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), Boyce 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:
  1. Making sure that the semantics of the attributes is clear in the schema
  2. Reducing the redundant information in tuples
  3. Reducing the NULL values in tuples
  4. 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

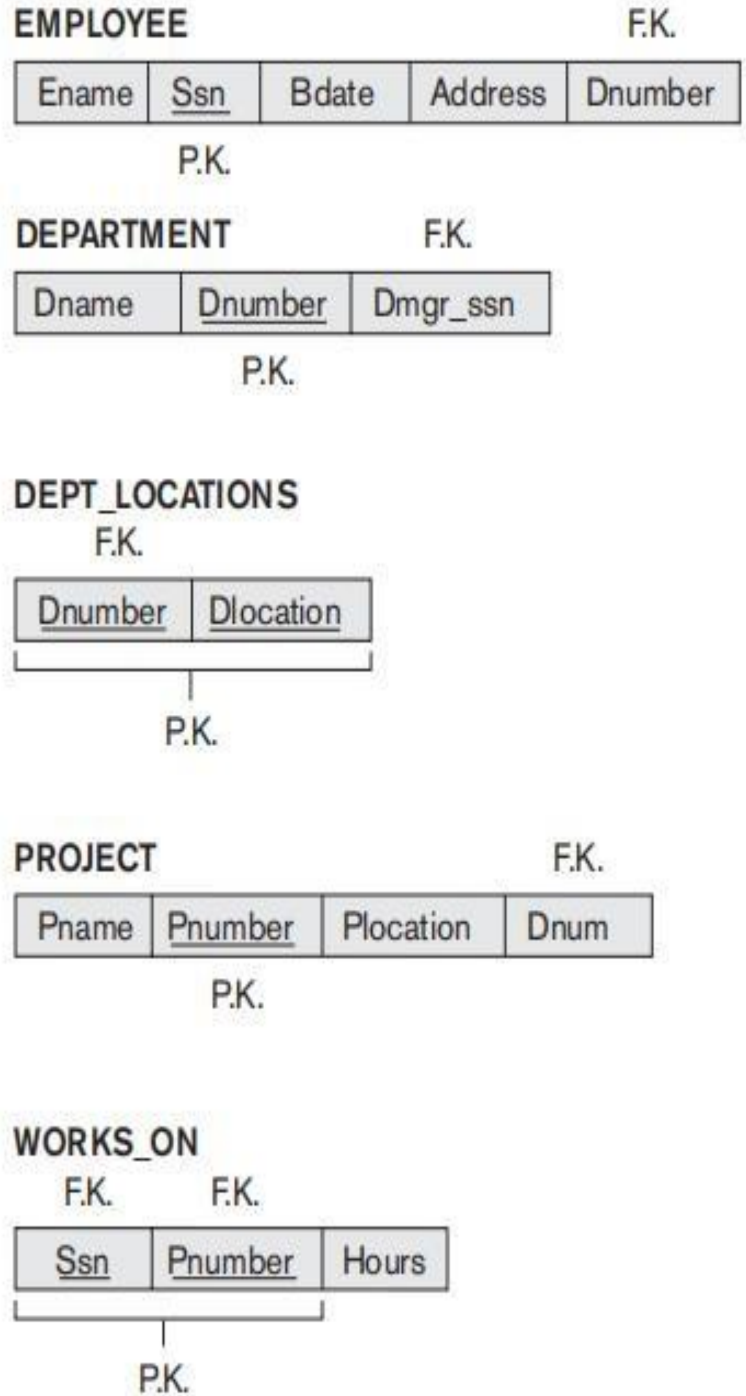


Figure 15.1
A simplified COMPANY relational database schema.

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

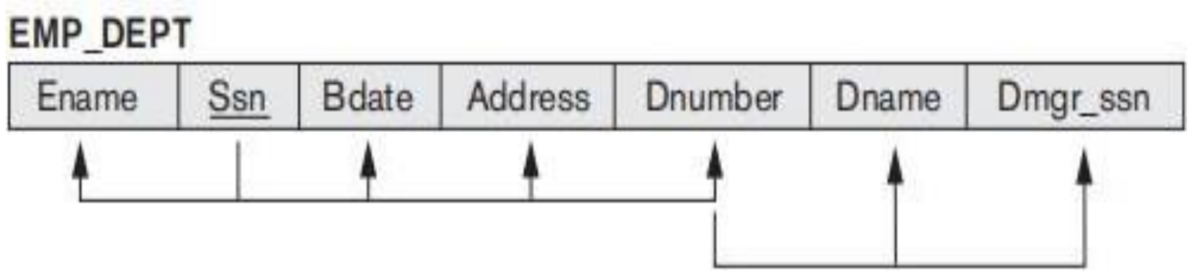
PROJECT

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.

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

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

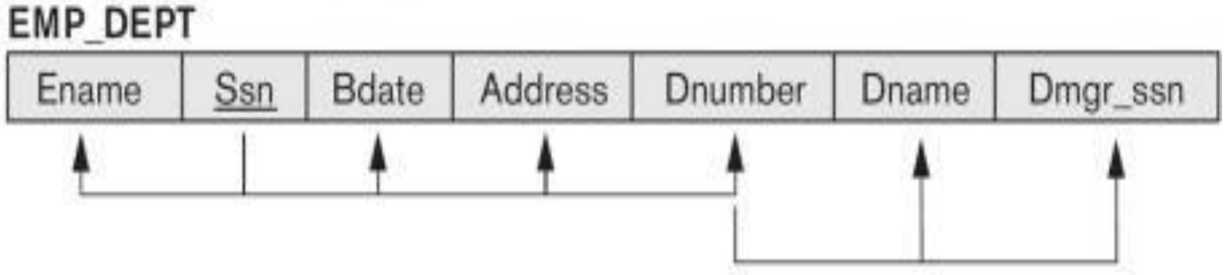
Redundant Information in Tuples and Update Anomalies

- 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)

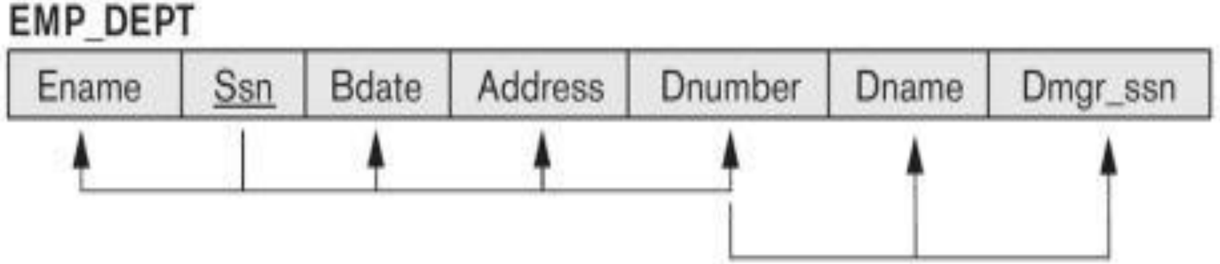


insertion anomalies: when adding an employee, we must assign them to a department or else use NULLs. When adding a new department with no employees, we have to use NULLs for the employee Ssn, which is supposed to be the primary key!

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 (Dname, Dmgr_ssn)

EMP_DEPT

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

Redundancy (Ename) Redundancy (Pname, Plocation)

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

Ename	Plocation

P.K. (Ename, Plocation)

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

P.K. (Ssn, Pnumber)

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

Functional dependencies

- 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;
- The whole database is described by a single **universal** relation schema R = {A1, A2, … , An}.

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].

This means that the values of the Y component of a tuple in r depend on the values of the X component;
alternatively, the values of the X component of a tuple uniquely (or **functionally**) determine the values of the Y component.

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

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

A	B
a1	b1
a2	b3
a1	b2
a2	b3

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

✗ a1 (b1,b2)

✓ a2 b3

$B \to A$

✓ b1 a1

✓ b3 a2

✓ b2 a1

So this is a valid FD

$B \to A$ 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 primary key

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 $A \rightarrow A$
   - Not a useful FD since we are not getting any important information here

The examples of trivial functional dependencies are-

$AB \longrightarrow A$

$AB \longrightarrow B$

$AB \longrightarrow AB$

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}

The examples of non-trivial functional dependencies are-

$AB \rightarrow BC$

$AB \rightarrow CD$

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 $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
  - 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→B, B→C, C→D,D→A}
- Closure of A, A^+=attribute which can be determined from A

(ie using closure if we can cover all attribute then it is called Candidate Key CK)

  - A^+=ABCD ✔ CK
  - B^+=BCDA ✔ CK
  - C^+=CDAB ✔ CK
  - D^+=DABC

- Candidate Keys of R are A,B,C,D

GATE Question: Consider the relation scheme R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R. What is the key for R?

A. {E,F}

B. {E, F, H}

C. {E, F, H, K, L}

D. {E}

Answer: Finding attribute closure of all given options, we get:

{E,F}+ = {EFGIJ}

{E,F,H}+ = {EFHGIJKLMN}

{E,F,H,K,L}+ = {EFHGIJKLMN}

{E}+ = {E}

{EFH}+ and {EFHKL}+ results in set of all attributes, but EFH is minimal. So it will be candidate key. So correct option is (B).

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.

Secondary Rules

- Union
  - If $A \rightarrow B$ holds and $A \rightarrow C$ holds, then $A \rightarrow BC$ holds.
- Composition
  - If $A \rightarrow B$ and $X \rightarrow Y$ holds, then $AX \rightarrow BY$ holds.
- Decomposition
  - If $A \rightarrow BC$ holds then $A \rightarrow B$ and $A \rightarrow C$ holds
- Pseudo Transitivity
  - If $A \rightarrow B$ holds and $BC \rightarrow D$ holds, then $AC \rightarrow D$ holds.

Equivalence of Sets of Functional Dependencies

Definition.

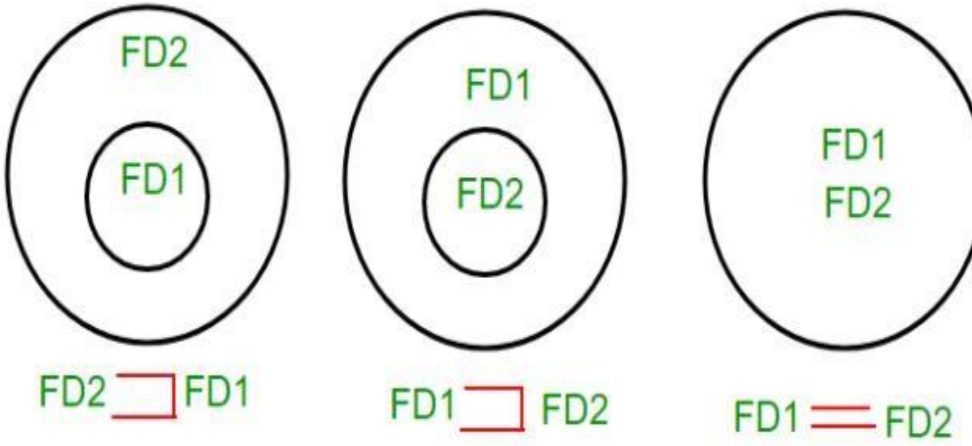
A set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.

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.

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.



Steps to find the Equivalence of sets of Functional Dependencies

Consider 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 is equivalent to FD1 .
  - These two FD sets are semantically equivalent as $FD1^{+} = FD2^{+}$.

Minimal cover of a set 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 (eg: A->C and AB->C)
- 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 $X \longrightarrow A$ in F with a dependency $Y \longrightarrow A$, 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.

Note: 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 is given (ii).

- Hence by the transitive rule (IR3), we get if (i) B → AB and (ii) AB → D , then 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}.

Normalization of Relations

- **Normalization of data** can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

  (1) minimizing redundancy and

  (2)minimizing the insertion, deletion, and update anomalies.
- It can be considered as a "filtering" process to make the design have successively better quality.
- Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- The process proceeds in a top-down fashion .

Normalization of Relations

- First proposed by Codd.
- Codd proposed three normal forms initially, called first, second, and third normal form.
- A **stronger definition of 3NF called Boyce-Codd normal** form (**BCNF**) was proposed later by Boyce and Codd.
- All these normal forms are based on a single analytical tool: **the functional dependencies** among the attributes of a relation.
- Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of **multivalued dependencies** and **join dependencies**, respectively.
- **Definition:** The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.
- **Denormalization:**
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

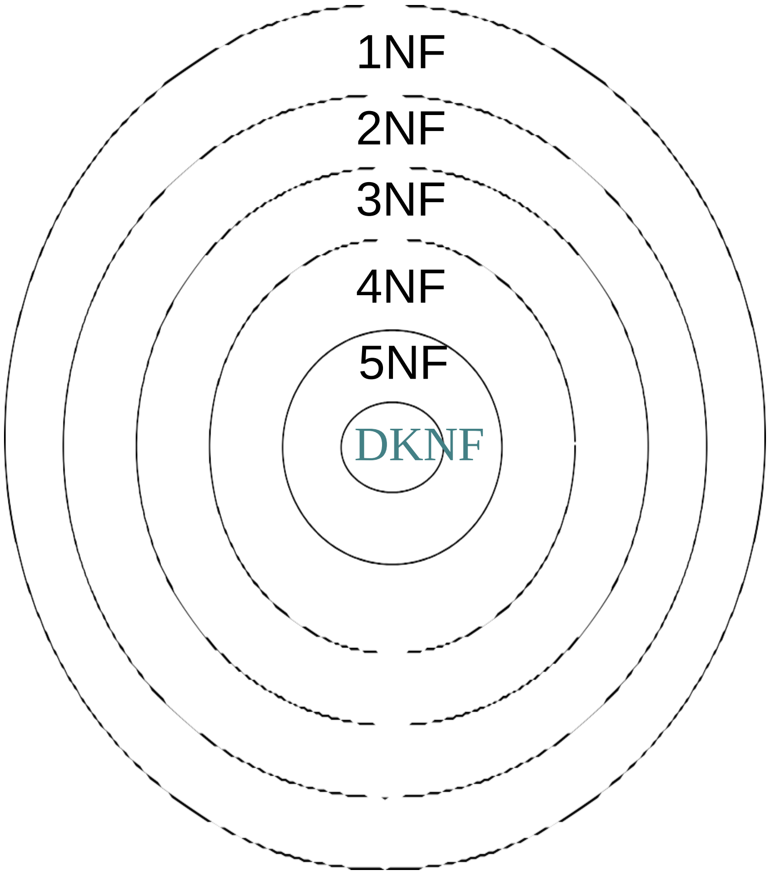
Practical Use of Normal Forms

- 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 rare, and hard to understand or to detect by the database designers and users.
- Thus, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.
- The database designers need not normalize to the highest possible normal form.
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons.

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)

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

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.

Remove the attribute Colour that violates 1NF and place it in a separate relation along with the primary key **Product Id** of **Table_Product**.

After decomposing it into First normal form

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

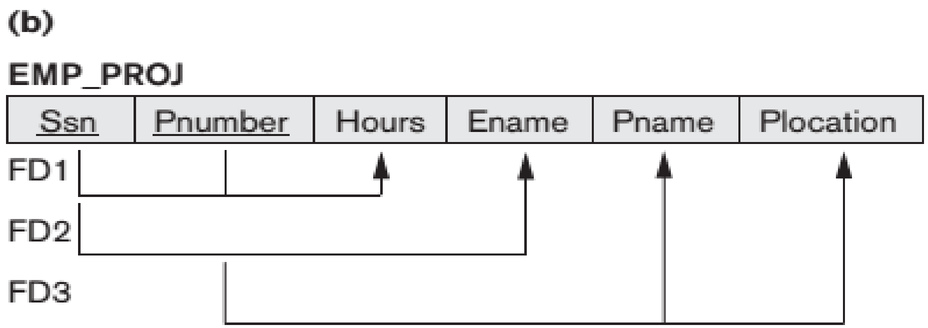
Second Normal Form (2NF)

- A table is said to be in 2NF if both the following conditions hold:
  - Table is in 1NF (First normal form)
  - A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.
- An attribute that is not part of any candidate key is known as non-prime attribute.

- A 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 any more;
- A functional dependency $X \rightarrow Y$ is a **partial dependency** if some attribute $A \varepsilon X$ can be removed from X and the dependency still holds; that is, for some $A \varepsilon X$, $(X - \{A\}) \rightarrow Y$.

Second Normal Form

- Consider the EMP_PROJ relation:
- Here the candidate key is {Ssn,Pnumber}



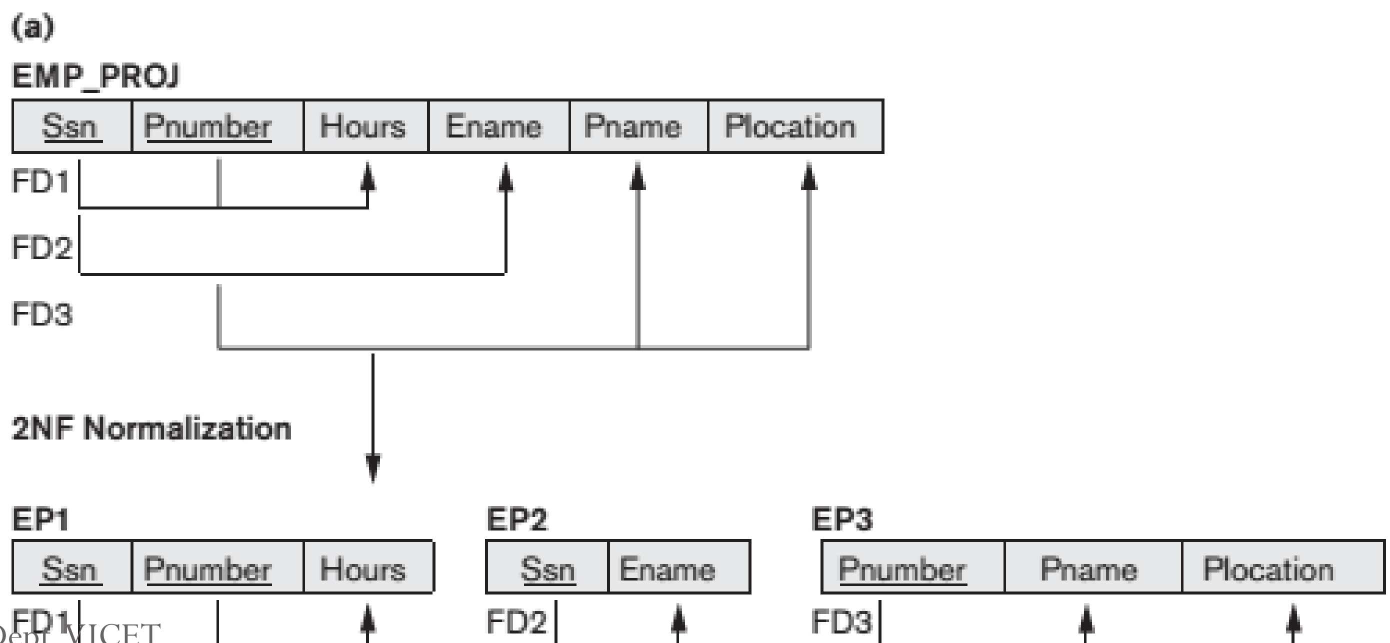
- {Ssn,Pnumber}→Hours is a **full dependency** (neither Ssn → Hours nor Pnumber → Hours holds).

- The dependency {Ssn}→Ename is a **partial dependency**.
- Similarly {Pnumber}→Pname,Plocation is also **partial dependency** .

- EMP_PROJ is in1NF but is not in 2NF. The nonprime attribute Ename violates 2NF because of FD2, and nonprime attributes Pname and Plocation violates 2NF because of FD3.
- The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key {Ssn, Pnumber} of EMP_PROJ, thus violating the 2NF test.

Second Normal Form

- If a relation schema is not in 2NF, it can be 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.



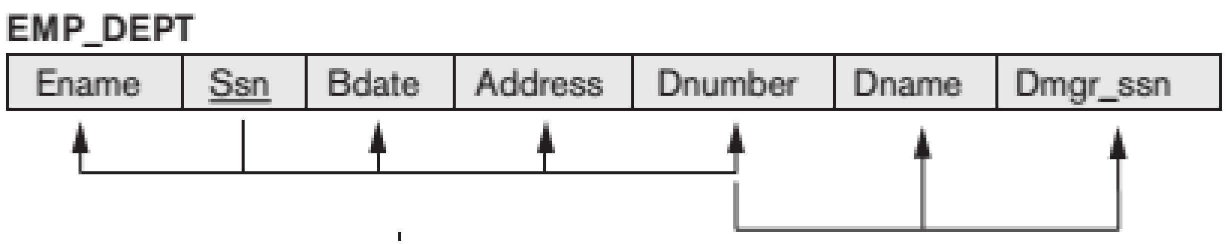
Third Normal Form

- **Third normal form (3NF)** is based on the concept of transitive dependency.
- A functional dependency X⟶Y in a relation schema R is a **transitive dependency** if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X → Z and Z →Y hold.
- **Definition1 :** A relation schema R is in **3NF** if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.
- **Defenition2 :** 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 table
  - Y is a prime attribute of table

Third Normal Form

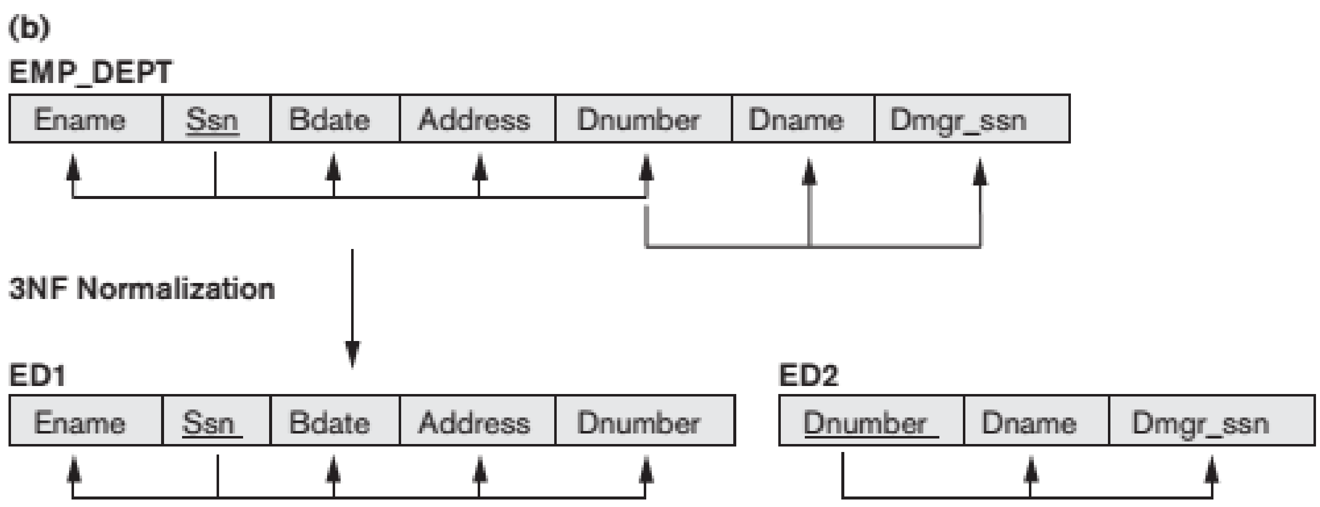
- **Example 1:** Consider EMP_DEPT Relation
- The dependency Ssn → Dmgr_ssn is transitive through Dnumber in EMP_DEPT, because both the dependencies Ssn → Dnumber and Dnumber → Dmgr_ssn hold *and* Dnumber is neither a key itself nor a subset of the key of EMP_DEPT.



- The dependency of Dmgr_ssn on Dnumber is undesirable in EMP_DEPT since Dnumber is not a key of EMP_DEPT.

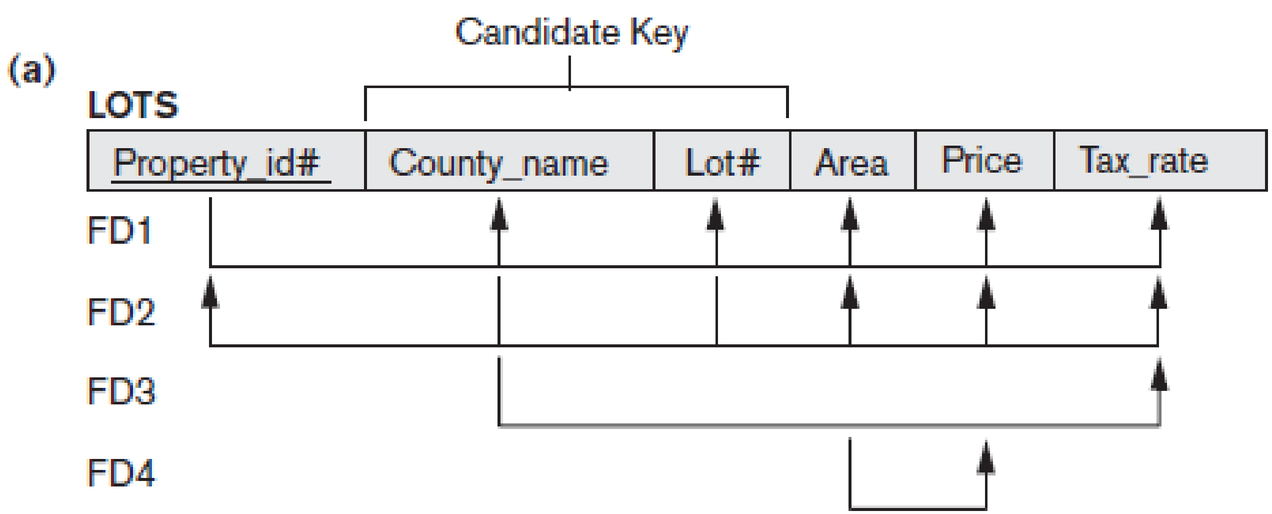
Third Normal Form

- The relation EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber.



Example 2:

- Consider the relation schema LOTS, which describes parcels of land for sale in various counties of a state.
- Suppose that there are **two candidate keys**: **Property_id#** and {**County_name,Lot#**}; that is, Lot numbers are unique only within each county, but Property_id# numbers are unique across counties for the entire state.

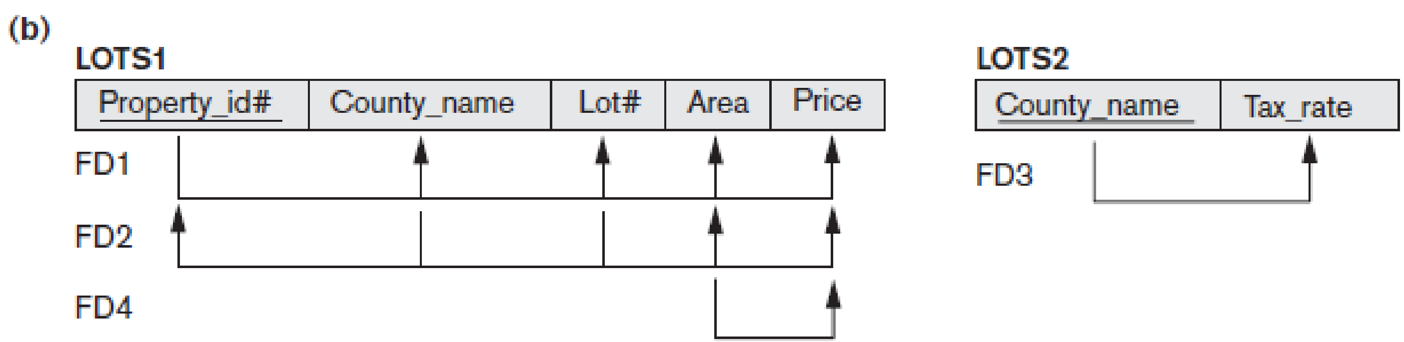


- Based on the two candidate keys Property_id# and {County_name, Lot#}, the functional dependencies FD1 and FD2 hold.
- We choose Property_id# as the primary key, so it is underlined.
- Suppose that the following two additional functional dependencies hold in LOTS:

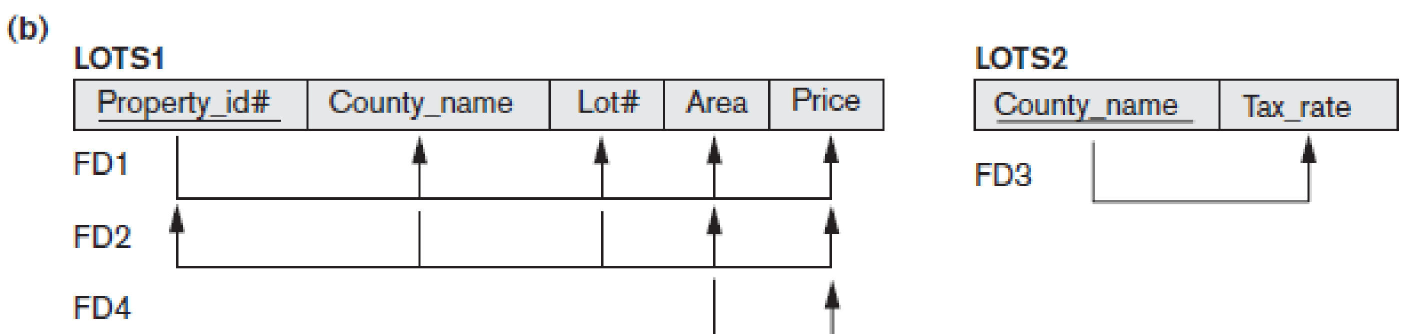
  FD3: County_name → Tax_rate

  FD4: Area → Price
- The dependency FD3 says that the tax rate is fixed for a given county (does not vary lot by lot within the same county)
- FD4 says that the price of a lot is determined by its area regardless of which county it is in.

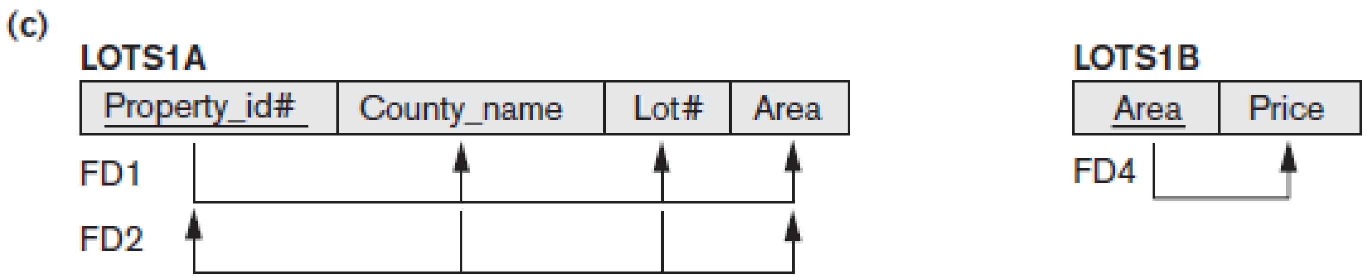
- The LOTS relation schema violates the general definition of 2NF because Tax_rate is partially dependent on the candidate key {County_name, Lot#}, due to FD3.
- To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2.



- We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name into another relation LOTS2.
- Both LOTS1 and LOTS2 are in 2NF.
- FD4 does not violate 2NF and is carried over to LOTS1.
- **Now consider the Definition of 3NF:** A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X→A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.
- According to this definition, LOTS2 is in 3NF.
- FD4 in LOTS1 violates 3NF because Area is not a superkey and Price is not a prime attribute in LOTS1.



- To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B.



- We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area into another relation LOTS1B.
- Both LOTS1A and LOTS1B are in 3NF.
- Thus to convert LOTS table to 3NF, it is divided into three : LOTS1A , LOTS1B and LOT2

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.
- **Defenition:** A table is in BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.
- That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF

Boyce-Codd Normal Form

- Consider a relation TEACH with the following dependencies:
- {student_id, subject} is a candidate key for this relation.
- Consider two functional dependencies of TEACH Relation

FD1: {student_id, subject} → professor

FD2: professor → subject

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

Boyce-Codd Normal Form

- This relation is in 1 NF, 2 NF and 3NF but not BCNF because of FD2.
- Decomposition of this relation schema into two schemas is not straightforward because it may be decomposed into one of the three following possible pairs:
  1. {student_id, professor} and {student_id, subject}.
  2. {subject, professor} and {subject, student_id}.
  3. {student_id, professor, } and{professor, subject}
- The desirable decomposition among those just shown is 3 because it will not generate spurious tuples after a join.

Boyce-Codd Normal Form

- The decomposition 3 for TEACH yields two relations in BCNF as:

  TEACH1(student_id, professor) and TEACH2(professor, subject) where {student_id, professor} will be the key in TEACH1 and professor will be the key in TEACH2

- This is an example of a case where we may reach the same ultimate BCNF design via alternate paths of normalization.

Nonadditive join(Lossless join) and Dependency preservation

- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess.
- These would include two properties:
  - The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
  - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

Nonadditive join(Lossless join)

- **Definition:** A decomposition D = {R1, R2, ..., Rm} of R has the lossless(nonadditive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all the relations in D: $*(\pi R1(r), \ldots, \pi Rm(r)) = r$.
- The word loss in **lossless** refers to **loss of information**, not to loss of tuples. If a decomposition does not have the lossless join property, we may get **additional spurious tuples** after the PROJECT (π) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous or invalid information.
- The Lossless join(nonadditive join) property ensures that no spurious tuples result after the application of PROJECT and JOIN operations.

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$ $R_1 = (A, B)$ $R_2 = (B, C)$
- After PROJECT (π) and NATURAL JOIN (*) operations are applied on R1 and R2, no spurious tuples are there in the result.

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

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 ie. we cannot reconstruct the original employee relation as it generates spurious tuples and so, this is a **lossy decomposition**.

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

employee

employee1

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

employee2

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
⋮				

Sinc

Testing for non additive join property

Algorithm 16.3. Testing for Nonadditive Join Property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R, and a set F of functional dependencies.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (* *comment* *).

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set $S(i, j) := b_{ij}$ for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i, j) *).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
      {if (relation R_i includes attribute A_j) then set $S(i, j) := a_j$;};}; (* each a_j 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 $X \rightarrow Y$ in F

{for all rows in S *that 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 appears 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 nonadditive join property; otherwise, it does not.

Example

Figure 16.1
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 = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2\}$

R_1 = EMP_LOCS = {Ename, Plocation}

R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → 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)

$R = \{\text{Ssn, Ename, Pnumber, Pname, Plocation, Hours}\}$

$D = \{R_1, R_2, R_3\}$

$R_1 = \text{EMP} = \{\text{Ssn, Ename}\}$

$R_2 = \text{PROJ} = \{\text{Pnumber, Pname, Plocation}\}$

$R_3 = \text{WORKS\_ON} = \{\text{Ssn, Pnumber, Hours}\}$

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

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	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)

Dependency Preservation

Defenition: A decomposition $D = \{R1, R2, \ldots, Rm\}$ of R is dependency-preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to F; that is, $((\pi R1(F)) \cup \ldots \cup (\pi Rm(F)))^{+} = F^{+}$.

Algorithm : Relational Synthesis into 3NF with **Dependency Preservation and Nonadditive Join Property**

- Input: A relation R and a set of functional dependencies F on the attributes of R.

Steps

1. Find a minimal cover G for F.
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes {X ∪ {A1} ∪ {A2} … ∪ {Ak} }, where X⟶A1, X⟶A2, …, X⟶Ak are the only dependencies in G with X as left-hand-side (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R.
4. Eliminate redundant relations from the resulting set of relations in the relational database schema.

Example: Consider the following relation: R(Emp_ssn, Pno, Esal, Ephone, Dno, Pname, Plocation) The following dependencies are present:

FD1: Emp_ssn → {Esal, Ephone, Dno}

FD2: Pno → { Pname, Plocation}

FD3: Emp_ssn, Pno → {Esal, Ephone, Dno, Pname, Plocation}

- By virtue of FD3, the attribute set {Emp_ssn, Pno} represents a key of the relation R.
- Let the Minimal cover G: {Emp_ssn → Esal, Ephone, Dno; Pno → Pname, Plocation}
- The second step produces relations R1 and R2 as shown below . However, now in step 3, we will generate a relation corresponding to the key {Emp_ssn, Pno}. Hence, the resulting design contains:

R1 (Emp_ssn , Esal, Ephone, Dno)

R2 (Pno, Pname, Plocation)

R3 (Emp_ssn, Pno)

- This design achieves both the desirable properties of **dependency preservation** and **nonadditive(lossless) join**.