Module 4: Normalization

SYLLABUS

- Different anomalies in designing a database, The idea of normalization, Functional dependency, Armstrong's Axioms (proofs not required), Closures and their computation, Equivalence of Functional Dependencies (FD), Minimal Cover (proofs not required).
- First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boysce Codd Normal Form (BCNF),
- Lossless join and dependency preserving decomposition, Algorithms for checking Lossless Join (LJ) and Dependency Preserving (DP) properties.

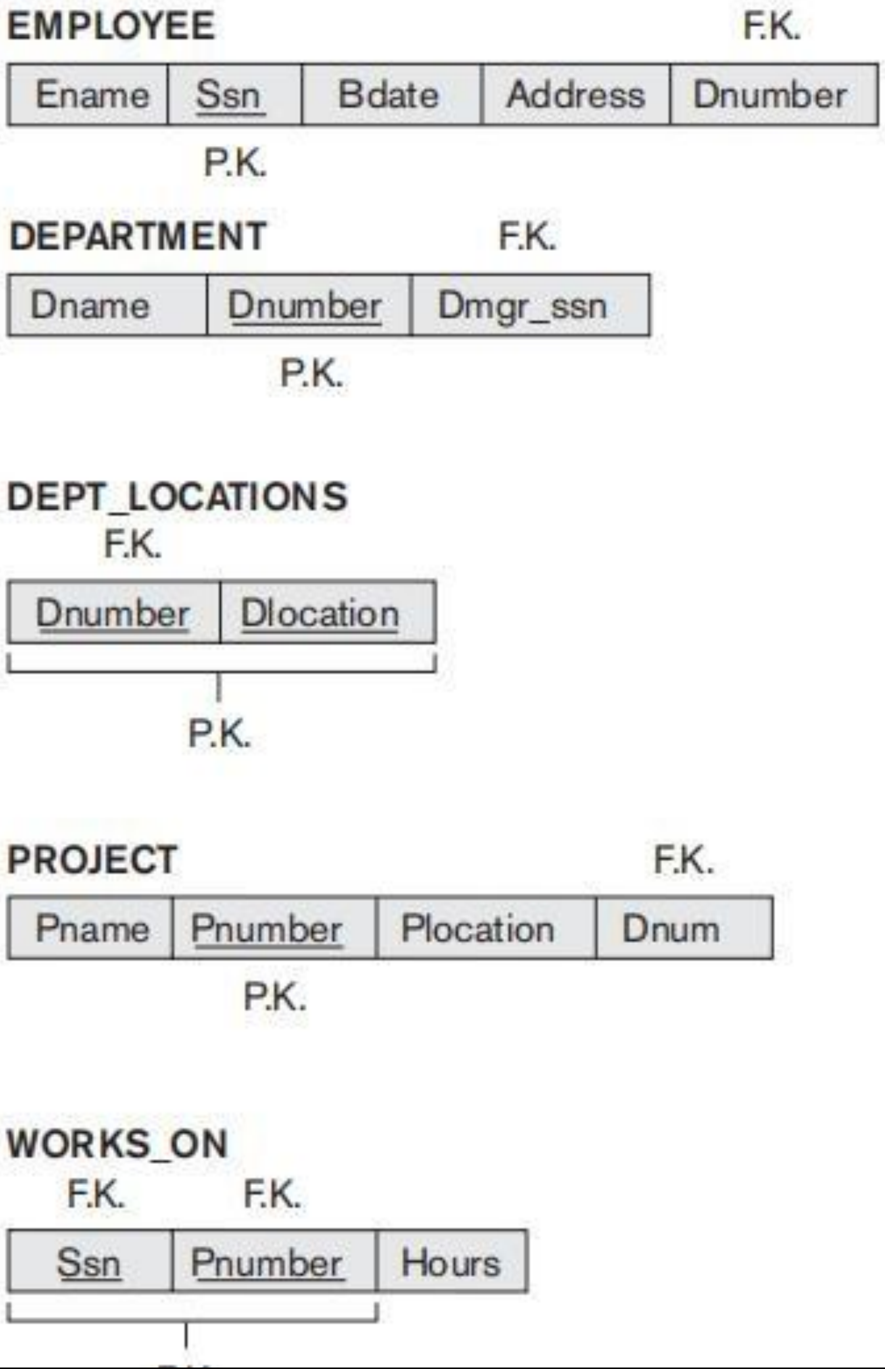
Informal Design Guidelines for Relation Schema

- Four informal guidelines that may be used as 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

- Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them.
- The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple

Figure 15.1
A simplified COMPANY relational database schema.



EMPLOYEE

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

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

Guideline 1

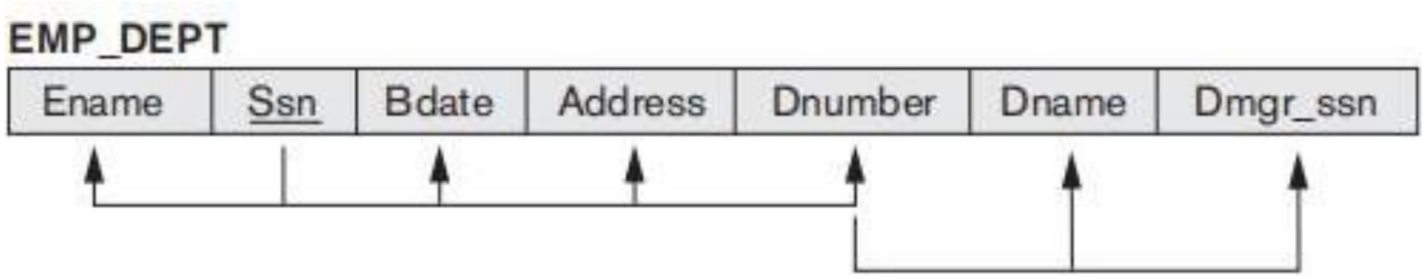
- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning.
- Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

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.

Important: We should aim to *design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret and understand.*

Examples of Violating Guideline 1



Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Sam	1234	12-03-1991	Pune	1	cse	4321
Ram	4321	21-04-1995	Delhi	2	ece	5432
Sita	5432	30-01-1992	Kerala	1	cse	43211

ambiquity

combine attributes from Employee and Department into single table this lacks meaning

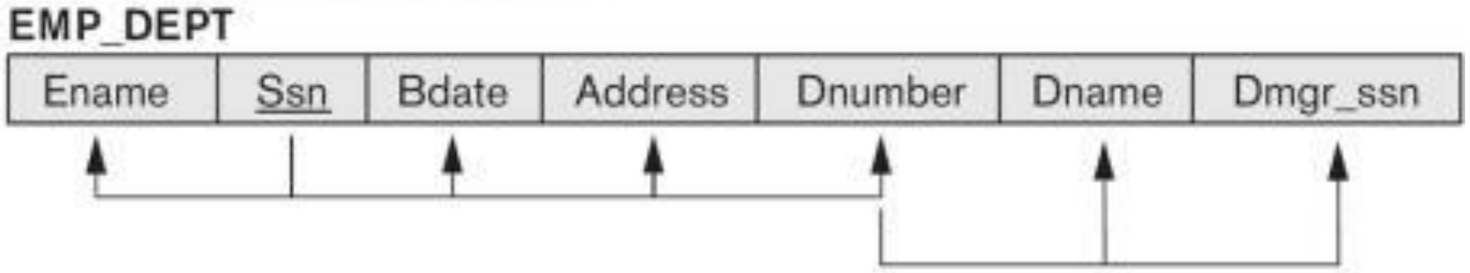
Redundant Information in Tuples and Update Anomalies

- Data redundancy is a condition created within a database or in which the same piece of data is held in two separate places.
- Redundancy leads to
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

Insertion Anomalies

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

Consider a relation EMP_DEPT (Ename, <u>Ssn,</u> Bdate, Address, Dnumber, Dname, Dmgr_ssn)

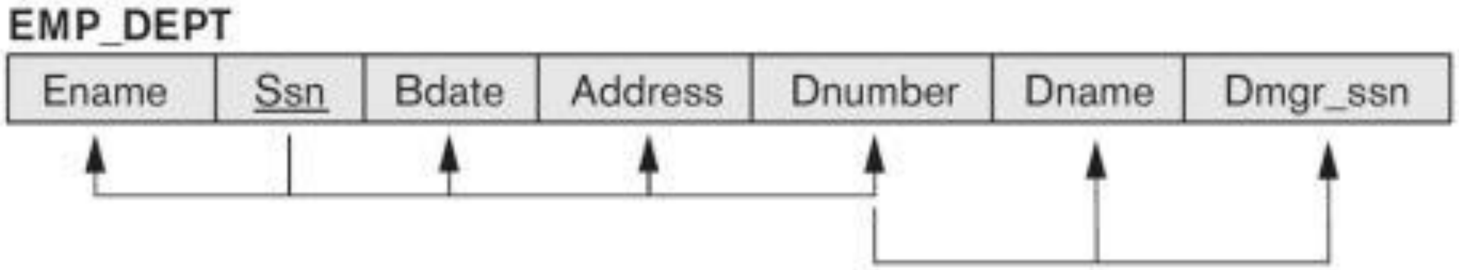


insertion anomalies: when adding an employee, we must assign them to a department or else use NULLs. When adding a new department with no employees, we have to use NULLs for the employee Ssn, which is supposed to be the primary key!

Deletion Anomalies

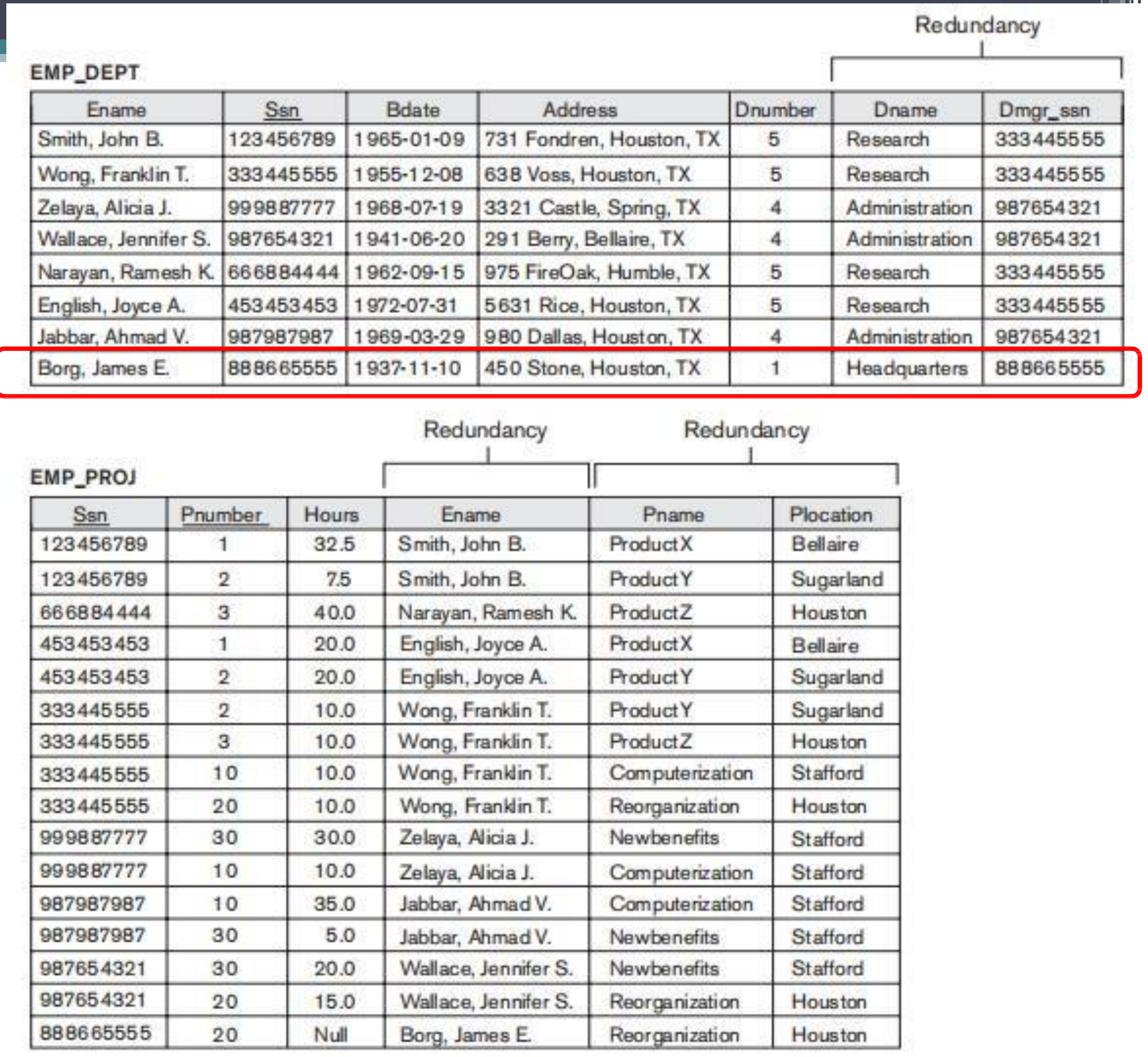
- If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.
- Consider the relation: EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
    - When a project is deleted, it will result in deleting all the employees who work on that project.
    - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Consider a relation EMP_DEPT (Ename, Ssn, Bdate, Address, Dnumber, Dname, Dmgr_ssn)



deletion anomalies: if we delete the last EMP_DEP record from a department, or if there is only one employee working in a department. Deleting that record means we have lost the information about the department!

Deleting Borg,James record leads to losing data about Head Quarters dept.
We cannot insert details about new department as no new employee recruited in it yet.
If the Dept manager changes we need to update updating Dmgr_ssn for all records.
Like wise we would have to update Pname for all records if project name is updated.


Redundancy
Redundancy
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

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

Modification Anomalies

- EMP_DEPT, if we change the value of one of the attributes of a particular department say,
  - the manager of department 5 we must update the tuples of all employees who work in that department;
  - otherwise, the database will become inconsistent.
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

- Consider the relation:
- EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer_x0002_Accounting” may cause this update to be made for all 100 employees working on project P1.

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.

NULL Values in Tuples

- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable
- NULL can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level
- Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied.
- if NULL values are present, the results may become unpredictable

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)
- For example, if only 15 percent of employees have individual offices,
  - there is little justification for including an attribute Office_number in the EMPLOYEE relation;
  - rather, a relation EMP_OFFICES(Essn, Office_number) can be created to include tuples for only the employees with individual offices

Generation of Spurious Tuples – avoid at any cost

- Consider the tables
  - EMP_LOCS(EName, PLocation)
  - EMP_PROJ1(SSN, PNumber, Hours, PName, PLocation)
- versus the table
  - EMP_PROJ(SSN, PNumber, Hours, EName, PName, PLocation)
- If we use the former as our base tables then we cannot recover all the information of the latter because trying to natural join the two tables will produce many rows not in EMP_PROJ.
- These extra rows are called spurious tuples.
- Another design guideline is that relation schemas should be designed so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way such that no spurious tuples are generated.

EMP_LOCS

<u>Ename</u>	<u>Plocation</u>

P.K.

EMP_PROJ1

<u>Ssn</u>	<u>Pnumber</u>	Hours	Pname	Plocation

P.K.

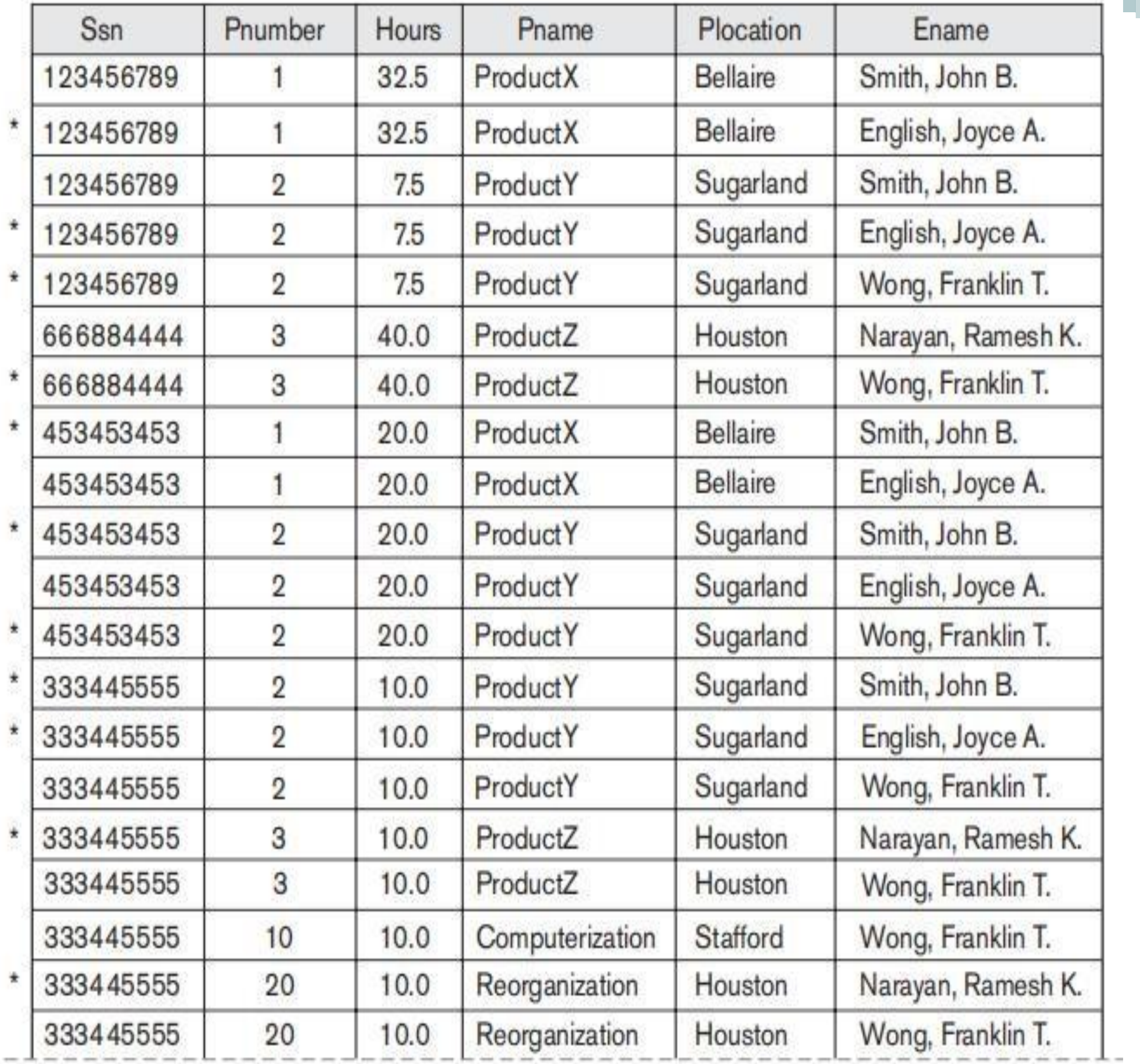
EMP_LOCS

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

EMP_PROJ1

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

- Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS.
- If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ. Additional tuples that were not in EMP_PROJ are called spurious tuples



	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

- Decomposing EMP_PROJ into EMP_LOCS and EMP_PROJ1 is undesirable because when we JOIN them back using NATURAL JOIN, we do not get the correct original information.
- This is because in this case Plocation is the attribute that relates EMP_LOCS and EMP_PROJ1, and Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.

R(A,B,C)

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c2

R1(A)

A
a1
a2

R2(B,C)

B	C
b1	c1
b2	c2

R1XR2

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c2
a2	b2	c2

SPURIOUS TUPLE

Guideline 4

- Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated.
- 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

- Anomalies that cause redundant work to be done during insertion into and modification of a relation, and that may cause accidental loss of information during a deletion from a relation
- Waste of storage space due to NULLs and the difficulty of performing selections, aggregation operations, and joins due to NULL values
- Generation of invalid and spurious data during joins on base relations with matched attributes that may not represent a proper (foreign key, primary key) relationship

Functional dependencies

- Functional Dependencies
  - Are used to specify formal measures of the "goodness" of relational designs
  - And keys are used to define normal forms for relations
  - Are constraints that are derived from the meaning and interrelationships of the data attributes
  - A functional dependency is a constraint between two sets of attributes from the database.
  - Suppose that our relational database schema has n attributes A1, A2, ..., An)

A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

Definition: A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[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]
- $X \rightarrow Y$ in R specifies a constraint on all relation instances r(R)
- Written as $X \rightarrow Y$; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Examples of functional dependencies

- 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

A	B
a1	b1
a2	b3
a1	b2
a2	b3

$A \rightarrow$ B   So this is not a valid FD no unique matching

X a1   (b1,b2)

✓ a2   b3

$B \rightarrow$ A   So this is a valid FD

✓ b1 a1

✓ b3 a2

✓ b2 a1

$B \rightarrow$ A implies

☐ B functionally determines A

☐ A functionally depends on B

☐ A is functionally determined by B

Exercise

EMPLOYEE(Eid, Ename, Eage, Dnum)

DEPT(Dno, Dname, Dloc)

Find valid FDs

1. Eid→Ename
2. Ename→Eid
3. Eage→Ename
4. Dno→Dname, Dloc

Eid → Ename

- Since Eid is a Primary Key so every value is unique
- So FD satisfies

Ename → Eid

•FD do not satisfy

Eid	Ename
1	Bob
2	Bob

Eage → Ename

- FD do not satisfy

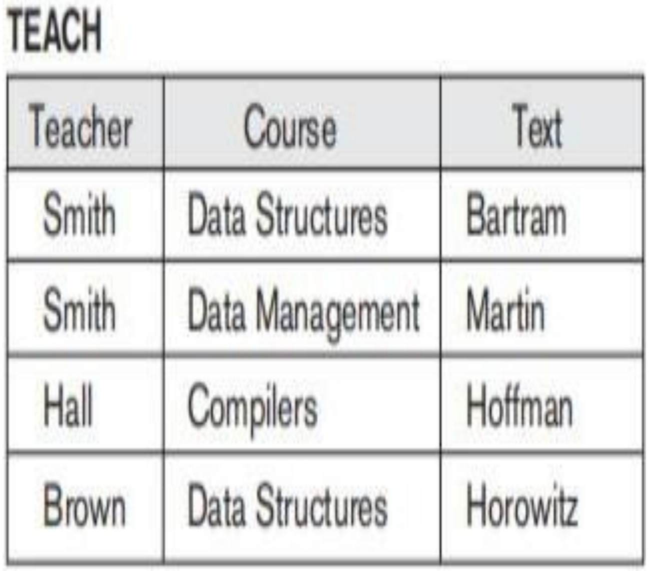
Eid	Ename	Eage
1	Bob	20
2	Bob	20

Dno → Dname, Dloc

- FD satisfy since Dno is a primary key

- A functional dependency is a property of the semantics or meaning of the attributes.
- Functional Dependancy must be valid for every relation state.
- Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint.
- Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R.
- Hence, the main use of functional dependencies is to describe further a relation schema R by specifying constraints on its attributes that must hold at all times

- A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R.
- Therefore, 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.



TEACH

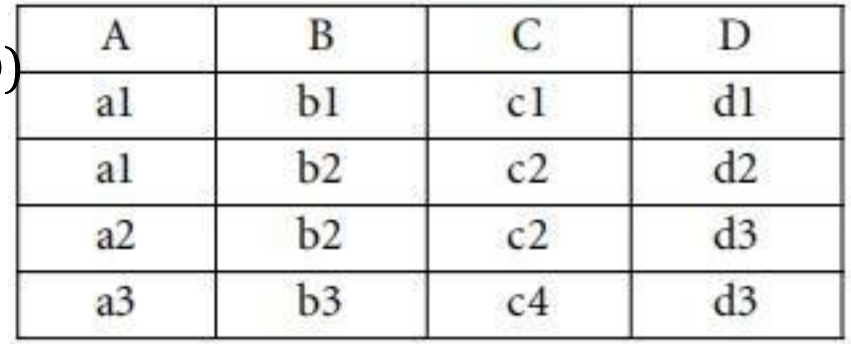
Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Although at first glance we may think that Text →Course, we cannot confirm this unless we know that it is true for all possible legal states of TEACH.
It is, however, sufficient to demonstrate a single counterexample to disprove a functional dependency.
For example, because ‘Smith’ teaches both ‘Data Structures’ and ‘Data Management,’ we can conclude that Teacher does not functionally determine Course

- Given a populated relation, one cannot determine which FDs hold and which do not unless the meaning of and the relationships among the attributes are known.
- All one can say is that a certain FD may exist if it holds in that particular extension.
- One cannot guarantee its existence until the meaning of the corresponding attributes is clearly understood.
- One can, however, emphatically state that a certain FD does not hold if there are tuples that show the violation of such an FD.

- following FDs may hold because the four tuples in the current extension have no violation of these constraints:
- B → C;
- C → B;
- {A, B} → C;
- {A, B} → D; and
- {C, D} → B.

R (A, B, C, D)



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

- However, the following do not hold because we already have violations of them in the given extension:
- A → B (tuples 1 and 2 violate this constraint);
- B → A (tuples 2 and 3 violate this constraint);
- D → C (tuples 3 and 4 violate it).

Types of Functional Dependancy

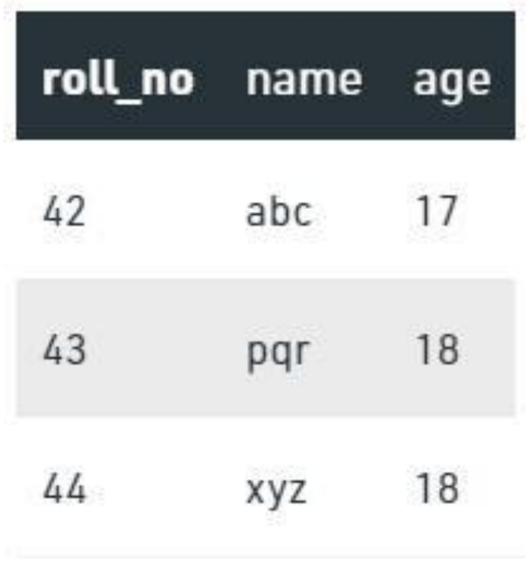
1. Trivial FD
   - In Trivial Functional Dependency, a dependent is always a subset of the determinant.
   - It is FD of the form A→A
   - Not a useful FD since we are not getting any important information here
   - Eg,

Eid→Ename

Eid, Ename→Ename // trivial

2. Non Trivial FD

- In Non-trivial functional dependency, the dependent is strictly not a subset of the determinant.
- i.e. If X → Y and Y is not a subset of X, then it is called Non-trivial functional dependency.



roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

roll_no → name is a non-trivial functional dependency, since the dependent name is not a subset of determinant roll_no

Similarly, {roll_no, name} → age is also a non-trivial functional dependency, since age is not a subset of {roll_no, name}

- Semi Non Trivial
  - Trivial with extra information
  - $AB \rightarrow BC$

Properties of Functional Dependencies

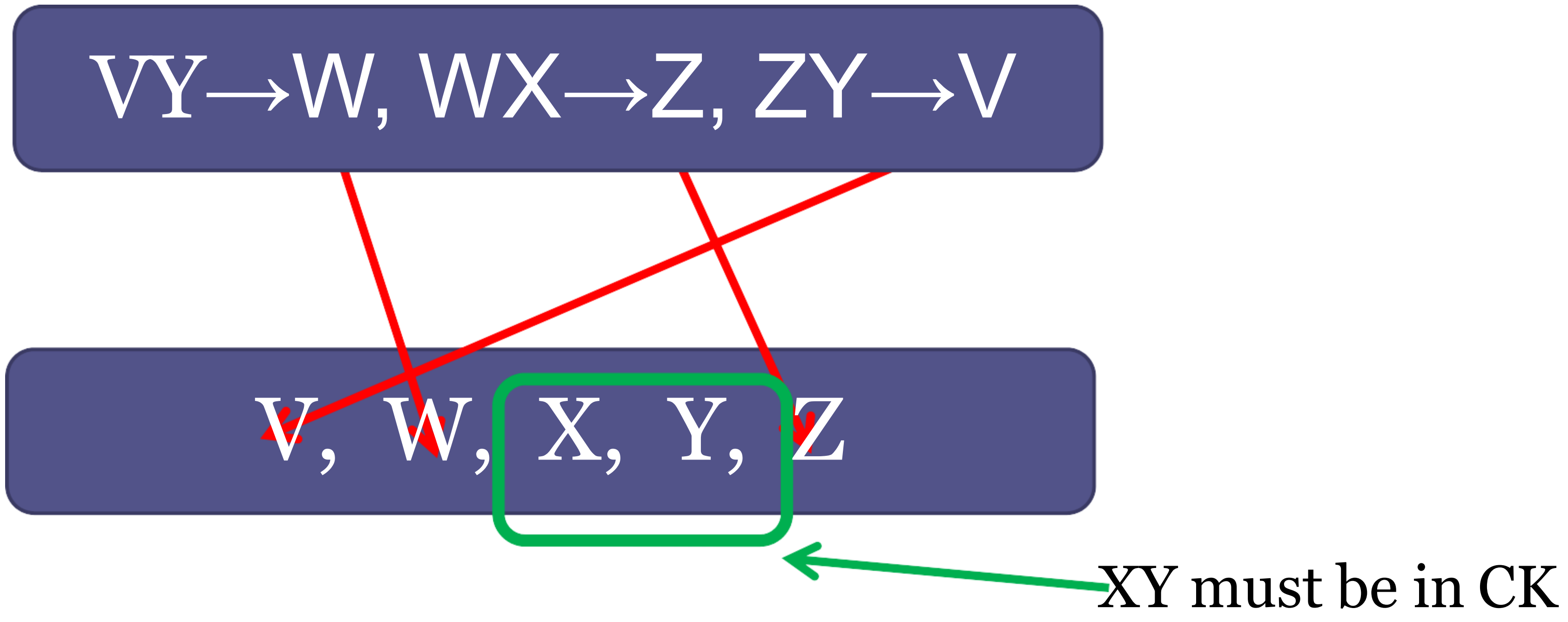
- There are several useful rules that let you replace one set of functional dependencies with an equivalent set.
- Some of those rules are as follows:
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
  - Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
  - Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Decomposition: If $X \rightarrow Y Z$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity: If $X \rightarrow Y$ and $W Y \rightarrow Z$, then $W X \rightarrow Z$
  - Composition: If $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

Closure set of attribute

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
- To find attribute closure of an attribute set:
  - Add elements of attribute set to the result set.
  - Recursively add elements to the result set which can be functionally determined from the elements of the result set

- R(A,B,C,D)with FD={$A \rightarrow B$, $B \rightarrow C$,$C \rightarrow D$,$D \rightarrow A$}
- Closure of A, A^+=attribute which can be determined from A
- A^+=ABCD (ie using closure if we can cover all attribute then it is called Candidate Key CK)
- B^+=BCDA ✔CK
- C^+=CDAB ✔ CK
- D^+=DABC ✔ CK
- Candidate Keys of R are A,B,C,D

- R(V,W,X,Y,Z)
- FD { VY→W, WX→Z, ZY→V}



XY+=XY □

VXY+=VXYWZ ✔ CK

WXY+=WXYZV ✔ CK

Armstrong’s Axioms in Functional Dependency

- The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W.
- Armstrong, that is used to test the logical implication of functional dependencies.
- If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F.
- Armstrong’s Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Axioms

- Axiom of reflexivity
  - If A is a set of attributes and B is subset of A, then A holds B. If $B \subseteq A$ then $A \rightarrow B$
  - This property is trivial property.
- Axiom of augmentation
  - If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds.
  - That is adding attributes in dependencies, does not change the basic dependencies.
  - If $A \rightarrow B$, then $AC \rightarrow BC$ for any C.
- Axiom of transitivity
  - Same as the transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds.
  - $A \rightarrow B$ is called as A functionally that determines B. If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Secondary Rules

- Union
  - If A→B holds and A→C holds, then A→BC holds.
  - If X→Y and X→Z then X→YZ
- Composition
  - If A→B and X→Y holds, then AX→BY holds.
- Decomposition
  - If A→BC holds then A→B and A→C hold. If X→YZ then X→Y and X→Z
- Pseudo Transitivity
  - If A→B holds and BC→D holds, then AC→D holds. If X→Y and YZ→W then XZ→W.

Why armstrong axioms refer to the Sound and Complete ?

- By sound, we mean that given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using the primary rules of Armstrong axioms holds in every relation state r of R that satisfies the dependencies in F.
- By complete, we mean that using primary rules of Armstrong axioms repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F.

Equivalence of Sets of Functional Dependencies

Definition.

A set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.

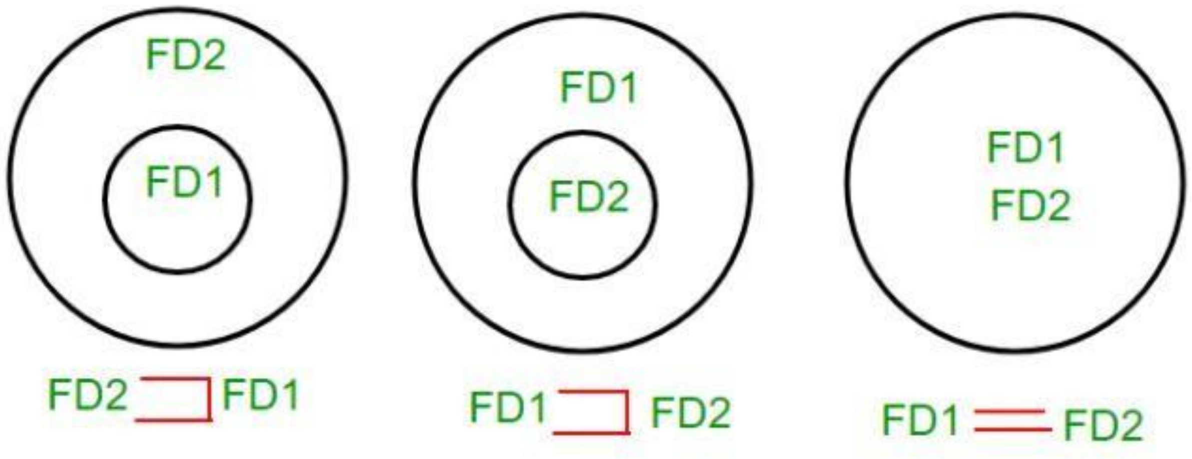
Definition

Two sets of functional dependencies E and F are equivalent if E+ = F+. Therefore, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions—E covers F and F covers E—hold.

- We can determine whether F covers E by calculating X^+ with respect to F for each FD
- $X \rightarrow Y$ in E, and then checking whether this X^+ includes the attributes in Y.
- If this is the case for every FD in E, then F covers E. We determine whether E and F are equivalent by checking that E covers F and F covers E.
- The following two sets of FDs are equivalent:
  - $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ and
  - $G = \{A \rightarrow CD, E \rightarrow AH\}$.

How to find relationship between two FD sets?

- Let FD1 and FD2 are two FD sets for a relation R.
  1. If all FDs of FD1 can be derived from FDs present in FD2, we can say that FD2 ⊃FD1.
  2. If all FDs of FD2 can be derived from FDs present in FD1, we can say that FD1 ⊃FD2.
  3. If 1 and 2 both are true, FD1=FD2.



A relation R(A,B,C,D) having two FD sets FD1 = {A->B, B->C, AB->D} and FD2 = {A->B, B->C, A->C, A->D}

- Step 1. Checking whether all FDs of FD1 are present in FD2
  - A->B in set FD1 is present in set FD2.
  - B->C in set FD1 is also present in set FD2.
  - AB->D in present in set FD1 but not directly in FD2 but we will check whether we can derive it or not. For set FD2, (AB)+ = {A,B,C,D}. It means that AB can functionally determine A, B, C and D. So AB->D will also hold in set FD2.
  - As all FDs in set FD1 also hold in set FD2, FD2 ⊃FD1 is true.

- Step 2. Checking whether all FDs of FD2 are present in FD1
  - A->B in set FD2 is present in set FD1.
  - B->C in set FD2 is also present in set FD1.
  - A->C is present in FD2 but not directly in FD1 but we will check whether we can derive it or not. For set FD1, (A)+ = {A,B,C,D}. It means that A can functionally determine A, B, C and D. SO A->C will also hold in set FD1.
  - A->D is present in FD2 but not directly in FD1 but we will check whether we can derive it or not. For set FD1, (A)+ = {A,B,C,D}. It means that A can functionally determine A, B, C and D. SO A->D will also hold in set FD1.
  - As all FDs in set FD2 also hold in set FD1, FD1 $\supset$FD2 is true.

- Step 3.
  - As FD2 ⊃FD1 and FD1 ⊃FD2 both are true FD2 =FD1 is true.
  - These two FD sets are semantically equivalent.

Minimal Sets of Functional Dependencies

- a minimal cover of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F+ of F.
- this property is lost if any dependency from the set F is removed;
- F must have no redundancies in it, and the dependencies in F are in a standard form
- a minimal set of dependencies as being a set of dependencies in a standard or canonical form and with no redundancies

- To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its right-hand side.
  2. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.
  3. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

- Condition 1 just represents every dependency in a canonical form with a single attribute on the right-hand side.
- Conditions 2 and 3 ensure that there are no redundancies in the dependencies either by having redundant attributes on the left-hand side of a dependency (Condition 2) or by having a dependency that can be inferred from the remaining FDs in F (Condition 3).

Definition. A minimal cover of a set of functional dependencies E is a minimal set of dependencies (in the standard canonical form and without redundancy) that is equivalent to E. We can always find at least one minimal cover F for any set of dependencies E

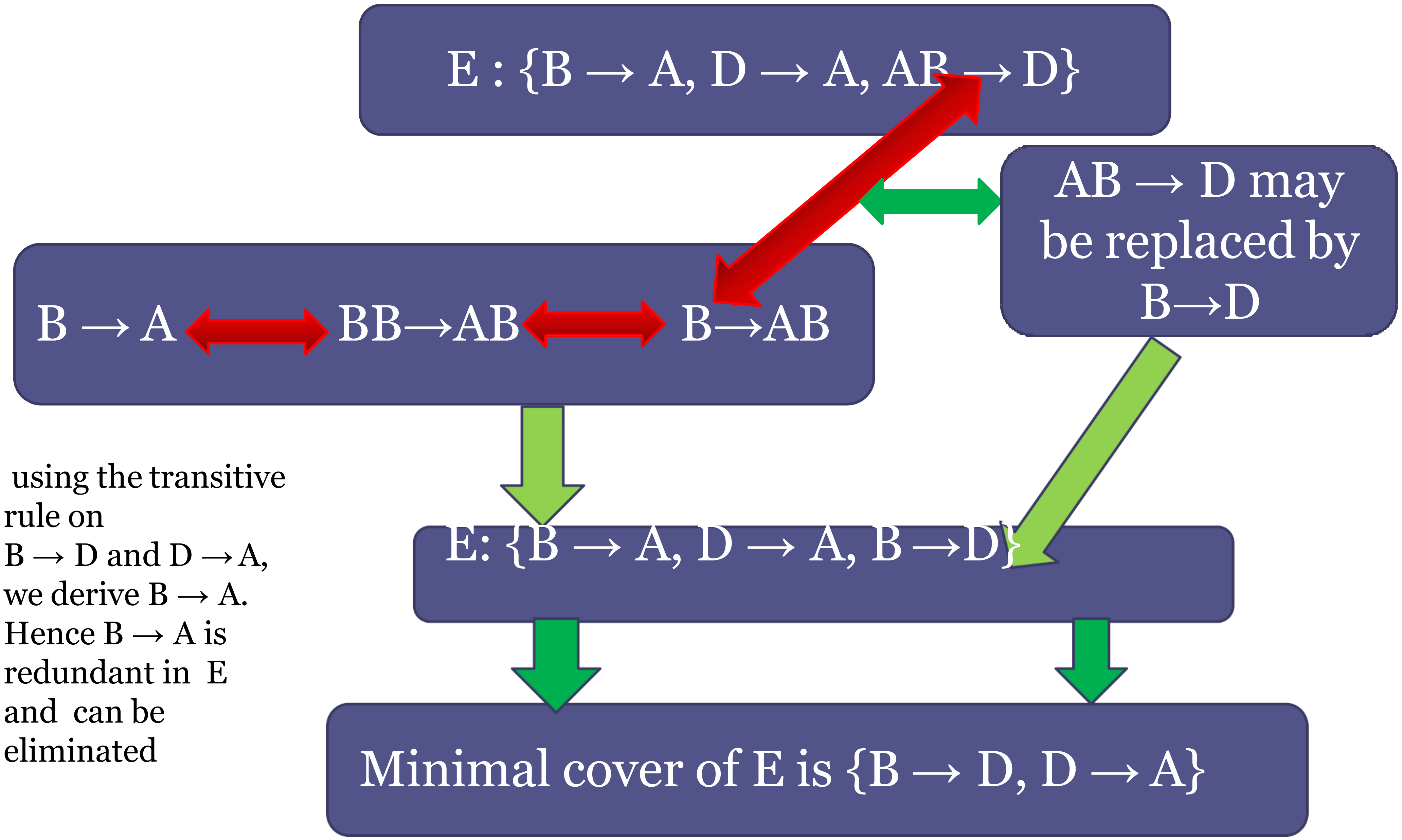
- If several sets of FDs qualify as minimal covers of E by the definition above, it is customary to use additional criteria for minimality.
- For example, we can choose the minimal set with the smallest number of dependencies or with the smallest total length

Example: find the minimal cover of set of FDs be E : {B → A, D → A, AB → D}

- Step 1
  - All above dependencies are in canonical form
  - that is, they have only one attribute on the right-hand side
- Step 2
  - we need to determine if AB → D has any redundant attribute on the left-hand side;
  - that is, can it be replaced by B → D or A → D?
  - Since B → A, by augmenting with B on both sides (IR2), we have BB → AB, or B → AB (i). However, AB → D as given (ii).

- Hence by the transitive rule (IR3), we get from (i) and (ii), B→ D. Thus AB → D may be replaced by B → D.
- We now have a set equivalent to original E, say
- E’: {B → A, D → A, B → D}.
- No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.

• Step 3
  - we look for a redundant FD in E‘.
  - By using the transitive rule on B → D and D → A, we derive B→ A.
  - Hence B → A is redundant in E’ and can be eliminated.
  - Therefore, the minimal cover of E is {B → D, D → A}.



E : {B → A, D → A, AB → D}

AB → D may be replaced by B→D

B → A BB→AB B→AB

E: {B → A, D → A, B →D}

Minimal cover of E is {B → D, D → A}

using the transitive rule on B → D and D → A, we derive B → A. Hence B → A is redundant in E and can be eliminated

Normalization of Relations

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

- 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

Practical Use of Normal Forms

- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect
- The database designers need not normalize to the highest possible normal form
  - usually up to 3NF and BCNF. 4NF rarely used in practice.
- Denormalization:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Definitions of Keys and Attributes Participating in Keys

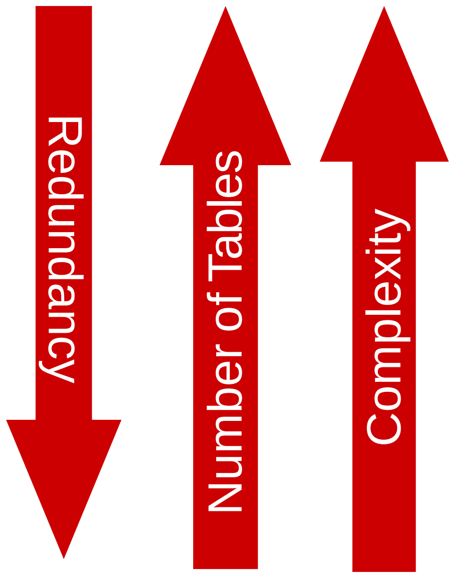
- A **superkey** of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- 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.

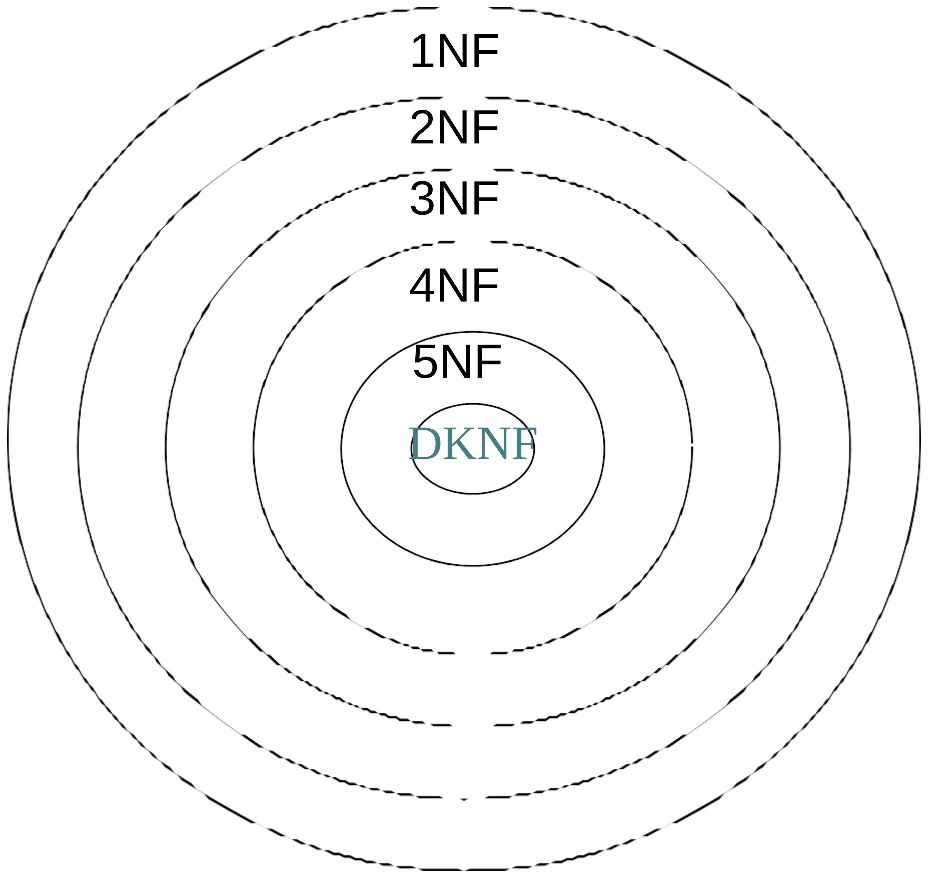
- If a relation schema has more than one key, each is called a **candidate** key.
    - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys.**
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.



Each higher level is a subset of the lower level

First Normal Form (1NF)

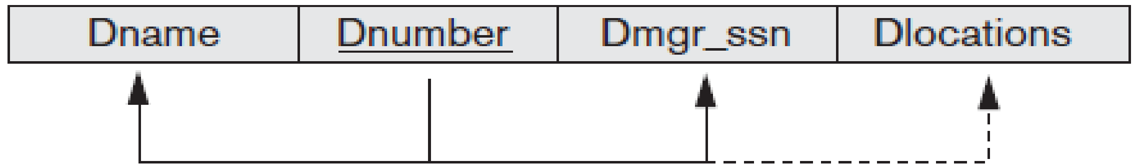
- First normal form enforces these criteria:
  - Eliminate repeating groups in individual tables.
  - Create a separate table for each set of related data.
  - Identify each set of related data with a primary key

•It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.

•It was defined to disallow multivalued attributes, composite attributes, and their combinations

(a)

DEPARTMENT



(b)

DEPARTMENT

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

(c)

DEPARTMENT

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

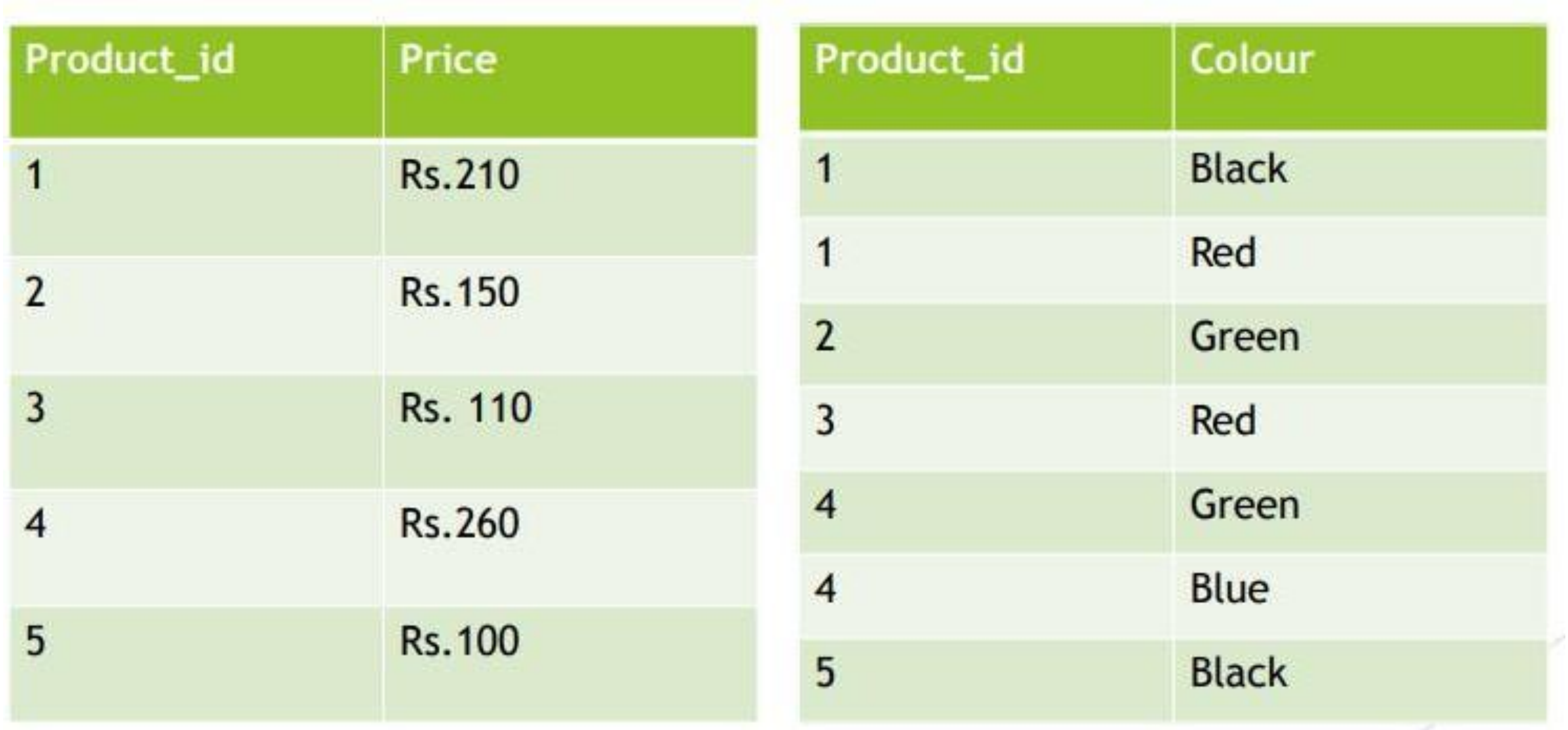
Figure 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Activate Windows
Go to Settings to activa

Table_Product		
Product Id	Colour	Price
1	Black, red	Rs.210
2	Green	Rs.150
3	Red	Rs. 110
4	Green, blue	Rs.260
5	Black	Rs.100

This table is not in first normal form because the “Colour” column contains multiple Values.

After decomposing it into first normal form it looks like:



Product_id	Price
1	Rs.210
2	Rs.150
3	Rs. 110
4	Rs.260
5	Rs.100

Product_id	Colour
1	Black
1	Red
2	Green
3	Red
4	Green
4	Blue
5	Black

Second Normal Form (2NF)

- Second normal form (2NF) is based on the concept of full functional dependency
- ▫ A functional dependency X → Y is a full 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 ε X, (X – {A}) does not functionally determine Y.
- Eg:{Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds).
- However, the dependency {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds.

Second Normal Form (2NF)

- Definition. A relation schema R is in 2NF if every nonprime attribute A in R is <u>fully functionally dependent</u> on the primary key of R.

  OR

- A table is said to be in 2NF if both the following conditions hold:
  - Table is in 1NF (First normal form)
  - No non-prime attribute is dependent on the proper subset of any candidate key of table.
- An attribute that is not part of any candidate key is known as non-prime attribute.

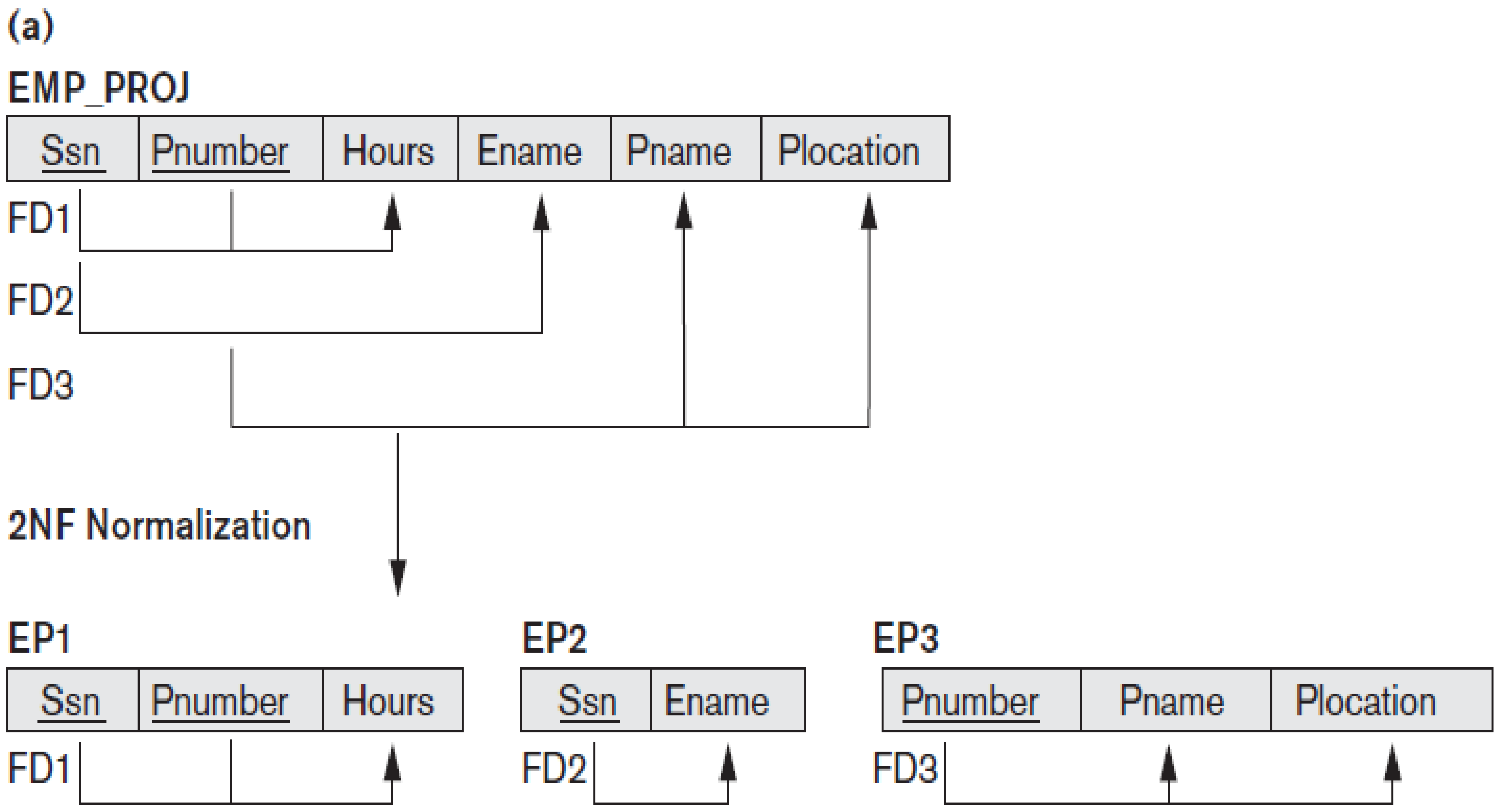

(a)
EMP_PROJ
Ssn
Pnumber
Hours
Ename
Pname
Plocation
FD1
FD2
FD3
2NF Normalization
EP1
Ssn
Pnumber
Hours
FD1
EP2
Ssn
Ename
FD2
EP3
Pnumber
Pname
Plocation
FD3

Table purchase detail

Customer_id	Store_id	Location
1	1	Patna
1	3	Noida
2	1	Patna
3	2	Delhi
4	3	Noida

- This table has a composite primary key i.e. customer id, store id. The non key attribute is location. In this case location depends on store id, which is part of the primary key.

After decomposing it into second normal form it looks like:

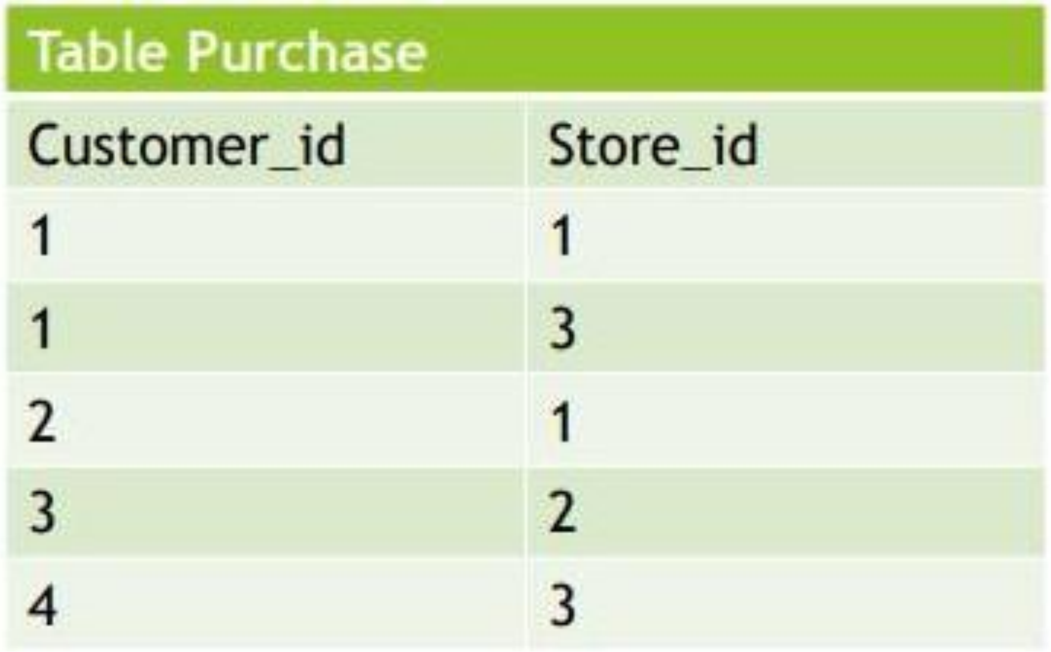


Table Purchase	
Customer_id	Store_id
1	1
1	3
2	1
3	2
4	3

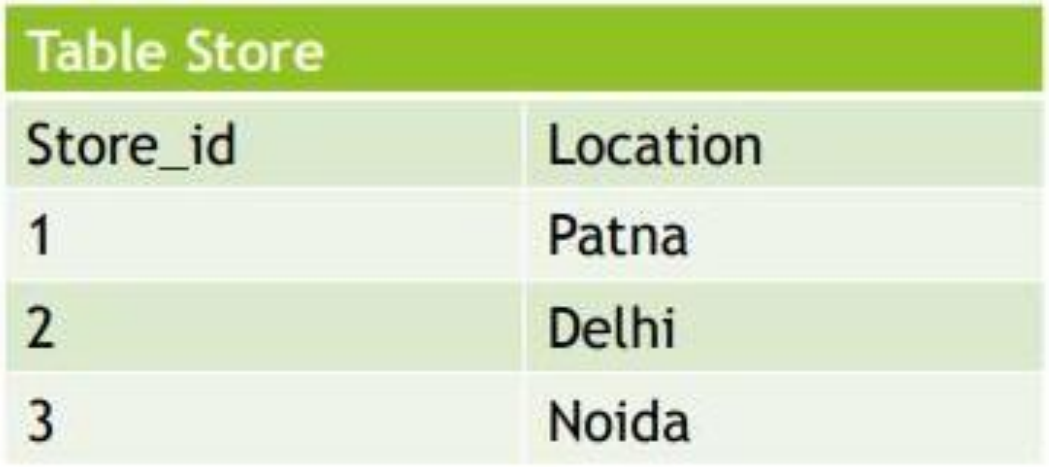
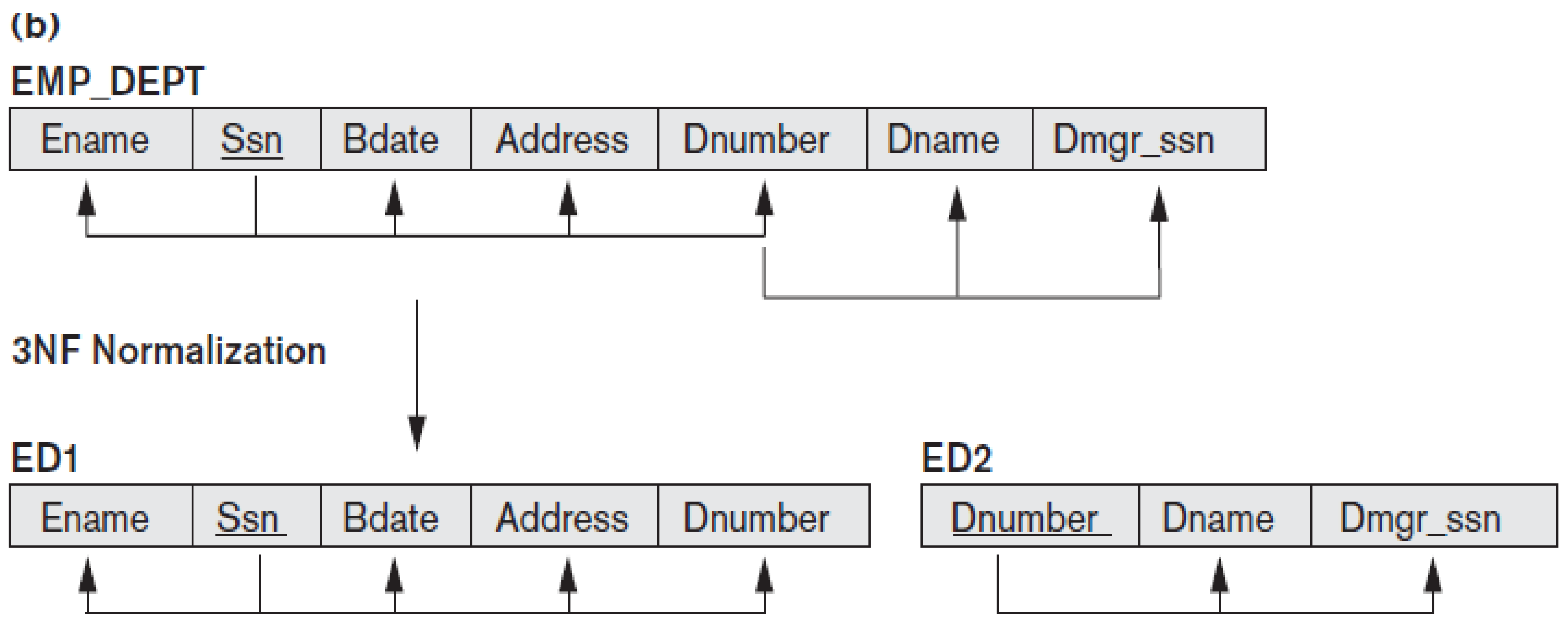


Table Store	
Store_id	Location
1	Patna
2	Delhi
3	Noida

Third Normal Form (3NF)

- A table design is said to be in 3NF if both the following conditions hold:
  - Table must be in 2NF
  - no nonprime attribute of R is transitively dependent

  on the primary key.
- An attribute that is not part of any candidate key is known as non-prime attribute.
- In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:
  - X is a super key of R
  - Y is a prime attribute of R
- An attribute that is a part of one of the candidate keys is known as prime attribute.



Normalizing EMP_DEPT into 3NF relations.

Table Book Details

Bood_id	Genre_id	Genre type	Price
1	1	Fiction	100
2	2	Sports	110
3	1	Fiction	120
4	3	Travel	130
5	2	sports	140

- In the table, book_id determines genre_id and genre_id determines genre type. Therefore book_idd determines genre type via genre_id and we have transitive functional dependency.

After decomposing it into third normal form it looks like:

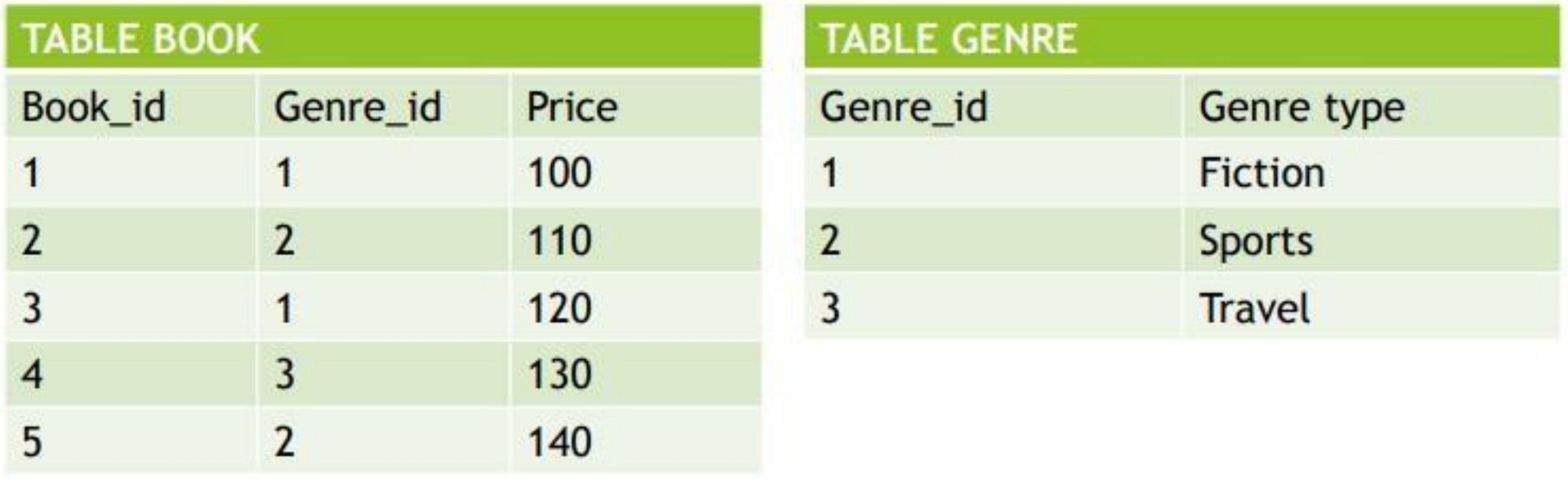


TABLE BOOK

Book_id	Genre_id	Price
1	1	100
2	2	110
3	1	120
4	3	130
5	2	140

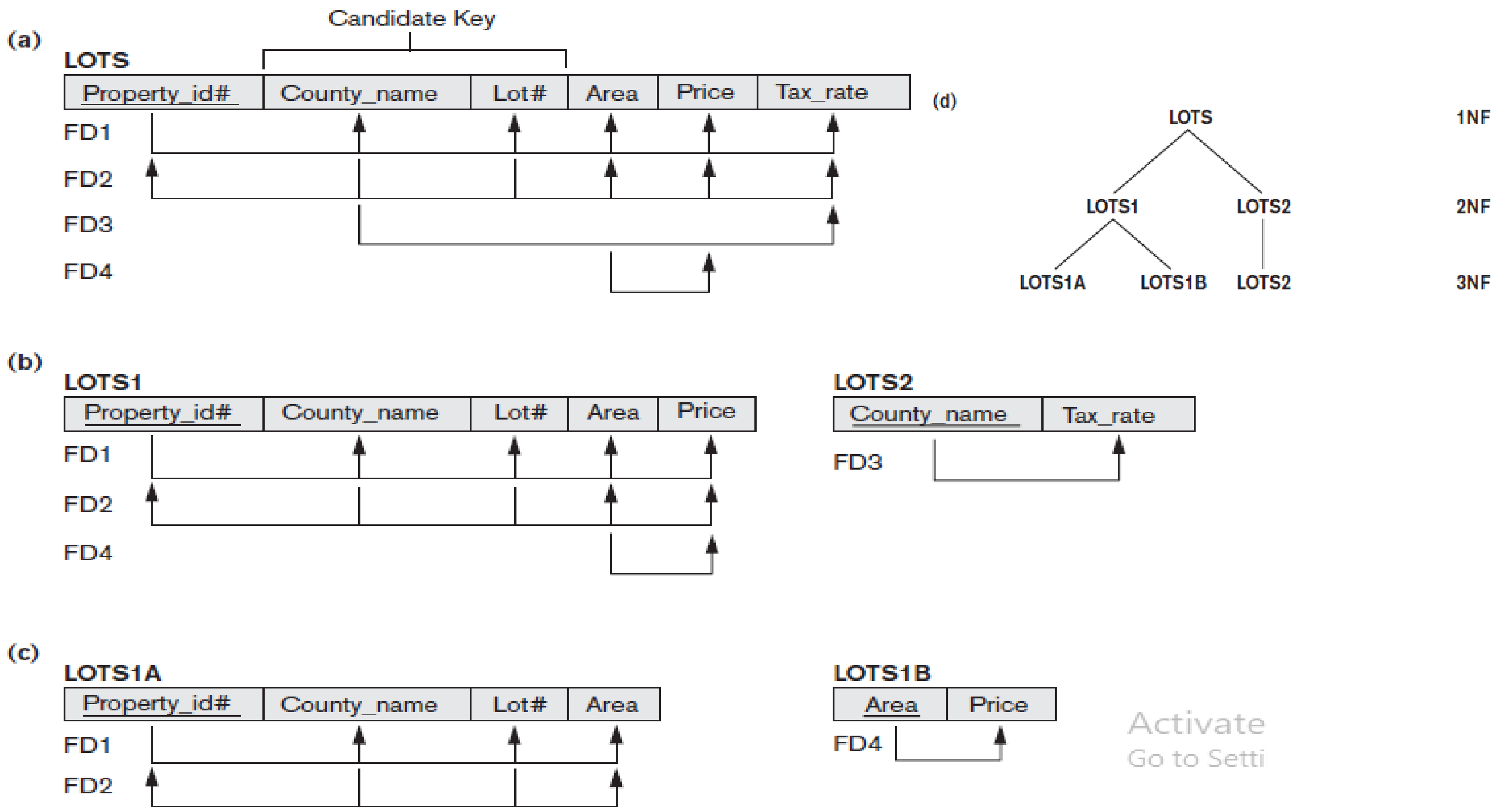
TABLE GENRE

Genre_id	Genre type
1	Fiction
2	Sports
3	Travel

Table 15.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Figure 15.12
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.



Activate
Go to Setti

Boyce-Codd Normal Form (BCNF)

- It is an advance version of 3NF that's why it is also referred as 3.5NF.
- BCNF is stricter than 3NF.
- A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of R.

Additional functional dependency FD5:Area→County_name. If we add this to the other dependencies, the relation schema LOTS1A still is in 3NF because County_name is a prime attribute.

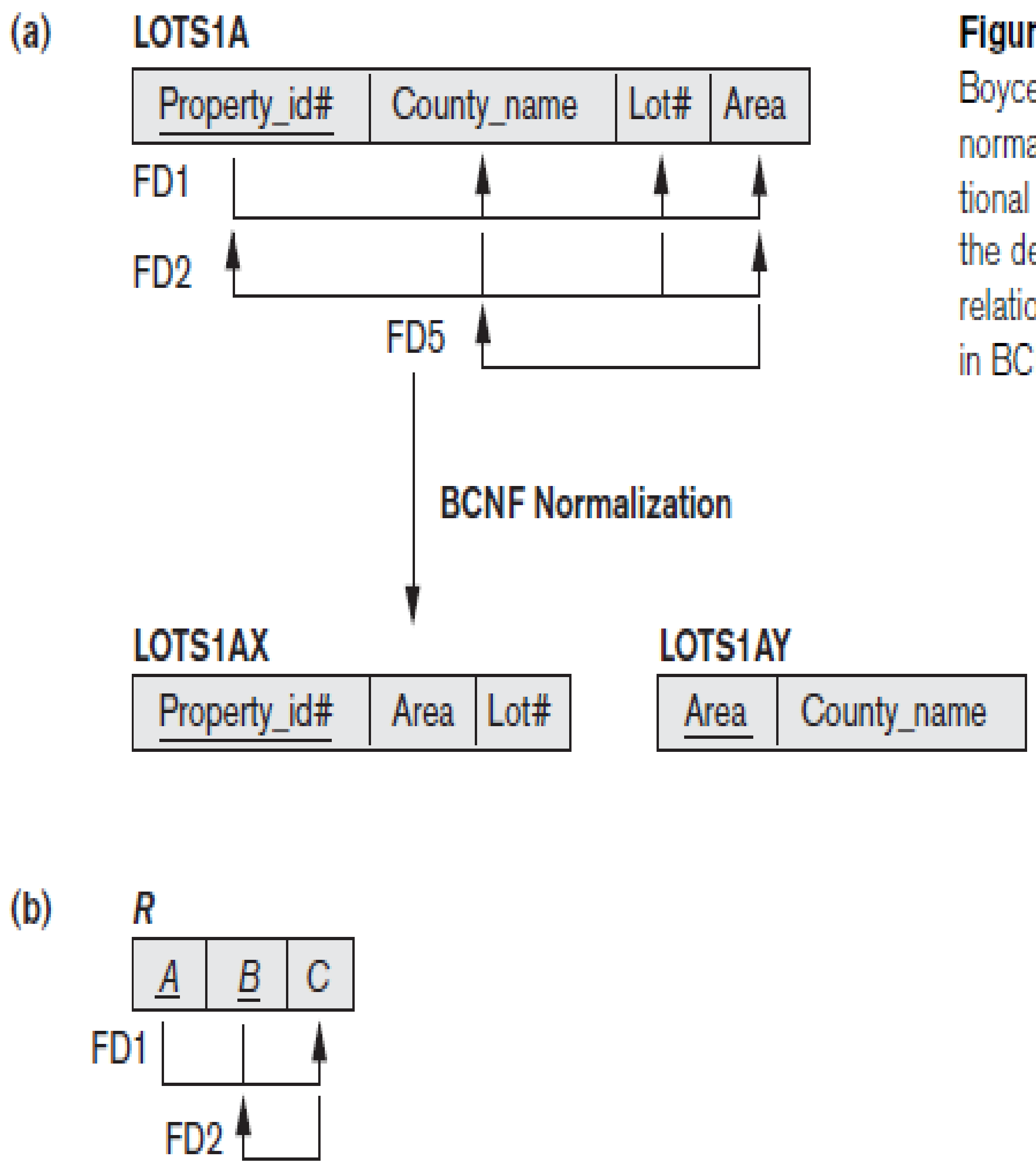


Figure 15.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Activate W

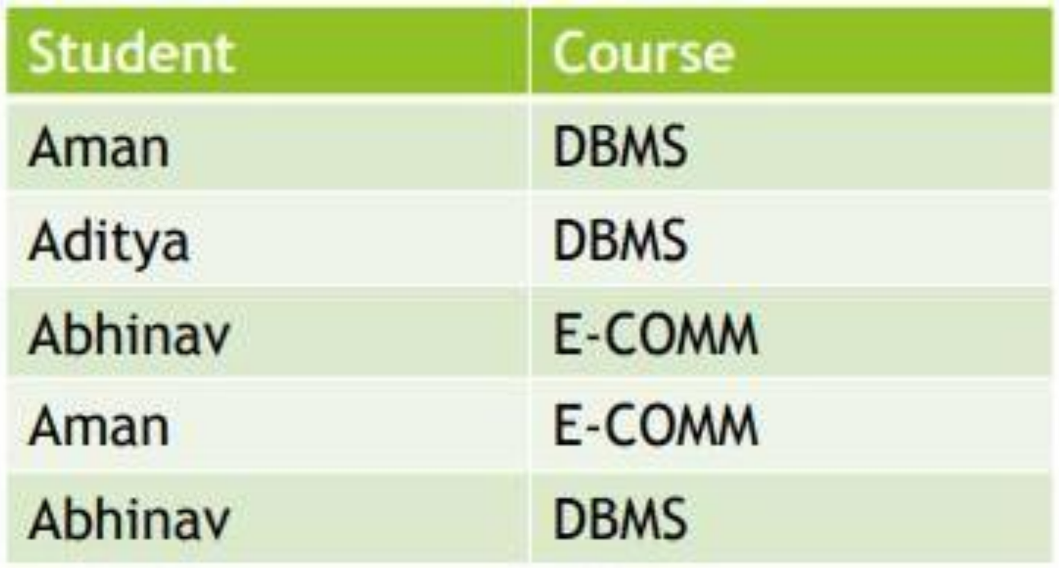
Student	Course	Teacher
Aman	DBMS	AYUSH
Aditya	DBMS	RAJ
Abhinav	E-COMM	RAHUL
Aman	E-COMM	RAHUL
abhinav	DBMS	RAJ

- KEY: {Student, Course}
- Functional dependency

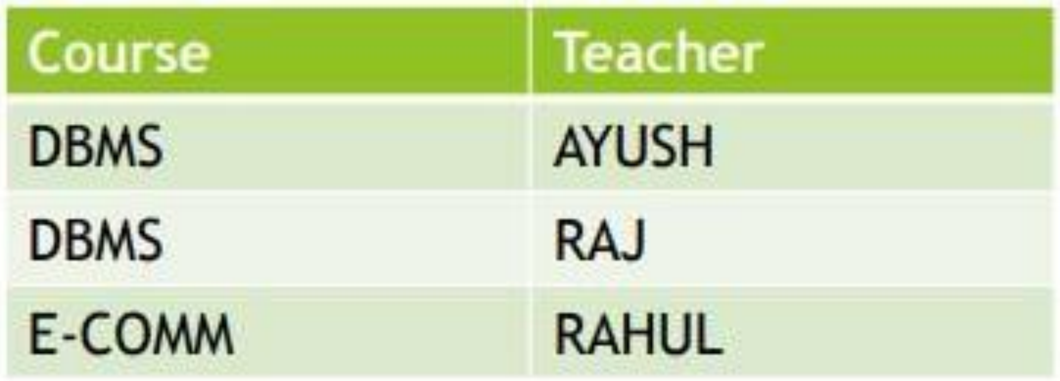
  {student, course} -> Teacher

  Teacher-> Course
- Problem: teacher is not superkey but determines course.

After decomposing it into Boyce-Codd normal form it looks like



Student	Course
Aman	DBMS
Aditya	DBMS
Abhinav	E-COMM
Aman	E-COMM
Abhinav	DBMS



Course	Teacher
DBMS	AYUSH
DBMS	RAJ
E-COMM	RAHUL

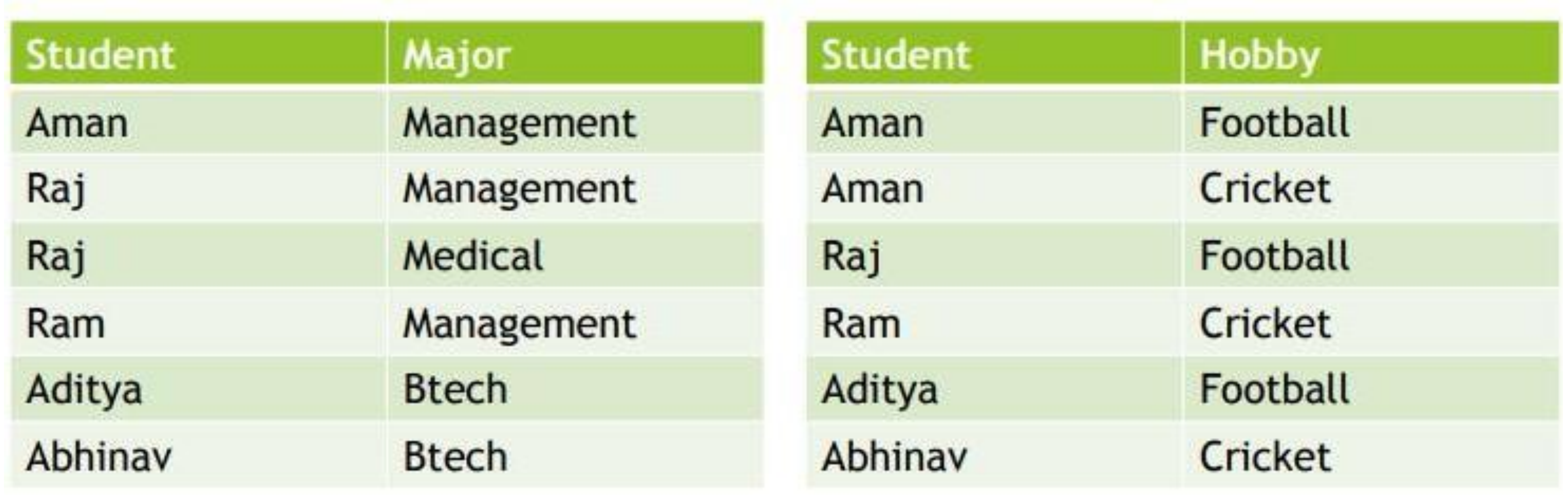
Fourth Normal Form (4NF)

- Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key.
- It builds on the first three normal forms (1NF, 2NF and 3NF) and the BoyceCodd Normal Form (BCNF).
- It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Student	Major	Hobby
Aman	Management	Football
Aman	Management	Cricket
Raj	Management	Football
Raj	Medical	Football
Ram	Management	Cricket
Aditya	Btech	Football
Abhinav	Btech	Cricket

- Key: {students, major, hobby}
- MVD: ->-> Major, hobby

After decomposing it into fourth normal form it looks like:



Student	Major
Aman	Management
Raj	Management
Raj	Medical
Ram	Management
Aditya	Btech
Abhinav	Btech

Student	Hobby
Aman	Football
Aman	Cricket
Raj	Football
Ram	Cricket
Aditya	Football
Abhinav	Cricket

Fifth Normal Form (5NF)

- A database is said to be in 5NF, if and only if,
  - It's in 4NF.
  - If we can decompose table further to eliminate redundancy and anomaly, and
  - when we re-join the decomposed tables by means of candidate keys,
  - we should not be losing the original data or any new record set should not arise.
- In simple words, joining two or more decomposed table should not lose records nor create new records.

Seller	Company	Product
Aman	Coca cola company	Thumps Up
Aditya	Unilever	Ponds
Aditya	Unilever	Axe
Aditya	Uniliver	Lakme
Abhinav	P&G	Vicks
Abhinav	Pepsico	Pepsi

- Key: {seller, company, product}
- MVD: Seller ->-> Company, product

  Product is related to company.

After decomposing it into fifth normal form it looks like:

Seller	Product
Aman	Thumps Up
Aditya	Ponds
Aditya	Axe
Aditya	Lakme
Abhinav	Vicks
Abhinav	Pepsi

Seller	Company
Aman	Coca cola company
Aditya	Unilever
Abhinav	P&G
Abhinav	Pepsico

Company	Product
Coca cola company	Thumps Up
Unilever	Ponds
Unilever	Axe
Unilever	Lakme
Pepsico	Pepsi
P&G	Vicks

Properties of Relational Decompositions

- Universal relation schema R = {A1, A2, …, An} :includes all the attributes of the database.
- We implicitly make the universal relation assumption, which states that every attribute name is unique.
- The set F of functional dependencies that should hold on the attributes of R is specified by the database designers and is made available to the design algorithms.
- Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas D= {R1, R2, …, Rm} that will become the relational database schema; D is called a decomposition of R.

- Attribute preservation
  - Make sure that each attribute in R will appear in at least one relation schema *Ri* in the decomposition so that no attributes are lost; This is called the attribute preservation condition of a decomposition.

    •Each individual relation Ri in the decomposition D be in BCNF or 3NF.

    •Consider the EMP_LOCS(Ename, Plocation). Joining EMP_LOCS with PROJECT(Pname, Pnumber, Plocation,Dnum) which is in BCNF—using Plocation as a joining attribute also gives rise to spurious tuples.

    •So we need other criteria, together with the conditions of 3NF or BCNF, prevent such bad designs

Dependency Preservation Property of a Decomposition

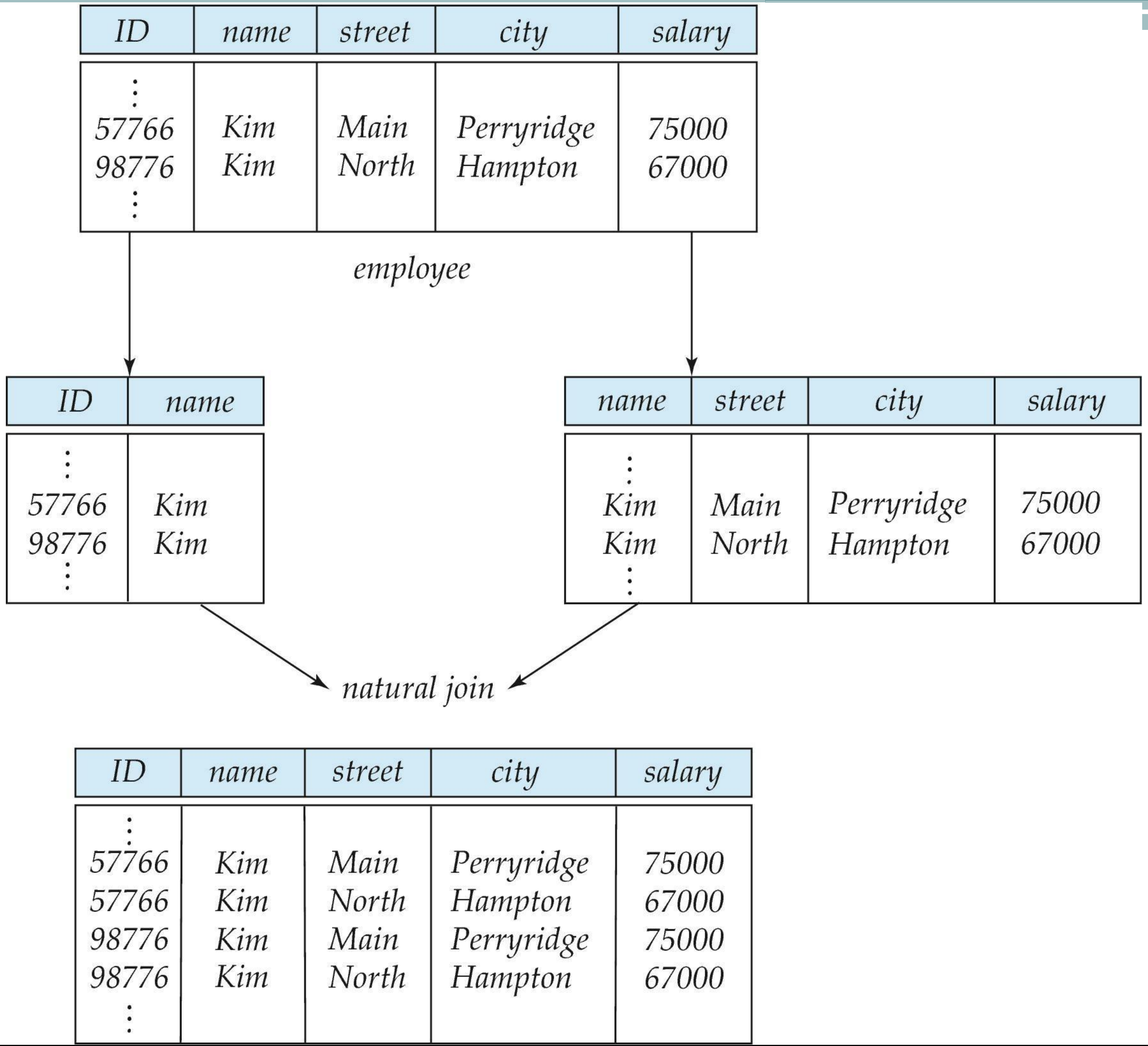
Definition. Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $\pi_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i. Hence, the projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left- and right-hand-side attributes are in R_i. We say that a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is **dependency-preserving** with respect to F if the union of the projections of F on each R_i in D is equivalent to F; that is, $((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$.

Lossy Decomposition

- Not all decompositions are good. Suppose we decompose *employee(ID, name, street, city, salary)* into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street*, *city*, *salary*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is 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

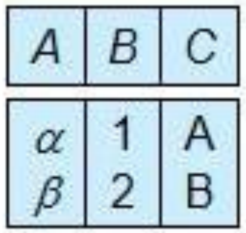
ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$

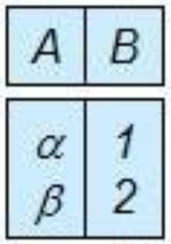
  $R_1 = (A, B)$ $R_2 = (B, C)$

In general decomposition is lossless provided certain functional dependencies hold; more on this later.



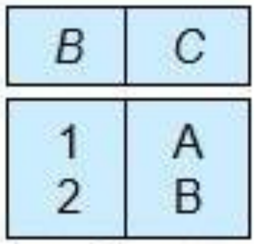
A	B	C
α	1	A
β	2	B

r



A	B
α	1
β	2

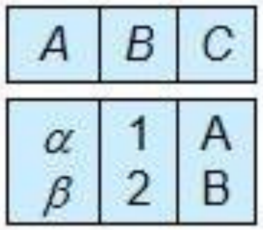
$\Pi_{A,B}(r)$



B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$



A	B	C
α	1	A
β	2	B

Algorithm For Testing for Lossless Join Property

- **Algorithm : Testing for Lossless Join Property**
  - **Input**: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.
2. Set S(i,j):=bij for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j) *).
3. For each row i representing relation schema Ri

```
For each row i representing relation schema Ri
        {for each column j representing attribute Aj
            {if (relation Ri includes attribute Aj) then set S(i,j):= aj;};};
```

  - (* each aj is a distinct symbol associated with index (j) *)

4. Repeat the following loop until a complete loop execution results in no changes to S

```
{for each functional dependency X γY in F
    {for all rows in S *which have the same symbols* in the columns corresponding to attributes in X
        {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:
            If any of the rows has an “a” symbol for the column, set the other rows to that *same* “a” symbol in the column.
            If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appear in one of the rows for the attribute and set the other rows to that same “b” symbol in the column ;};
    };
};
```

5. If a row is made up entirely of “a” symbols, then the decomposition has the lossless join property; otherwise it does not.

Nonadditive join test for *n*-ary decompositions. (a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test. (b) A decomposition of EMP_PROJ that has the lossless join property. (c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(a) $R = \{\text{Ssn, Ename, Pnumber, Pname, Plocation, Hours}\}$ $\qquad D = \{R_1, R_2\}$

$R_1 = \text{EMP\_LOCS} = \{\text{Ename, Plocation}\}$

$R_2 = \text{EMP\_PROJ1} = \{\text{Ssn, Pnumber, Hours, Pname, Plocation}\}$

$F = \{\text{Ssn} \rightarrow \text{Ename; Pnumber} \rightarrow \{\text{Pname, Plocation}\}; \{\text{Ssn, Pnumber}\} \rightarrow \text{Hours}\}$

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(No changes to matrix after applying functional dependencies)

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

(c)

R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2, R_3\}$
R_1 = EMP = {Ssn, Ename}
R_2 = PROJ = {Pnumber, Pname, Plocation}
R_3 = WORKS_ON = {Ssn, Pnumber, Hours}

F = {Ssn $\rightarrow$ Ename; Pnumber $\rightarrow$ {Pname, Plocation}; {Ssn, Pnumber} $\rightarrow$ Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matrix S after applying the first two functional dependencies;
last row is all "a" symbols so we stop)

Activate Windows
Go to Settings to activa

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A [illegible]is **dependency preserving**, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.
- See book for efficient algorithm for checking dependency preservation

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of R, that is, it is a superkey of R.
- **Simplified test**: To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF.
- However, **simplified test using only *F* is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$
    - Decompose R into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
    - Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
  - or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency
    $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$
    can be shown to hold on R_i, and R_i violates BCNF.
  - We use above dependency to decompose R_i

E.g. given { $A \rightarrow B$, $BC \rightarrow D$} and decomposition $R1$ (A,B) and $R2$ (A,C,D, E), A+ = ABC, so R2 violates BCNF due to the dependency A → BC

BCNF Decomposition Algorithm

```
$result$ := $\{R\}$;
$done$ := false;
compute $F^+$;
**while (not** $done$) **do**
  **if** (there is a schema $R_i$ in $result$ that is not in BCNF)
    **then begin**
          let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
              holds on $R_i$ such that $\alpha \rightarrow R_i$ is not in $F^+$,
             and $\alpha \cap \beta = \emptyset$;
           $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
          **end**
    **else** $done$ := **true;**
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A,B)$

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

- *course* is in BCNF
  - How do we knowthis?
- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

BCNF and Dependency Preservation

- It is not always possible to get a BCNF decomposition that is dependency preserving
- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join

References

- https://www.geeksforgeeks.org/armstrongs-axioms-in-functional-dependency-in-dbms/