Module 4: Normalization

PREPARED BY SHARIKA T R, SNGCE

PREPARED BY SHARIKA T R, SNGCE

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.

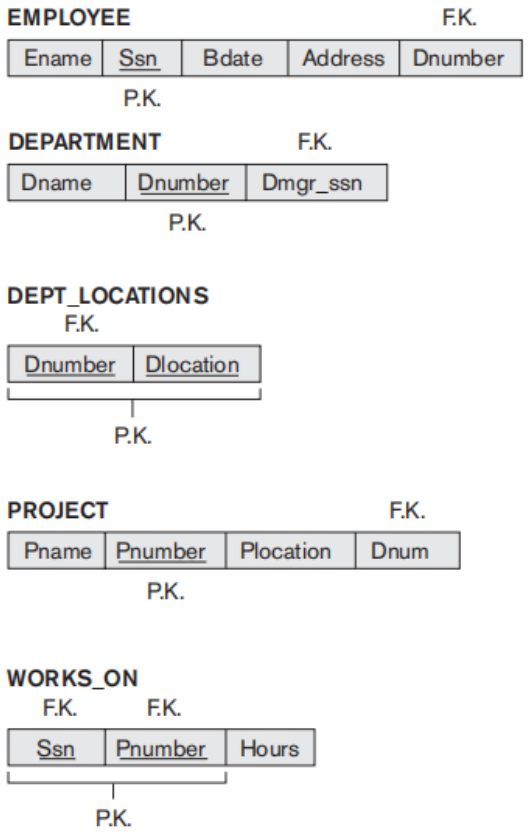
Informal Design Guidelines for Relation Schema

- 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

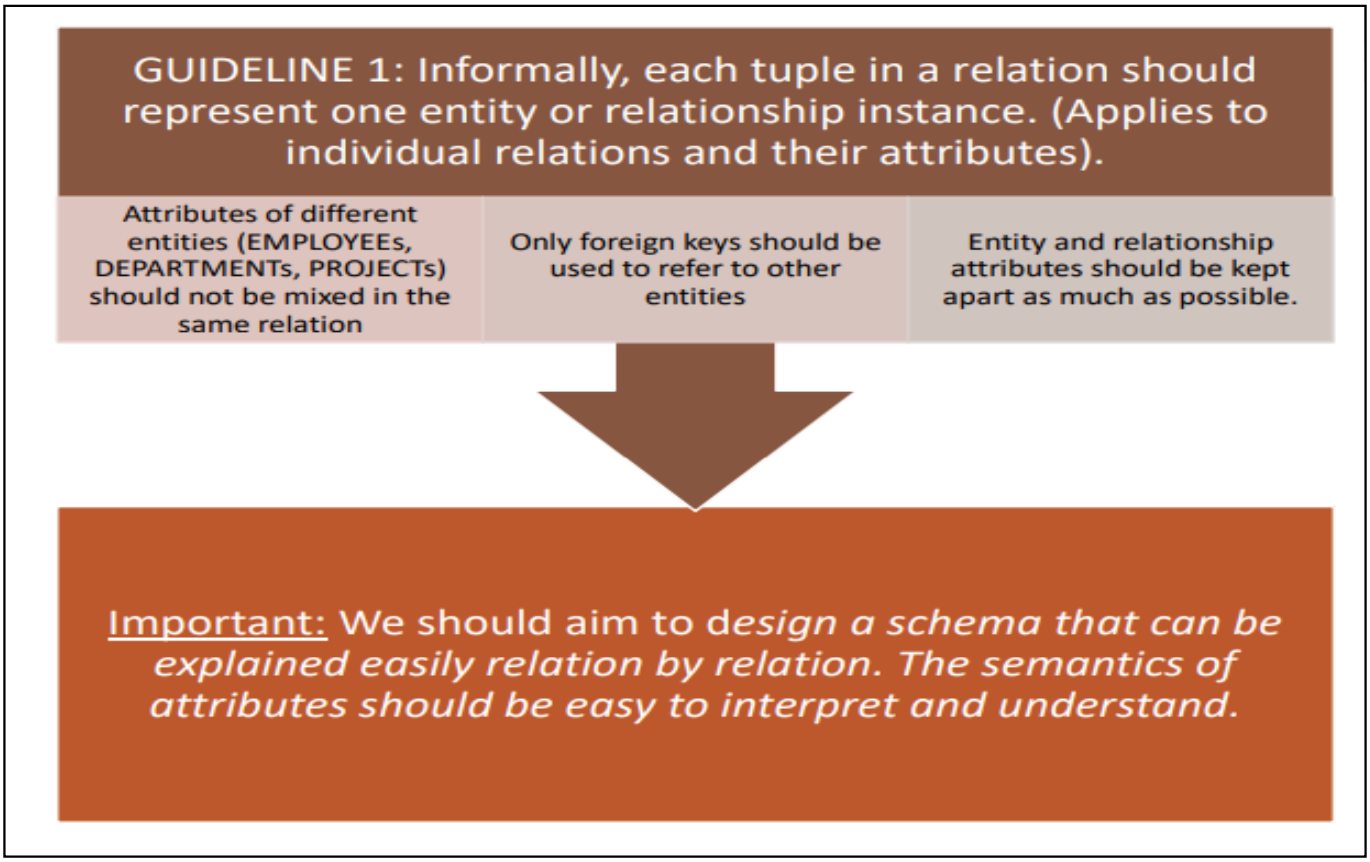
Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

PROJECT

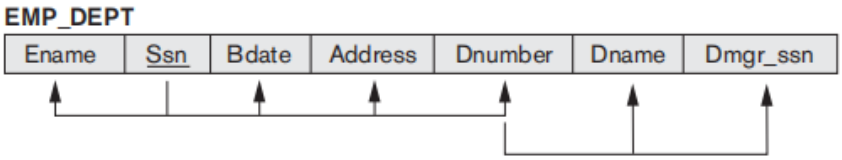
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

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.



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

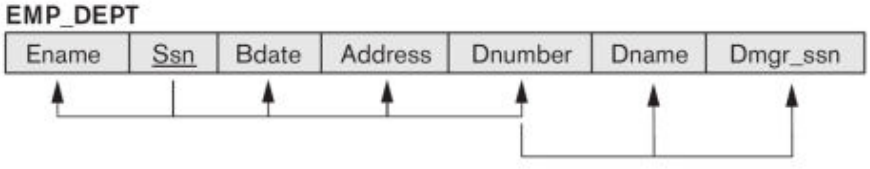
Redundant Information in Tuples and Update Anomalies

- 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, Ssn, Bdate, Address, Dnumber, Dname, Dmgr_ssn)

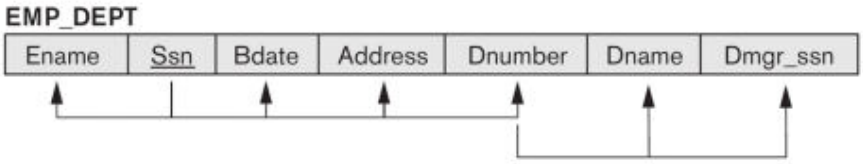


insertion anomalies: when adding an employee, we must assign them to a department or else use NULLs. When adding a new department with no employees, we have to use NULLs for the employee Ssn, which is supposed to be the primary key!

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

EMP_DEPT

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

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

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.

EMP_PROJ1

Ssn	Pnumber	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

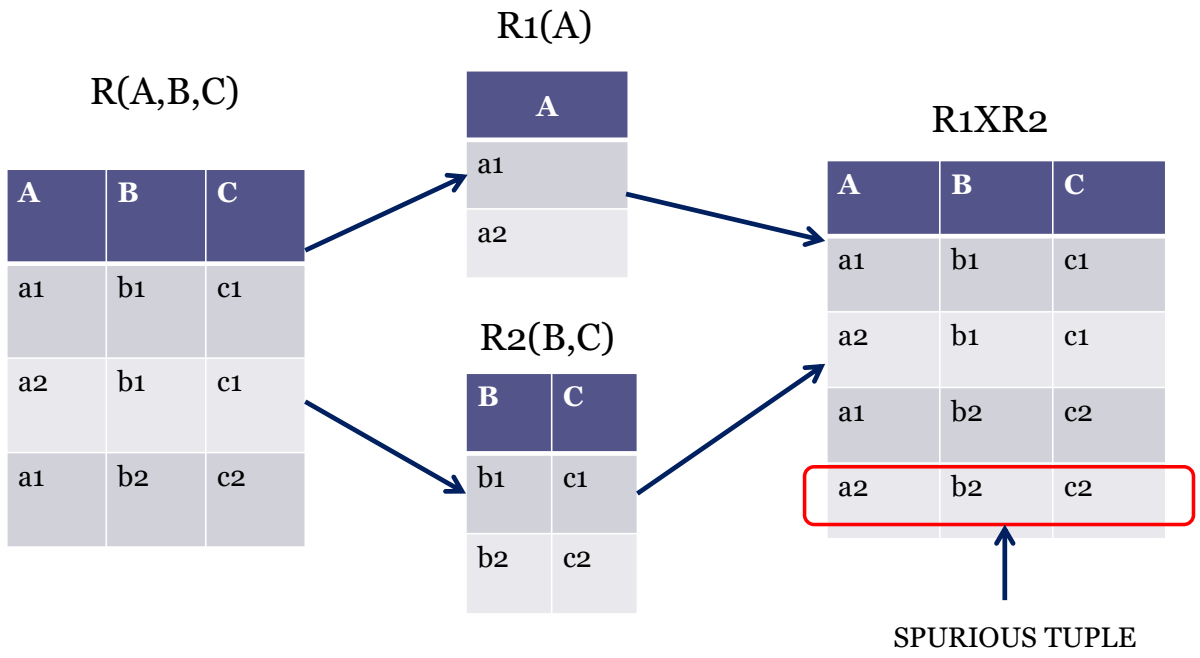
EMP_PROJ1

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.



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

PREPARED BY SHARIKA T R, SNGCE

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 → 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→Y holds if whenever two tuples have the same value for X, they must have the same value for Y
- For any two tuples t1 and t2 in any relation instance r(R):
  - If t1[X]=t2[X], then t1[Y]=t2[Y]
- X→Y in R specifies a constraint on all relation instances r(R)
- Written as X→Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

PREPARED BY SHARIKA T R, SNGCE

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 → B So this is not a valid FD no unique matching
X a1 (b1,b2)
✓ a2 b3

B → A So this is a valid FD
✓ b1 a1
✓ b3 a2
✓ b2 a1

B → 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→BC

Properties of Functional Dependencies

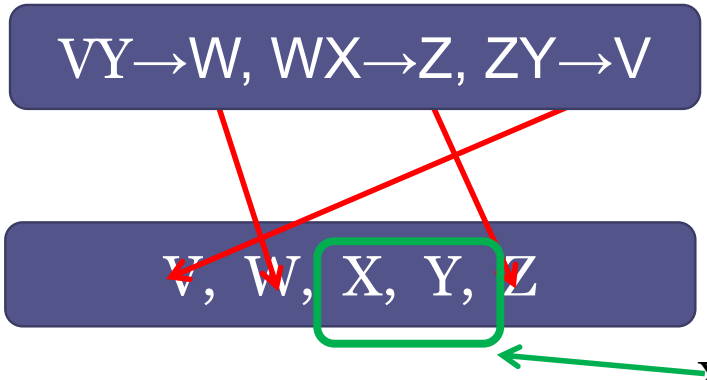
- There are several useful rules that let you replace one set of functional dependencies with an equivalent set.
- Some of those rules are as follows:
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow Y Z$
  - Transitivity: If $X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

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
- 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 must be in CK

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

PREPARED BY SHARIKA T R, SNGCE

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

Secondary Rules

PREPARED BY SHARIKA T R, SNGCE

- 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

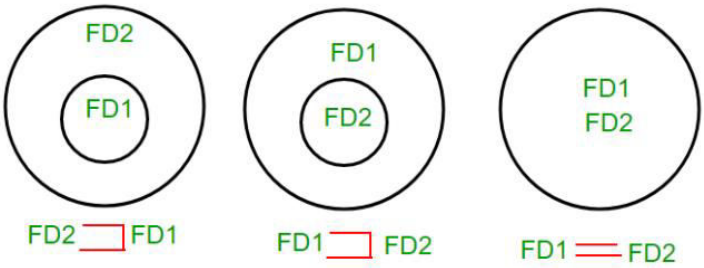
Two sets of functional dependencies E and F are equivalent if **E+ = F+.** Therefore, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions—E covers F and F covers E—hold.

PREPARED BY SHARIKA T R, SNGCE

- 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.

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.

FD1 = {A->B, B->C, AB->D} and
FD2 = {A->B, B->C, A->C, A->D}

- 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 ⊃ 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 FDs F is a minimal set of functional dependencies Fmin that is equivalent to F.
- We can think of a minimal set of dependencies as being a set of dependencies in a standard or canonical form and with no redundancies.
- A canonical cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, it is also called as Irreducible set.
- Canonical cover is free from all the extraneous functional dependencies

- 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).

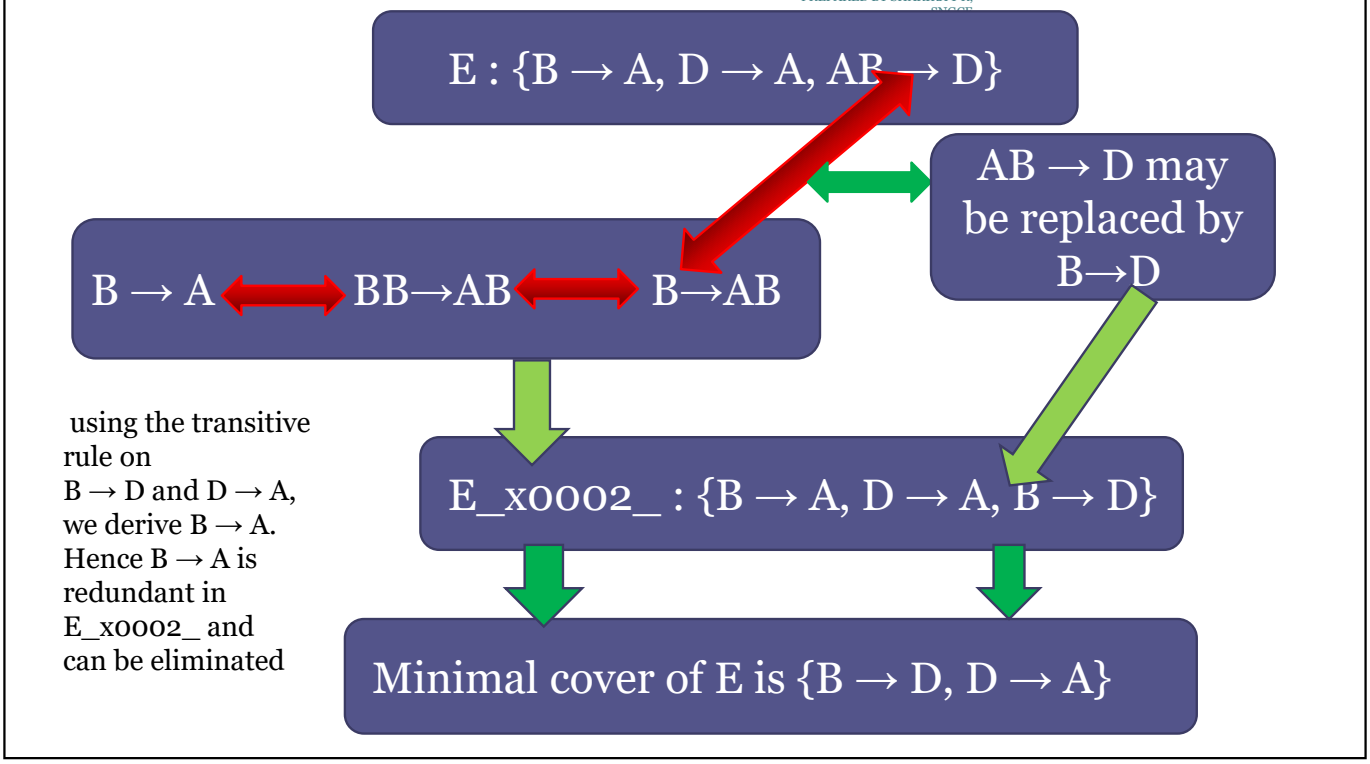
Definition. A minimal cover of a set of functional dependencies E is a minimal set of dependencies (in the standard canonical form and without redundancy) that is equivalent to E. We can always find at least one minimal cover F for any set of dependencies E

- 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}.



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

- 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” or “purification” 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 normalization procedure provides database designers with the following:
  - A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
  - A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree 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

- The process of normalization through decomposition should satisfy the following two properties
- 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.

- Denormalization is the process of storing the join of higher normal form relations as a base relation,

PREPARED BY SHARIKA T R, SNGCE

Definitions of Keys and Attributes Participating in Keys

- A superkey of a relation schema R = {A1, A2, ... , An} is a set of attributes S ⊆ R with the property that no two tuples t 1 and t 2 in any legal relation state r of R will have t 1[S] = t 2[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
- The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key K = {A1, A2, ..., Ak} of R, then K – {Ai } is not a key of R for any Ai , 1 ≤ i ≤ k.

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

- {Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys

- Ssn,pnumber→hour
- {SSn,pnumber} is a key
- But SSn is not a key

(b)

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation

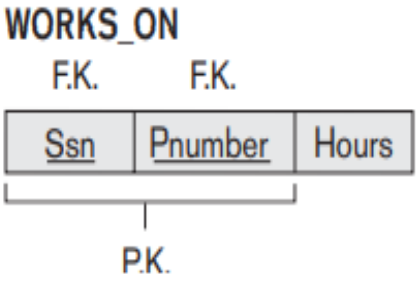
FD1
FD2
FD3

Candidate key

- 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.
- In a practical relational database, each relation schema must have a primary key.
- If no candidate key is known for a relation, the entire relation can be treated as a default superkey.

Prime and Non Prime Attributes

- An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.
- An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.



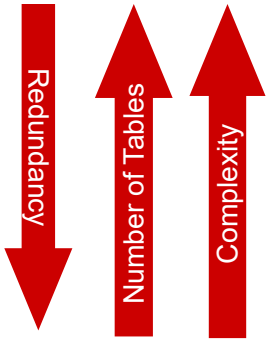
- both Ssn and Pnumber are prime attributes of WORKS_ON, whereas other attributes of WORKS_ON are nonprime

Practical Use of Normal Forms

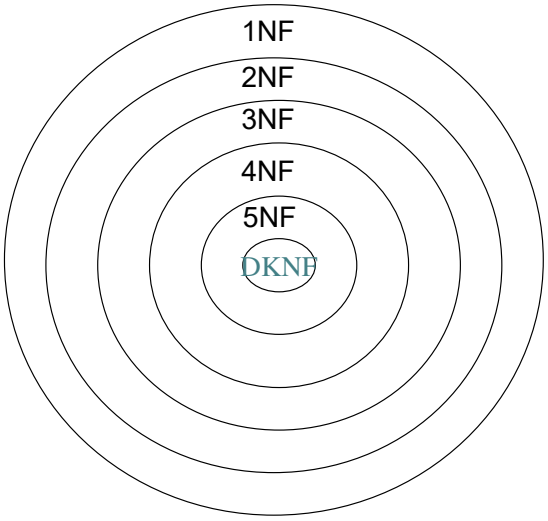
- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect
- The database designers need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

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

Disallow multivalued attributes, composite attributes, and their combinations.

- 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.
- In other words, 1NF disallows relations within relations or relations as attribute values within tuples.
- The only attribute values permitted by 1NF are single atomic (or indivisible) values

- this is not in 1NF because Dlocations is not an atomic attribute

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

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

- There are three main techniques to achieve first normal form for such a relation:
- 1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation},

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

2) Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT.

- In this case, the primary key becomes the combination {Dnumber, Dlocation}.
- This solution has the disadvantage of introducing redundancy in the relation.

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

3) If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3.

This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations

Dname	Dnumber	Dmrg_ssn	Dlocation 1	Dlocation 2	Dlocation 3

- Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values.
- First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it.

- EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})

(a)
EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)
EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Second Normal Form

- 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.

- A functional dependency X → Y is a partial dependency if some attribute A ε X can be removed from X and the dependency still holds; that is,
- for some A ε X, (X – {A}) → Y.

- Give a relation R(A,B,C) with FD ={AB → C, B → C}
- Find CK
  - AB+=ABC it is a CK
  - So prime attribute are AB
  - Non prime attribute is C
  - AB → C This is a Full Functional Dependency
  - B → C Here non prime attributes depend on a part of key
- A non prime attribute partially depends on key then it is called Partial Fuctional Dependency
- If there is Partial dependency then R not in 2NF

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

Definition. A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R

- The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key.

  If the primary key contains a single attribute, the test need not be applied at all.

- SSn→Ename

- If a relation schema is not in 2NF, it can be second normalized or 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.

Qn.Consider Relation Std_Faculty(Sid,Cid, Mark, Faculty).With FD={Sid,Cid → Marks, Cid → Faculty}. Is this in 2NF?

Ans. Find CK,

(Sid,Cid)+=Sid, Cid,Marks, Faculty

So (Sid,Cid) is a candidate key here

Here Prime attribute=Sid,Cid

Non Prime attribute=Marks, Faculty

Now check for Parial functional dependency

Sid,Cid → Marks ✔Full functional dependent

Cid → Faculty ☐ Partial dependency

So not in 2NF

- To eliminate redundancy and convert to 2NF → Decompose to sub relations
- For partial dependency create table

  R1(Sid,Cid, Mark) and R2(Cid, Faculty)

  FD1{Sid,Cid → Marks}✔ FD2{Cid → Faculty}✔
- No partial dependency in R1 and R2 So this is in 2NF

- If a non prime attribute depends on a proper subset of any key of R then there is Partial Dependency

Proper subset: A proper subset of a set A is a subset of A that is not equal to A. In other words, if B is a proper subset of A, then all elements of B are in A but A contains at least one element that is not in B.

{B,C}'s proper subset has {B,C} BC not included.

- In Partial dependency will be of form

Proper subset of any key of R → Non Prime attributes

- If there is Partial dependency then not in 2NF

Third Normal Form

- Third normal form (3NF) is based on the concept of transitive dependency.
- Conditions for 3NF

1. Relation should be in 2NF
2. No transitive dependency is allowed

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.

- If a non prime attribute is transitively depends on Key through another non prime attribute then there is transitive dependency, such a dependency is not allowed in 3NF

Consider Relation Employee_Dept(Eid,Ename,Dno, Dmgrid).With FD={Eid → Ename, Eid → Dno, Dno → Dmgrid}. Is this in 3NF?

Ans. First check for 2NF

No partial dependency so is in 2NF

Find CK, Eid+=Eid,Ename,Dno,Dmgrid ✔CK

Prime Attribute= {Eid}

Non Prime Attributes={Ename,Dno, Dmgrid}

Now check for transitive dependency,

Eid → Dno,

Dno → Dmgrid

- This is transitive dependency. So not in 3NF

- Definition. According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and **no** nonprime attribute of R is transitively dependent on the primary key.

- To make this 3NF we have to decompose relation Employee_Dept into two relations
- Employee(Eid,Ename,Dno) and
- Dept(Dno, Dmgrid)

A relation R is in 3NF if for every FD X → Y
Either X is a SK or
Y is a prime attribute of R

PREPARED BY SHARIKA T R, SNGCE

- FD={Eid → Ename, Eid → Dno, Dno → Dmgrid}
- Here, Eid → Ename ✔//holds since Eid Super Key
- Eid → Dno✔//holds since Eid Super Key
- Dno → Dmgrid ☐ //Dno is not super key nor Dmgrid is prime. so violate 3NF

Boyce-Codd normal form (BCNF)

- Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
- S
- That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in 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 the table.

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

Summary

- 1NF: Ensure Atomicity
- 2NF: Must be in 1NF + Ensure no partial dependency

Proper subset of any key of R → Non Prime attributes
or //BOTH NOT ALLOWED IN 2NF
Prime attribute → Non Prime attributes

- 3NF: Must be in 2NF & No transitive dependency

A relation R is in 3NF if for every FD X → Y
Either X is a SK or
Y is a prime attribute of R

- BCNF: X → Y Where X is a Super Key

- Non Prime → Prime
- Prime → Non Prime
- These are allowed in 2NF and 3NF
- Removed by BCNF

Lossless and Lossy Decomposition in DBMS

- Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.
- There are mainly two types of decompositions in DBMS-

1. Lossless Decomposition
2. Lossy Decomposition

Lossless Decomposition in DBMS

- Lossless join decomposition is a decomposition of a relation R into relations R1,R2 such that if we perform natural join of two smaller relations it will return the original relation.
  - This is effective in removing redundancy from databases while preserving the original data..

In other words by lossless decomposition it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

- In Lossless Decomposition we **select the common element** and the criteria for selecting common element is that the common element must be a **candidate key or super key** in either of relation R1,R2 or both.
- Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+

$$R1 \cap R2 \rightarrow R1$$

OR

$$R1 \cap R2 \rightarrow R2$$

Example 1

- Given R(A,B,C)
- FD={A→B, B→C, C→A}
- is decomposed to R1(A B) and R2(BC) check wheather it is lossless join or not

ANS. There is a common attribute in R1 and R2 ie, B. Now check B is a candidate key for R1 or R2

B^+=BCA

ie B can be a candidate key in both R1 and R2 so is Lossless decomposition

Example 2

- Given R(A,B,C,D)
- FD={AB→CD, D→A}
- is decomposed to R1(A C) and R2(B C D) check wheather it is lossless join or not

ANS. There is a common attribute in R1 and R2 ie, C. Now check C is a candidate key for R1 or R2

$C^+=C$

ie C is not a candidate key in both R1 and R2 so is Lossy decomposition

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 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→Y 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.

Using algorithm find decomposition is Lossless or not

- Given R(A,B,C,D,E)
- Decomosed to R1(A,B,C), R2(B,C,D), R3(C,D,E)
- FD={AB→CD, A→E, C→D}

Let's construct a table of the above relation R, R1 R2 and R3 and insert value in form of bij or aj using ALGO STEP1
(Create an initial matrix S with one row i for each relation in Ri in D, and one column j for each attribute aj in R).

S					
	A	B	C	D	E
R1					
R2					
R2					

Created a table using R = { A B C D E } where every attribute of R is represented in each column. And initial value of each decomposed table R1 R2 and R3 in the format of bij, where i is the row and j is the column using ALGO STEP2 (Set *S(i, j)* := bij for all matrix entries. (* each bij is a distinct symbol associated with

S					
	A	B	C	D	E
R1	b11	b12	b13	b14	b15
R2	b21	b22	b23	b24	b25
R2	b31	b32	b33	b34	b35

- Now insert value in row R1 R2 and R3 as "aj" using R1 = { A,B ,C } R2 = { B,C,D} and R3 = { C,D,E } using ALGO STEP3 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

S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- Given Functional Dependencies are FD={AB→CD, A→E, C→D}
- Using step 4 of above algorithm, if there exist a functional dependency X → Y, and for two tuples t1, and t2 if
- t1 [X] = t2 [X] then we must have
- t1 [Y] = t2 [Y]

- Find in the above table that is there any FD X → Y whose X are equal then make Y also equal.
- **Step A:** By using the above FD AB→CD, rows of A and B column do not have any same value, No action taken

S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- **Step B:** By using FD A→ E on the above table we found that A has no same values so No action taken

S					
	A	B	C	D	E
R1	a1	a2	a3	b14	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- **Step C:** Since by using above FD: C → D IN C all tuple have same value a3 so we will make b values in D to a value

S					
	A	B	C	D	E
R1	a1	a2	a3	~~b14~~ a4	b15
R2	b21	a2	a3	a4	b25
R2	b31	b32	a3	a4	a5

- Now look for any row with all a values. Here there is no such row so we can conclude this is a lossy join

Dependency Preservation

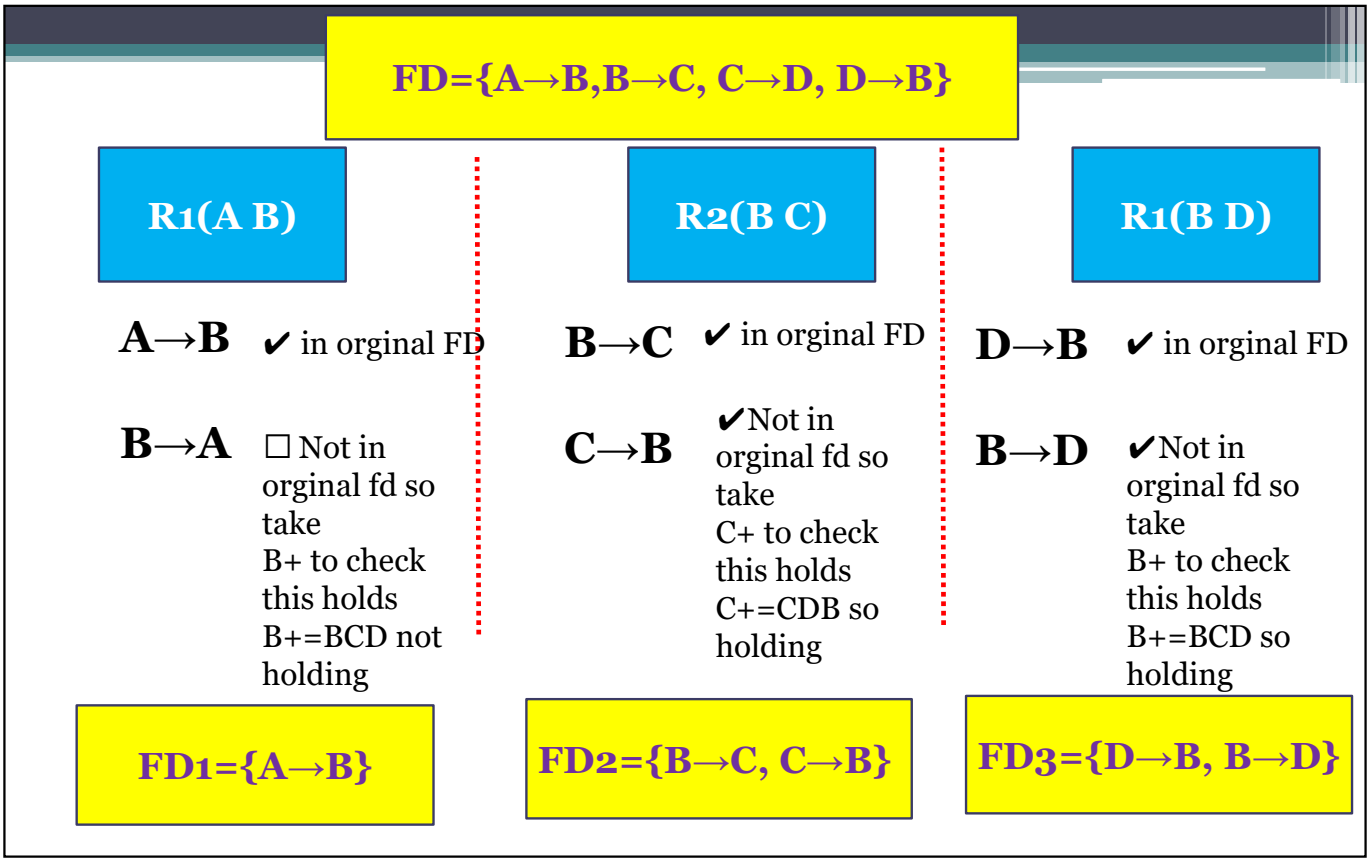
- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.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

Algorithm to check for Dependency Preservation

```
begin;
for each X → Y in F and with R (R1, R2, …, Rn)
{
        let Z = X;
        while there are changes in Z
        {
                from i=1 to n
                        Z = Z ∪ ((Z ∩ Ri)+ ∩ Ri) w.r.t to F;
        }
        if Y is a proper subset of Z, current fd is preserved
        else decomposition is not dependency preserving;
}
this is a dependency preserving decomposition;
end;
```

Example

- R(A,B,C,D)
- FD={A→B,B→C, C→D, D→B}
- Decomposed into R1(AB), R2(B,C) and R3(B,D)



- Now take FD that hold in R1, R2 ,R3 to find FD1∪FD2∪FD3
- Now check these dependences are preserved in FD

FD1∪FD2∪FD3

FD={A→B,B→C, C→D, D→B}

{A→B, → ✔ in orginal FD

B→C, → ✔ in orginal FD

C→B, → ✔ not in orginal FD. Take C+ corresponds to FD1∪FD2∪FD3. C+=CBD. B is covered here

B→D, → ✔ not in orginal FD. Take B+ corresponds to FD1∪FD2∪FD3. B+=BCD. D is covered here

D→B} → ✔ in orginal FD

So all FD is preserved in FD1∪FD2∪FD3

References

- Elmasri R. and S. Navathe, Database Systems: Models, Languages, Design and Application Programming, Pearson Education, 2013.
- https://www.geeksforgeeks.org/armstrongs-axioms-in-functional-dependency-in-dbms/