and foreign key.

EMP

EName	SSN	BDate	Address	DNumber

DEPT

DNumber	DName	DMGRSSN

2. Redundant value in tuples and update anomalies

- o Data redundancy refers to copying of the same data in the same table or multiple tables.

Example: Consider the following "EMP_DEPT" relation.

EMP_DEPT

EName	**<u>SSN</u>**	**BDate**	**Address**	**DNumber**	**DName**	**DMGRSSN**
John	12345	09-01-1965	Houston	5	Research	33455
Alicia	33455	08-08-1955	Bellaire	5	Research	33455
….						

Redundancy

- There are problems that result from having such large table:

  1. There is a waste of storage space due of data redundancy.
     - In the previous example, the attribute values (DNumber, DName, and DMGRSSN) are repeated for every employee who works for the same department.

  2. There are anomalies in updating such relation
     - These can be classified into insertion anomalies, deletion anomalies, and modification (update) anomalies.

       A. Insertion Anomalies: there are two types:
          - To insert a new employee tuple into EMP_DEPT, we must include either the **consistent** attribute value for the department that the employee works for, or nulls (if the employee does not work for a department as yet).

          - It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This cause a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity not a department entity.

       B. Deletion Anomalies:
          - Deleting departments will delete their employees.

          - If we delete an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

C. Modification (Update) Anomalies:

- Updates causing the updating of too many rows. for example, if we change the department name of department 5 – we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.

Guideline2:

Minimize the amount of data redundancy in the database.

3. ***Null values in tuples***

- If some attributes do not apply to many tuples, then the occurrence of data in those columns is sparse and storage space is wasted.

- ***Example***: if only 10% of employees have individual office, there is little justification for including the office attributes in the employee relation; rather, a relation EMP_OFFICE (SSN, OfficeNo, OfficeLoc, OfficeTelNo) can be created to include tuples for only the employees with individual offices.

- Queries involving tables having null values might have difficulty dealing with the null values, namely joins and aggregates.

Guideline3:

- Try to minimize the occurrences of NULL values in your design.
- Attributes that are NULL frequently could be placed in separate relations (with the primary keys).

4. ***Spurious Tuples***

- ***Decomposition*** is the process of breaking down larger tables into smaller tables. If the resulting tables are then joined, we should get the original large table. If we do, the decomposition is called ***lossless decomposition***. If we do not get the original table, the decomposition is called ***lossy decomposition***.

- *Example*: Consider the following table

Employee

EID	EName	TelNo	SDate
12345	Kim	555-1234	01/23/2004
54321	Kim	555-5432	05/03/2001

We break up the Employee table into two smaller tables

Emp1

EID	EName
12345	Kim
54321	Kim

Emp2

EName	TelNo	SDate
Kim	555-1234	01/23/2004
Kim	555-5432	05/03/2001

Then we join these two tables back together

EID	EName	TelNo	SDate	
12345	Kim	555-1234	01/23/2004	
12345	Kim	555-5432	05/03/2001	***Spurious***
54321	Kim	555-1234	01/23/2004	***Tuples***
54321	Kim	555-5432	05/03/2001	

This is not the same table. Thus, this is a ***lossy decomposition***.

- To do ***lossless decomposition***, you must decompose on a key (a key existing in the new tables).

Guideline4:

- 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.

7.2 Functional Dependencies

- Functional Dependencies FDs are:
    - Tools for analysis.
    - Used to specify formal measures of the "goodness" of relational designs. And keys are used to define normal forms for relations
    - Constraints that are derived from the meaning and interrelationships of the data attributes.

Definitions

- FD is a constraint between two sets of attributes from the database. Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y.
- $X \rightarrow Y$ means Y is functionally dependent on X or X functionally determines Y.
- $X \rightarrow Y$ 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].
- If $X \rightarrow Y$ holds in r, this does not say whether or not $Y \rightarrow X$ holds in r.
- **Example:** consider r(A,B) with the following instances of r:

A	B
1	4
2	5
1	7

On these instances, $A \rightarrow B$ does not hold, but $B \rightarrow A$ does hold.

- **Example:** Consider r(X,Y,Z) with the following instances of r:

X	Y	Z
X1	Y1	Z1
X2	Y1	Z1
X3	Y2	Z3
X4	Y3	Z2

$X \rightarrow Y$ but Y not determine X
$X \rightarrow Z$ but Z not determine X
$Y \rightarrow Z$ and $Z \rightarrow Y$

- ***Note:*** if X is a candidate key of r – this implies that $X \rightarrow$ r. because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X.

- K is a superkey for relation schema r if and only if K→r (K determines all attributes of r. all attributes of relation occur on the right-hand side of the FD).
- K is a candidate key for r if and only if
  - K→r, and
  - No α K, α→r (no subset of K is a superkey)

- **Example:** Consider r(A,B,C) with the following instances of r:

A	B	C
A1	B1	C1
A2	B1	C1
A1	B2	C1
A4	B3	C2

  Is {A,B} candidate key?

- **Example:** Consider the relation EMP (SSN, Ename, Deptno). Suppose EMP is used to represent many-to-one relationship from EMP to DEPT, where SSN is a key in the EMP relation and Deptno is a key in the DEPT relation.
  - Are the following FDs holds in EMP relation?
    1. {SSN}→{Deptno}
    2. { Deptno}→{SSN}
  - What about many-to-many relationship and one-to-one relationship?

- FD is a property of the semantics or meaning of the attributes.
  - Examples:

    {SSN} → {Ename}
    {PNumber} → {Pname, Plocation}
    {SSN, Pnumber} → {Hours}

- Relation extensions r(R) that satisfy the FD constraints are called legal extensions (or legal relation states) of R.
- A FD is a property of the relation schema (intension) R, not of a particular legal relation state (extension) r of R. hence, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.
- **Example**: Consider the following relation

Teacher	Course	Text
T1	DS	X
T1	C++	Y
T2	Java	Z

We can infer from the relation state (extension) that Course → Text and Text → Course. After insert the following tuple

T3	DS	W

The Text → Course but Course not determine Text. After that, when insert the following tuple

T4	Advanced Java	Z

The Text not determine the Course

- FD constraint must hold at all times.

Trivial VS. Nontrivial FDs

- A FD is trivial iff RHS (dependent) is a subset of the LHS (determinant).
  $X \rightarrow Y$ is trivial if $Y \subseteq X$
- Examples:
  {SSN}→{SSN}
  {SSN,Ename}→{Ename}
- A FD is nontrivial if RHS is not a subset of the LHS.
  $X \rightarrow Y$ is nontrivial if $Y \not\subset X$
- Example:
  {SSN}→{Ename}

Inference Rules for Functional Dependencies

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- The set of all dependences is called the closure of F and is denoted by F^+.

A. ***Armstrong's Axioms***

- We can find all of F^+ by applying Armstrong's axioms.

1. ***IR1:*** Reflexivity rule
   If $Y \subseteq X$, then $X \rightarrow Y$
   Ex: {CNo,CName,BDate} → {CName,BDate}

2. ***IR2:*** Augmentation rule
   $X \rightarrow Y \models XZ \rightarrow YZ$
   Ex: {CNo}→{CName} ⊨ {CNo,BDate}→{CName,BDate}

3. ***IR3:*** Transitive rule
   $\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow Z\}$
   Ex: {CNo→BDate, BDate→Age_Category} ⊨ {CNo→Age_Category}

- These rules are:
  - Sound: generate only functional dependencies that actually hold. And
  - Complete: generate all functional dependencies that hold.

- **Example:** Consider R(X,Y,Z) with the following FDs
  $F=\{X \rightarrow Y,$
  $X \rightarrow Z\}$
  We can infer the following FDs

1. X→XY (Augmentation by X)
2. YX→YZ (Augmentation by Y)
3. X→YZ (Transitivity)

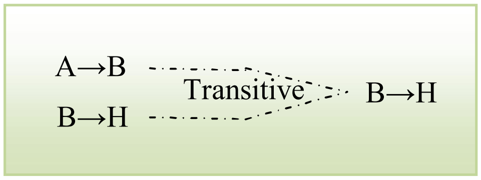
- **Example:** Consider R(A,B,C,G,H,I) with the following FDs

  F={A→B,
  A→C,
  CG→H,
  CG→I,
  B→H}

  Proof the following FDs

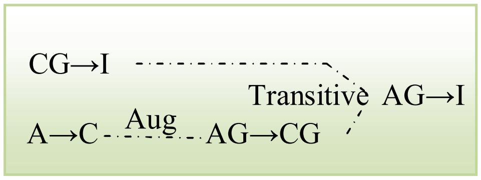
  1. A→H

     By transitivity from A→B and B→H



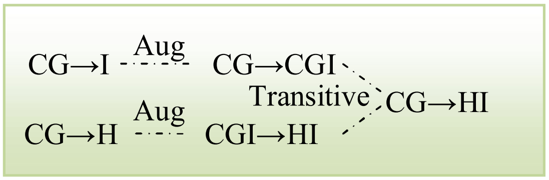
  2. AG→I

     By augmentation A→C with G, to get AG→CG and then transitivity with CG→I



  3. CG→HI

     By augmentation CG→I with CG, to get CG→CGI, then by augmentation CG→H with I, to get CGI→HI, and then by transitivity CG→HI



B. Additional Inference rules

- Can be derived from Armstrong's Axioms.
- We can further simplify manual computation of F^+ by using the following additional rules.

4. ***IR4:*** Decomposition rule
   $\{X \to YZ\} \models \{X \to Y \text{ and } X \to Z\}$

$$\begin{tikzcd}
\text{By reflexivity} \arrow[r, dashed] & YZ \to Y \arrow[dr, dashed, "\text{Transitive}"'] & \\
 & & X \to Y \\
X \to YZ \arrow[urr, dashed] & &
\end{tikzcd}$$

$$\begin{tikzcd}
\text{By reflexivity} \arrow[r, dashed] & YZ \to Z \arrow[dr, dashed, "\text{Transitive}"'] & \\
 & & X \to Z \\
X \to YZ \arrow[urr, dashed] & &
\end{tikzcd}$$

5. ***IR5:*** Union rule
   $\{X \to Y, X \to Z\} \models \{X \to YZ\}$

$$\begin{tikzcd}
X \to Y \arrow[r, dashed, "\text{Aug}"] & X \to XY \arrow[dr, dashed, "\text{Transitive}"'] & \\
 & & X \to YZ \\
X \to Z \arrow[r, dashed, "\text{Aug}"] & XY \to YZ \arrow[ur, dashed] &
\end{tikzcd}$$

6. ***IR6:*** Composition rule
   $\{X \to Y, W \to Z\} \models \{XW \to YZ\}$

$$\begin{tikzcd}
X \to Y \arrow[r, dashed, "\text{Aug}"] & XW \to YW \arrow[dr, dashed, "\text{Transitive}"'] & \\
 & & XW \to YZ \\
W \to Z \arrow[r, dashed, "\text{Aug}"] & WY \to YZ \arrow[ur, dashed] &
\end{tikzcd}$$

7. ***IR7:*** Psuedotransitive rule
   $\{X \to Y, WY \to Z\} \models \{WX \to Z\}$

$$\begin{tikzcd}
WY \to Z \arrow[drr, dashed, "\text{Transitive}"'] & & \\
 & & WX \to Z \\
X \to Y \arrow[r, dashed, "\text{Aug}"] & XW \to YW \arrow[ur, dashed] &
\end{tikzcd}$$

Closure of attribute sets, X+

- F^+ can grow quite large, as we keep applying rules to find more FDs.

- Sometimes we want to find all of F^+, and other times we just want to find part of it.
- We are often interested in finding the part that tells us whether or not some subset of attributes x is a superkey for R.
- If you can uniquely determine all attributes in R by some subset of attributes X, then X is a superkey for R.
- The closure of X under F, denoted X^+, is the subset of attributes that are uniquely determined by X under F.

- ***Definition:***
    - Given a schema R, a set X of attributes in R, and a set F of FDs that hold for R, then the set of all attributes of R that are functionally dependent on X is called closure of X under F (X^+)

- ***Algorithm:*** Determining X^+, the closure of X under F.

```
X+ := X;
 Repeat
      oldX+ := X+ ;
      for each functional dependency Y→Z in F do
             if Y ⊆ X+ then X+ := X+ U Z;
 Until (X+ = oldX+);
```

- **Example:** Consider R(A,B,C,D,E,F) with the following FDs
  F={A→BC,
  B→CF,
  B→E,
  CD→EF}
  Find $\{AB\}^+$

oldX+	X+
AB	AB
	ABC
	ABCF
	ABCEF
ABCEF	

  $\{AB\}^+ = \{A\ B\ C\ E\ F\}$
  {AB} is not a superkey because it is not determine R

- **Example:** Consider R(A,B,C,G,H,I) with the following FDs
  F={A→B,
  CG→H,
  CG→I,
  A→C,
  B→H}

Is {AG} a candidate key?

oldX+	X+
AG	**AG**
	ABG
	ABCG
	ABCGH
ABCGH	
	ABCGHI
ABCGHI	

$\{AG\}^+ = \{A\ B\ C\ G\ H\ I\}$
{AG} is a superkey

{AG} is a candidate key, because no subset of AG is a superkey. ($\{A\}^+ = \{A\ B\ C\ H\}$ and $\{G\}^+ = \{G\}$)

- **Example:** Consider R(A,B,C,D,E) with the following FDs

$$F=\{AB\rightarrow C,\ A\rightarrow D,\ D\rightarrow E,\ AC\rightarrow B\}$$

Is {AB} a candidate key?

7.3 Normalization

- Formal process for designing relational database schema.
- Start from larger tables and decompose them until each smaller table follows a "good" form.
- The normalization process takes a database schema through a series of steps that reduce at each step the amount of **data redundancy** and the number of **anomalies**.
- In normalization theory, there are multiple levels of normal forms, denoted as NF1, NF2, etc. which represent each step in the normalization process.
- We assume that a set of functional dependencies is given for each relation, and that each relation has a designated key. The FDs and Keys are used to tests for normal form.
- The normalization process, as first proposed by Codd (1972), takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
- 1NF, 2NF, 3NF, and BCNF based on the functional dependencies and keys.

Definitions:

- 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].

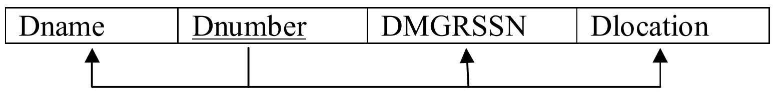
- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.
- If a relation schema has more than one key, each is called a **candidate** key.
    - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **alternate 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 (1NF)

- Disallows having a set of values (multivalue attribute), a tuple of values (composite attribute), or a combination of both (complex attribute) as an attribute value for a single tuple.
- The only attribute values permitted by 1NF are single atomic (or indivisible) values.

- **Example1: (Multivalue attribute)**

    Consider a relation DEPARTMENT, which has a multivalue attribute DLocation. Each department can have multiple locations. DEPARTMENT (Dname, <u>Dnumber</u>, DMGRSSN, {Dlocation})



There are two ways we can look at the Dlocations attribute:

1. The domain of Dlocation contains atomic values. In this case, Dlocation is not functionally dependent on Dnumber.

Dname	<u>Dnumber</u>	DMGRSSN	Dlocation
Research	5	33455	LocX
Research	5	33455	LocY
Research	5	33455	LocZ
Administration	4	98765	LocW
Headquarters	1	88677	LocZ

2. The domain of Dlocation sets of values and hence is nonatomic in this case, Dnumber→Dlocation.

Dname	<u>Dnumber</u>	DMGRSSN	Dlocation
Research	5	33455	{LocX, LocY, LocZ}
Administration	4	98765	LocW
Headquarters	1	88677	LocZ

There are three main techniques to achieve first normal form for such a relation:

I. *First Solution:* Decompose the non 1NF relation into two 1NF relations, by remove the attribute Dlocation that violate 1NF and place it in a separate relation Dept_Location. The primary key of this relation is the combination (Dnumber, Dlocation).

Dname	<u>Dnumber</u>	DMGRSSN
Research	5	33455
Administration	4	98765
Headquarters	1	88677

<u>Dnumber</u>	<u>Dlocation</u>
5	LocX
5	LocY
5	LocZ
4	LocW
1	LocZ

This solution is the ***<u>best</u>*** because no redundancy in the relations and no limit placed on a maximum number of location values.

II. *Second solution:* Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a department.
The primary key is (Dnumber, Dlocation)

Dname	<u>Dnumber</u>	DMGRSSN	<u>Dlocation</u>
Research	5	33455	LocX
Research	5	33455	LocY
Research	5	33455	LocZ
Administration	4	98765	LocW
Headquarters	1	88677	LocZ

-ve: *introduce redundancy in the relation.*

III. *Third solution:* If a maximum number of values is known for the attribute – for example, if it is known that at most three locations can exist for a department. Replace the Dlocation attribute by three atomic attributes: Dloc1, Dloc2 and Dloc3

Dname	<u>Dnumber</u>	DMGRSSN	Dloc1	Dloc2	Dloc3
Research	5	33455	LocX	LocY	LocZ
Administration	4	98765	LocW	*NULL*	*NULL*

Headquarters	1	88677	LocZ	*NULL*	*NULL*

-ve: *introduce Null values in the relation if most departments have fewer than 3 locations. This solution breaks guideline #3 by introducing too many nulls and thus wasting space.*

- **Example2: (Composite attribute)**

  Consider a relation EMPLOYEE, which has a composite attribute Name. Each Name have first and last name. EMPLOYEE (SSN, Name (FName, LName), Address)

SSN	Name		Address
	FName	LName	
2314	Adam	James	X
6543	Sali	John	Y
7642	Smith	King	Z

  The EMPLOYEE relation is not in 1NF, because the Name attribute not atomic attribute

  Solution, separate each component in the individual column.

SSN	FName	LName	Address
2314	Adam	James	X
6543	Sali	John	Y
7642	Smith	King	Z

- **Example3: (Complex attribute (nested relations))**

  Consider a relation EMP_PROJ, which has a complex attribute Projs. Which consist of Pnumber and Hours.

  EMP_PROJ (SSN, Ename, Projs {(Pnumber, Hours)})

SSN	Ename	PNumber	Hours
2314	Adam	1	32.5
		2	7.5
6543	Sali	3	40.5
7642	Smith	2	10.0
		3	10.0
		10	15.0
		20	12.0

  SSN is the primary key of the EMP_PROJ relation and Pnumber is the partial key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber.

To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it; the primary key of the new relation will combine the partial key with the primary key of the original relation.

Solution:

<u>SSN</u>	Ename

<u>SSN</u>	<u>PNumber</u>	Hours

Second Normal Form (2NF)

- 2NF is based on the concept of full functional dependency.
- X→Y is a fully functional dependency, if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A □ X, (X-{A}) does not functionally determine Y.
- X→Y is a partial dependency if some attribute A □ X can be removed from X and the dependency still holds.
- Example:

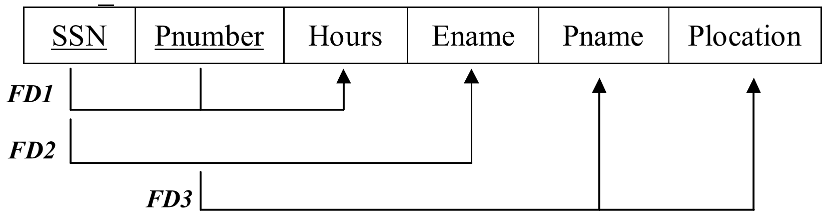
  {SSN, Pnumber}→{Hours} is a full dependency, because neither {SSN→Hours} nor {Pnumber→Hours} holds.

  {SSN, Pnumber}→{Ename} is a partial dependency, because {SSN→Ename} holds.

A. 2NF based on the primary key

- 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
  - If the primary key contains a single attribute, the relation in the 2NF.
- **Example:** Consider the following relation. Determine if the relation in 2NF, if not normalize the relation into 2NF.

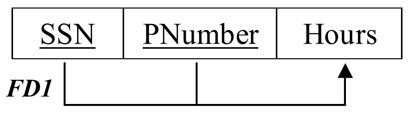
EMP_PROJ

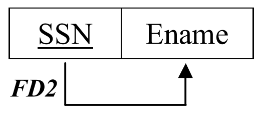


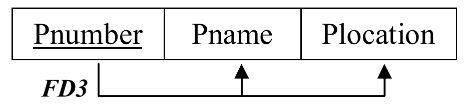
- There are two problems in the previous relation:

A. The semantic of the attributes are not clear.
B. Redundant data (update anomalies)

- EMP_PROJ relation is in 1NF but is not in 2NF, because the nonprime attribute Ename violate 2NF (FD2); as do the nonprime attributes Pname and Plocation (FD3).
- FD2 and FD3 make Ename, Pname and Plocation partially dependent on the primary key {SSN, Pnumber} of EMP_PROJ
- Solution: separate each FD violate 2NF into new relation as follows:







- After decomposition:
  - No redundant values
  - No spurious tuples through join

- **Example:** Consider the following relation. Determine if the relation in 2NF, if not normalize the relation into 2NF.

Dept	Item	Price
D1	T3	50
D1	T2	40
D2	T5	40
D2	T2	40

FD1: {Dept, Item}→{Price}
FD2: {Item}→{Price}
The relation is in 1NF but it is not in 2NF (FD2)

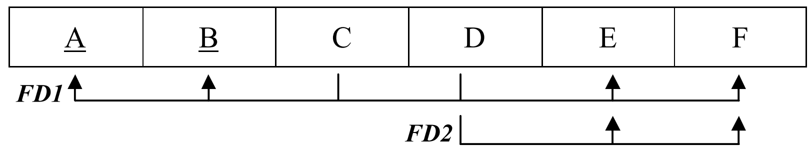
Solution:

Dept	Item
D1	T3
D1	T2
D2	T5
D2	T2

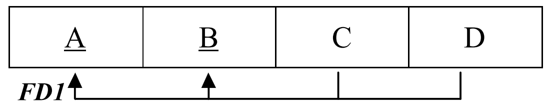
Item	Price
T3	50
T2	40
T5	40

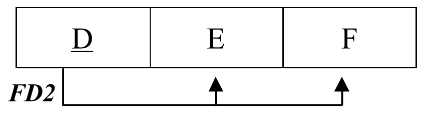
B. General definition of 2NF

- A relation schema R is in 2NF if every nonprime attribute A in R is fully dependent on any key of R.
- **Example:** Consider the following relation. It is in 2NF or not?



The relation in 1NF but it is not in 2NF (FD2)
Solution:





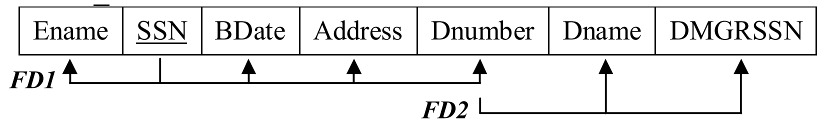
Third Normal Form (2NF)

- 3NF is based on the concepts of transitive dependency.
- X→Y in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both X→Z and Z→Y hold.

A. 3NF based on the primary key

- 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.
- Example: Consider the following relation. Determine if the relation in 3NF, if not normalize the relation into 3NF.

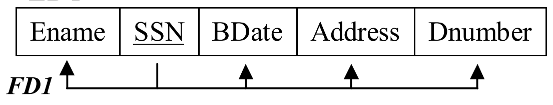
EMP_DEPT



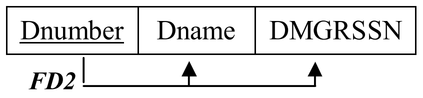
{SSN→DMGRSSN} is transitive dependent through Dnumber (SSN→Dnumber, Dnumber→DMGRSSN), and Dnumber is neither a key itself nor a subset of the key of EMP_DEPT.

Solution:

ED1



ED2



Note: A natural join operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples.

- **Example:** Consider the following relation. Determine if the relation in 3NF, if not normalize the relation into 3NF.

Item	Category	Discount
T3	A	10
T2	B	2
T5	C	12
T7	B	2

FD1: {Item}→{Category}
FD2: {Item}→{Discount}
FD3: {Category}→{Discount}
The relation is in 1NF and in 2NF but it is not in 3NF (FD3)

Solution:

Item	Category
T3	A
T2	B
T5	C
T7	B

Category	Discount
A	10
B	2
C	12

B. General definition of 3NF

- A relation schema R is in 3NF if, whenever a nontrivial functional dependency X→A holds in R, either
  1. X is a superkey of R.
     Or
  2. A is a prime attribute of R.

- **Example:** Consider the following relation

<u>Deptno</u>	DName	Location
10	R	LocX
20	A	LocY
30	H	LocZ

FD1: {Deptno}→{DName, Location}
FD2:{DName}→{Location}

The relation is in 3NF because DName is a superkey.

- **Example:** Consider the following relation

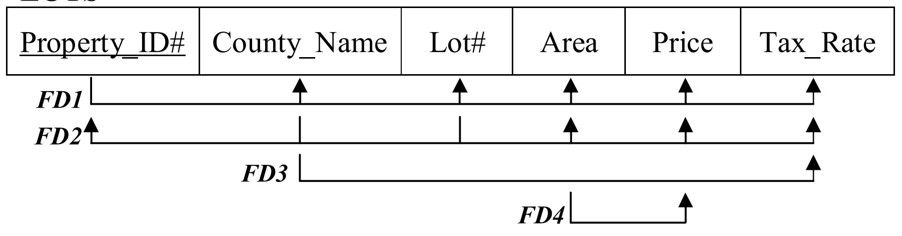
<u>SSN</u>	<u>Pnumber</u>	Hours
1234	101	10
5678	101	10
1234	103	20
4444	101	10
4444	103	30

FD1: {SSN, Pnumber}→{Hours}
FD2:{Hours}→{Pnumber}

The relation is in 3NF because Pnumber is a prime attribute.

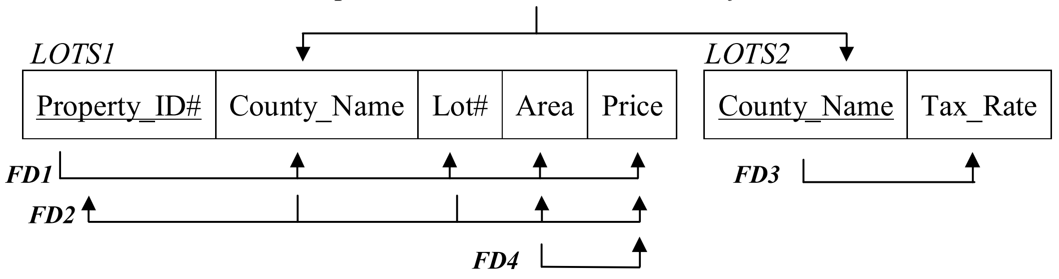
- **Example:** Consider the following relation, it is in the 3NF? If not normalize the relation into 3NF.

LOTS



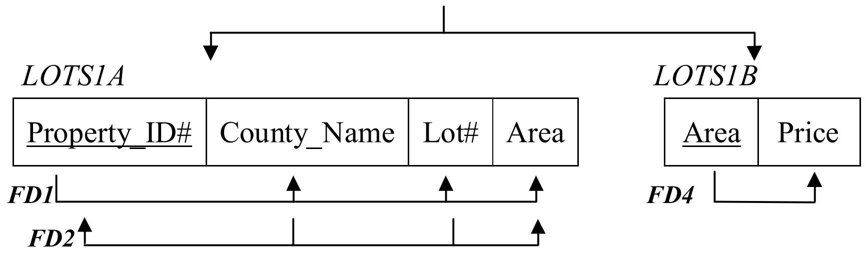
The LOTS relation in 1NF but it is not in 2NF (FD3)

Decompose LOTS into two relations as follows



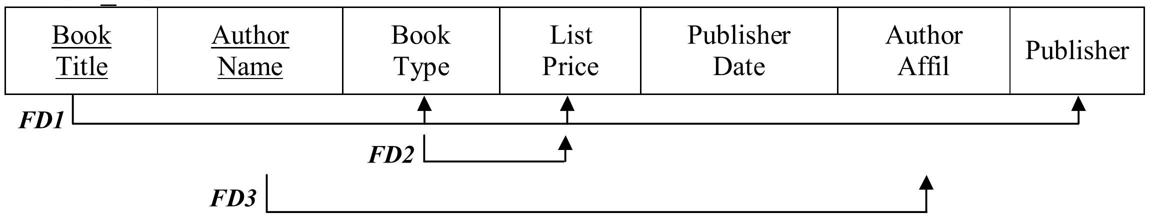
LOTS1 in 2NF but not in 3NF (FD4)

Decompose LOTS1 into two relations as follows



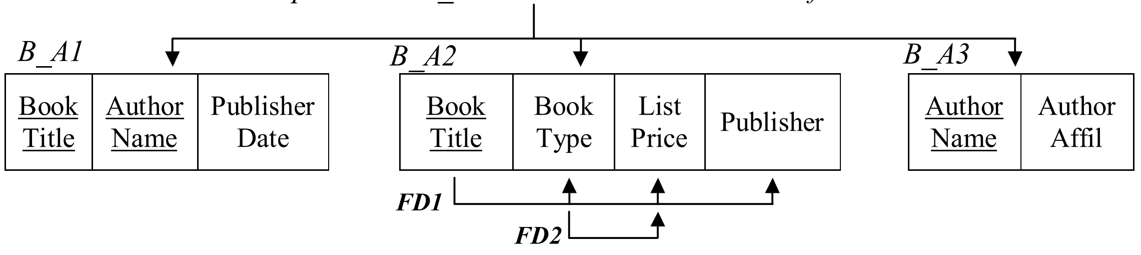
- **Example:** Consider the following relation, it is in the 3NF? If not normalize the relation into 3NF.

BOOK_AUTH



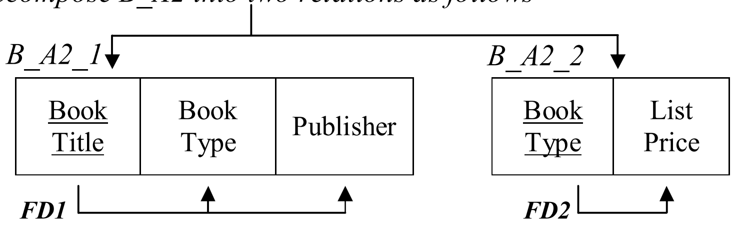
The BOOK_AUTH relation in 1NF but it is not in 2NF (FD1 and FD3)

Decompose BOOK_AUTH into three relations as follows



The B_A2 in 2NF but not in 3NF (FD2)

Decompose B_A2 into two relations as follows

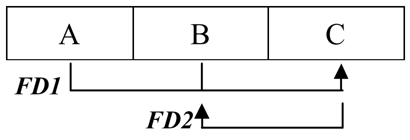


Finally there are four relations (B_A1, B_A3, B_A2_1, B_A2_2)

Boyce_Codd Normal Form (BCNF)

- BCNF was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF, because every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF.
- A relation schema R is in BCNF if whenever a nontrivial FD X→A holds in R, then X is a superkey of R.
- The only difference between the definitions of BCNF and 3NF is the second condition of 3NF, which allows A to be prime, is absent from BCNF.

- **Example:** Consider the following relation



a1	**b1**	**c1**
a1	b2	c3
a2	b2	c4
a3	b2	c3
a3	b3	c5

The relation is in 3NF but it is not in BCNF (FD2)
Solution: Decompose the relation into two relations

A	C

C	B

Note: *the decomposition lose the functional dependency FD1*

- **Example:** Consider the following relation

Student	Course	Instructor
Jamal	DB	Randa
Hosam	DB	Sawsan
Hosam	OS	Ali
Hosam	Theory	Osama
Nada	DB	Randa
Nada	OS	Belal
Mohammad	DB	Anas
Omar	DB	Sawsan

FDs:
FD1: {Student, Course}→{Instructor}

FD2: {Instructor}→{Course}

The relation is in 3NF, but it is not in BCNF (FD2)

Solution:
There are three possible decomposition of the relation

1. {student, instructor} and {student, course}
2. {course, instructor } and {course, student}
3. {instructor, course } and {instructor, student}

Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.

Student	Instructor
Jamal	Randa
Hosam	Sawsan
Hosam	Ali
Hosam	Osama
Nada	Randa
Nada	Belal
Mohammad	Anas
Omar	DB

Instructor	Course
Randa	DB
Sawsan	DB
Ali	OS
Osama	Theory
Belal	OS
Anas	DB

- **Example:** Consider the following relation for Chocolate Factory

CustID	Name	Order#	OrderDate	Product Code	Prod Desc	Price	Qty
C004	Adams, Anne	Ord001	03/08/2000	P100	TopDeck	3.25	150
C004	Adams, Anne	Ord001	03/08/2000	P300	KitKat	3.10	100
C003	Jones, Carol	Ord002	05/08/2000	P200	BarOne	2.95	240
C002	Black, Roger	Ord003	05/08/2000	P500	MilkyBar	3.20	370
C002	Black, Roger	Ord003	05/08/2000	P400	Flake	3.40	120
C001	Smith, John	Ord004	09/08/2000	P300	KitKat	3.10	280
C005	Rhodes, Sean	Ord005	12/08/2000	P400	Flake	3.40	150
C002	Black, Roger	Ord006	13/08/2000	P100	TopDeck	3.25	320
C001	Smith, John	Ord007	15/08/2000	P500	MilkyBar	3.20	240

How would you split the previous relation to minimize data redundancy?