Module 4

Chapter 1: Database Design Theory

4.0 Introduction

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. This module discuss the basic and higher normal forms.

4.1 Objectives

- To study the process of normalization and refine the database design
- To normalize the tables upto 4NF and 5NF
- To study lossless and lossy join operations
- To study inference rules
- To study other dependencies and Normal Forms.

4.2 Introduction to DB design

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or Enhanced-ER (EER) data model. These models make the designer identify entity types and relationship types and their respective attributes, which leads to a natural and logical grouping of the attributes into relations.

Database Design deals with coming up with a 'good' schema. There are two levels at which we can discuss the goodness of relation schemas:

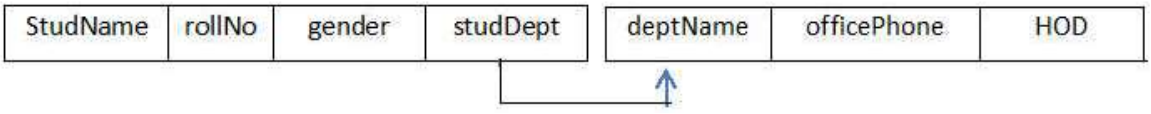
1. The logical (or conceptual) level—how users interpret the relation schemas and the meaning of their attributes.
2. The implementation (or physical storage) level—how the tuples in a base relation are stored and updated. This level applies only to schemas of base relations

An Example

- STUDENT relation with attributes: studName, rollNo, gender, studDept
- DEPARTMENT relation with attributes: deptName, officePhone, hod
- Several students belong to a department
- studDept gives the name of the student's department

Correct schema:

Student Department



Incorrect schema:

Studdept

StudName	rollNo	gender	deptName	officePhone	HOD

Problems with bad schema

- **Redundant storage of data:**
  - Office Phone & HOD info -stored redundantly once with each student that belongs to the department
  - wastage of disk space
- **A program that updates Office Phone of a department**
  - must change it at several places
  - more running time
  - error -prone

4.3 Informal Design Guidelines for Relation Schemas

- 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
- These measures are not always independent of one another

4.3.1 Imparting Clear Semantics to Attributes in Relations

- **semantics** of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple
- 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 easier it is to explain the semantics of the relation, the better the relation schema design will be

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

- if a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning
- 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

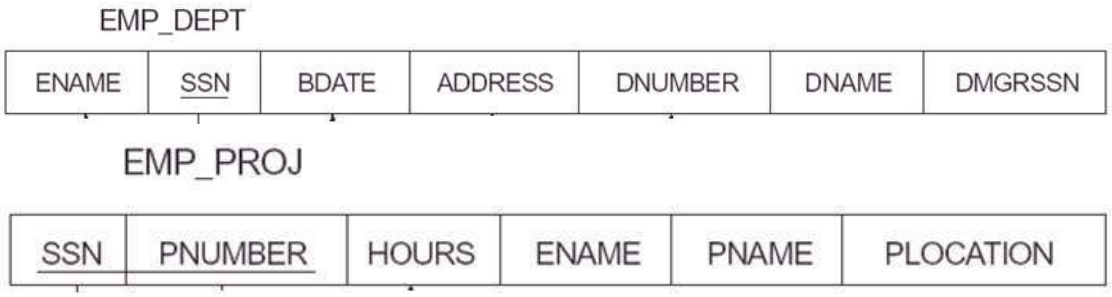


Fig: schema diagram for company database

- Both the relation schemas have clear semantics
- A tuple in the EMP_DEPT relation schema represents a single employee but includes additional information— the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager.
- A tuple in the EMP_PROJ relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location (Plocation)
- logically correct but they violate Guideline 1 by mixing attributes from distinct real-world entities:
  - EMP_DEPT mixes attributes of employees and departments
  - EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship
- They may be used as views, but they cause problems when used as base relations

4.3.2 Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations
- Grouping attributes into relation schemas has a significant effect on storage space
- For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT with that for an EMP_DEPT base relation
- In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department

- In contrast, each department's information appears only once in the DEPARTMENT relation. Only the department number Dnumber is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	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

Figure 1: One possible database state for the COMPANY relational database schema

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

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

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

WORKS_ON

Essn	Pno	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

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure 1 : One possible database state for the COMPANY relational database schema

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

Fig: Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 1

- Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into:
  - insertion anomalies
  - deletion anomalies,
  - modification anomalies

Insertion Anomalies

- Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

1. To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs
   - For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT
   - In the design of Employee in fig 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation
2. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee
   - This violates the entity integrity for EMP_DEPT because Ssn is its primary key
   - This problem does not occur in the design of Figure 1 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

Deletion Anomalies

- The problem of deletion anomalies is related to the second insertion anomaly situation just discussed
  - 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
  - This problem does not occur in the database of Figure 2 because DEPARTMENT tuples are stored separately.

Modification Anomalies

- In 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

Guideline 2

- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly
- The second guideline is consistent with and, in a way, a restatement of the first guideline
- These guidelines may sometimes have to be violated in order to improve the performance of certain queries.

4.3.3 NULL Values in Tuples

- If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples
  - this can waste space at the storage level
  - may lead to problems with understanding the meaning of the attributes
  - may also lead to problems with specifying JOIN operations
  - how to account for them when aggregate operations such as COUNT or SUM are applied
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.
- Moreover, NULLs can have multiple interpretations, such as the following:
  - The attribute *does not apply* to this tuple. For example, Visa_status may not apply to U.S. students.
  - The attribute value for this tuple is *unknown*. For example, the Date_of_birth may be unknown for an employee.
  - The value is *known but absent*; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

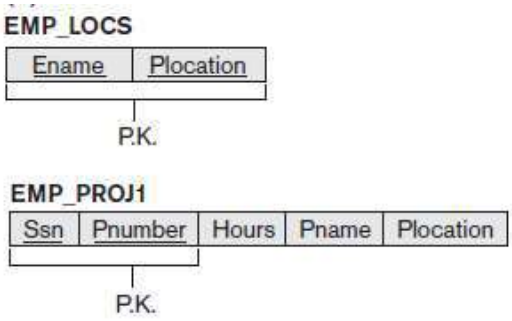
Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation
- Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns with the appropriate key columns

- 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.

4.3.4 Generation of Spurious Tuples

- Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ



- A tuple in EMP_LOCS means that the employee whose name is Ename works on *some project* whose location is Plocation
- A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation.

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** because they represent spurious information that is not valid.
- The spurious tuples are marked by asterisks (*)

	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.

4.4 Functional Dependencies

- Formal tool for analysis of relational schemas that enables us to detect and describe some of the problems in precise terms

Definition of Functional Dependency

- A functional dependency is a constraint between two sets of attributes from the database.
- Given a relation R, a set of attributes X in R is said to **functionally determine** another attribute Y, also in R, (written X → Y) if and only if each X value is associated with at most one Y value.
- X is the determinant set and Y is the dependent attribute. Thus, given a tuple and the values of the attributes in X, one can determine the corresponding value of the Y attribute.
- The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.
- A functional dependency is a property of the semantics or meaning of the attributes.
- The database designers will use their understanding of the semantics of the attributes of R to specify the functional dependencies that should hold on all relation states (extensions) r of R.

- Consider the relation schema EMP_PROJ;

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

- From the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn→Ename

b. Pnumber →{Pname, Plocation}

c. {Ssn, Pnumber}→Hours

- These functional dependencies specify that

  (a) the value of an employee's Social Security number (Ssn) uniquely determines the employee name (Ename)

  (b) the value of a project's number (Pnumber) uniquely determines the project name (Pname) and location (Plocation), and

  (c) a combination of Ssn and Pnumber values uniquely determines the number of hours the employee currently works on the project per week (Hours).

- Alternatively,we say that Ename is functionally determined by (or functionally dependent on) Ssn, or *given a value of Ssn, we know the value of Ename,* and so on.
- Relation extensions r(R) that satisfy the functional dependency constraints are called **legal relation states** (or **legal extensions**) of R
- 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

Diagrammatic notation for displaying FDs

- Each FD is displayed as a horizontal line
- The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD
- The right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.

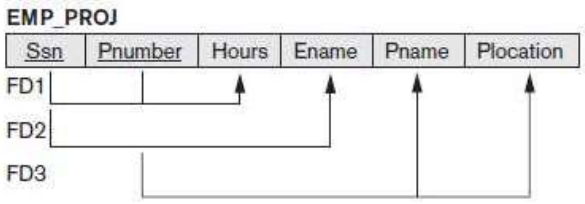


Fig: diagrammatic notation for displaying FDs

Example:

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

- The following FDs *may hold* because the four tuples in the current extension have no violation of these constraints:
  - $B \rightarrow C$
  - $C \rightarrow B$
  - $\{A, B\} \rightarrow C$
  - $\{A, B\} \rightarrow D$
  - $\{C, D\} \rightarrow B$.
- The following *do not* hold because we already have violations of them in the given extension:
  - $A \rightarrow B$ (tuples 1 and 2 violate this constraint)
  - $B \rightarrow A$ (tuples 2 and 3 violate this constraint)
  - $D \rightarrow C$ (tuples 3 and 4 violate it)

Normal Forms Based on Primary Keys

We assume that a

- Set of functional dependencies is given for each relation
- Each relation has a designated primary key
- This information combined with the tests (conditions) for normal forms drives the normalization process for relational schema design
- First three normal forms for relation takes into account all candidate keys of a relation rather than the primary key

4.4.1 Normalization of Relations

- The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**.
- Initially, Codd proposed three normal forms, which he called first, second, and third normal form
- All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation
- A fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively
- **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.
- Thus, 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

4.4.2 Practical Use of Normal Forms

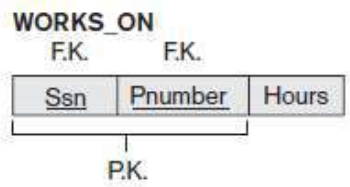
- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.
- The database designers *need not* normalize to the highest possible normal form
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons
- **Definition: Denormalization** is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

4.4.3 Definitions of Keys and Attributes Participating in Keys

- **Superkey:** specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value
- **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
- Example:
    - The attribute set {Ssn} is a key because no two employees tuples can have the same value for Ssn

•Any set of attributes that includes Ssn—for example, {Ssn, Name, Address}—is a superkey

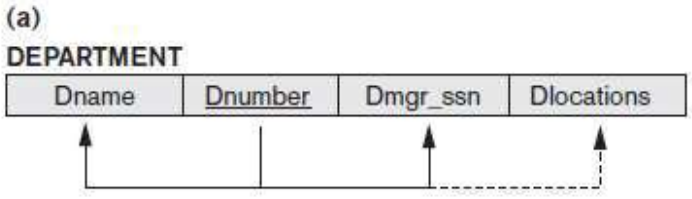
- If a relation schema has more than one key, each is called a **candidate key**
- One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**
- In a practical relational database, each relation schema must have a primary key
- If no candidate key is known for a relation, the entire relation can be treated as a default superkey
- For example {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key
- **Definition.** 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



- In WORKS_ON relation Both Ssn and Pnumber are prime attributes whereas other attributes are nonprime.

4.4.4 First Normal Form

- Defined to 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
- 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**.
- Consider the DEPARTMENT relation schema shown in Figure below



- Primary key is Dnumber
- We assume that each department can have a number of locations

- The DEPARTMENT schema and a sample relation state are shown in Figure below

DEPARTMENT

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

- As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure
- There are two ways we can look at the Dlocations attribute:
  - The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber
  - The domain of Dlocations contains sets of values and hence is nonatomic. In this case, Dnumber→Dlocations because each set is considered a single member of the attribute domain
- In either case, the DEPARTMENT relation is not in 1NF

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}. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
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. Querying on this attribute becomes more difficult; forexample, consider how you would write the query: List the departments that have 'Bellaire' as one of their locations in this design.

- Of the three solutions, 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.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

- Figure above shows how the EMP_PROJ relation could appear if nesting is allowed
- Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours) *within each tuple* represents the employee's projects and the hours per week that employee works on each project.
- The schema of this EMP_PROJ relation can be represented as follows:

  EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})
- Ssn is the primary key of the EMP_PROJ relation and Pnumber is the **partial** key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber
- To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combine the partial key with the primary key of the original relation
- Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2,

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

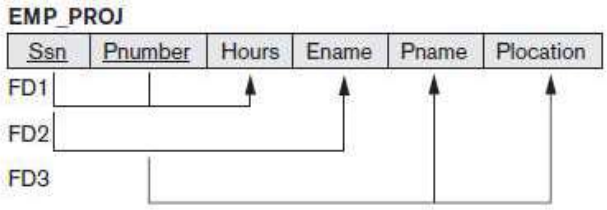
Ssn	Pnumber	Hours

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

4.4.5 Second Normal Form

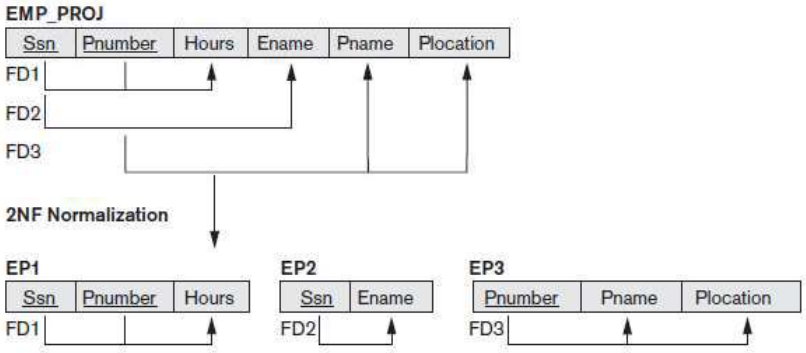
- **Second normal form (2NF)** is based on the concept of full functional dependency
- A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ε X, (X – {A}) does not functionally 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



- In the above figure , {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds)
- {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds
- **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

- 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

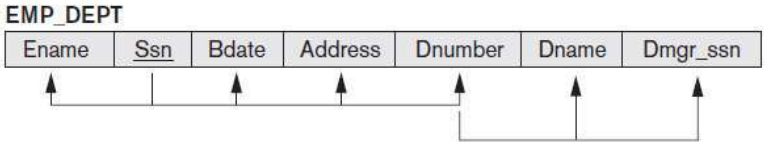
- The EMP_PROJ relation is in 1NF but is not in 2NF.
- The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3
- The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key {Ssn, Pnumber} of EMP_PROJ, thus violating the 2NF test.
- 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.**
- Therefore, the functional dependencies FD1, FD2, and FD3 lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure below, each of which is in 2NF.



4.4.6 Third Normal Form

- **Transitive functional dependency**

  A functional dependency $X \rightarrow Y$ in a relation schema R is a **transitive dependency** if there exists a set of attribute Z that are neither a primary nor a subset of any key of R(candidate key) and both X → Z and Y → Z holds
- **Example:**

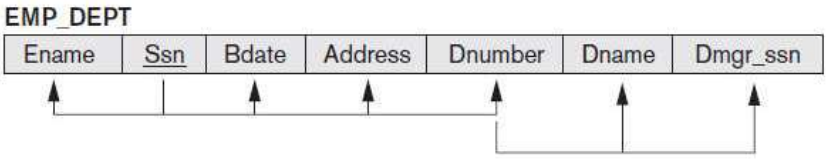


  - **SSN → DMGRSSN is a transitive FD** since SSN → DNUMBER and DNUMBER → DMGRSSN hold

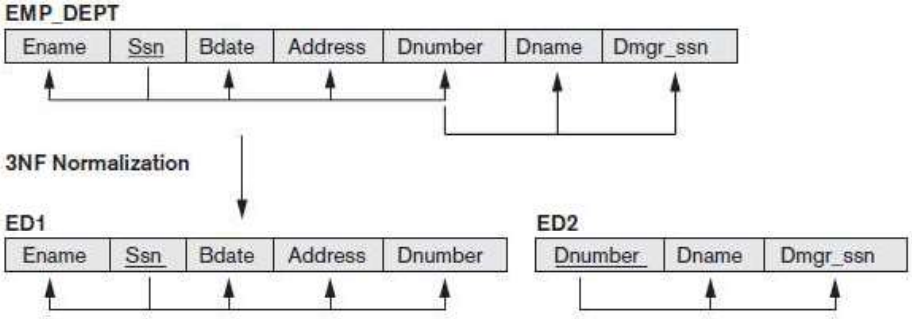
Dnumber is neither a key itself nor a subset of the key of EMP_DEPT

- **SSN → ENAME is non-transitive** since there is no set of attributes X where SSN → X and X → ENAME

- **Definition: A relation schema R is in third normal form (3NF) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key**
- The relation schema EMP_DEPT is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber



- We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2



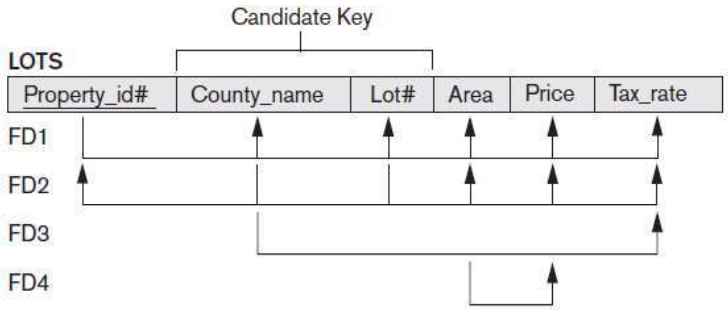
- ED1 and ED2 represent independent entity facts about employees and departments
- A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples
- Problematic FD
  - Left-hand side is part of primary key
  - Left-hand side is a non-key attribute
- 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations
- In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies because these types of dependencies cause the update anomalies

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

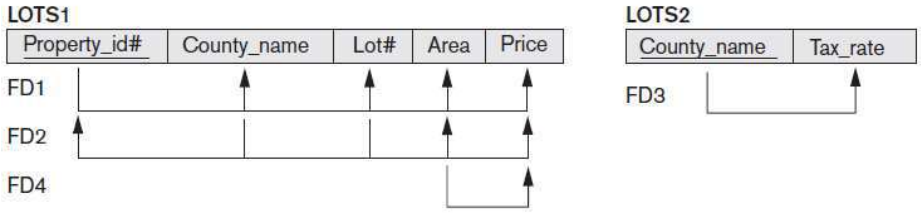
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally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

4.5 General Definition of Second and Third Normal Form

- Takes into account all candidate keys of a relation into account
- **Definition of 2NF:** 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
- 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.

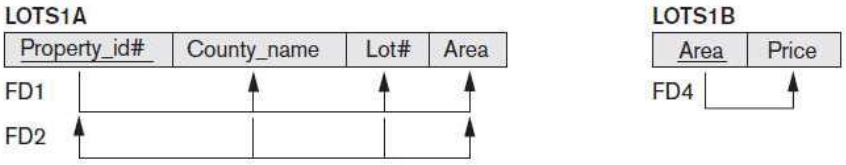


- Based on the two candidate keys Property_id# and {County_name, Lot#}, the functional dependencies FD1 and FD2 hold
  - FD1: Property_id → { County_name,Lot#,Area,Price,Tax_rate}
  - FD2:{County_name,Lot#} →{Property_id, Area,Price,Tax_rate}
  - FD3: County_name → Tax_rate
  - FD4: Area → Price
- We choose Property_id# as the primary key, but no special consideration will be given to this key over the other candidate key
- 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



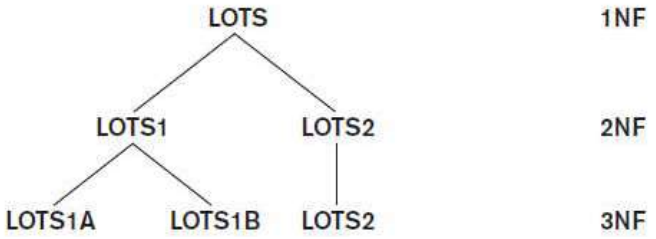
- We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2.
- Both LOTS1 and LOTS2 are in 2NF.

- **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



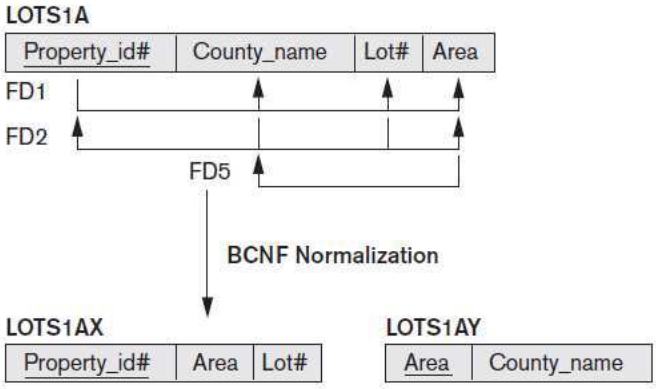
- We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area (the lefthand side of FD4 that causes the transitive dependency) into another relation LOTS1B.

- Both LOTS1A and LOTS1B are in 3NF



4.6 Boyce-Codd Normal Form

- **Boyce-Codd normal form (BCNF)** was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF
- Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF
- **Definition.** A relation schema R is in **BCNF** if whenever a nontrivial functional dependency X→A holds in R, then X is a superkey of R
- The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows *A* to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF
- In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A
- FD5 satisfies 3NF in LOTS1A because County_name is a prime attribute (condition b), but this condition does not exist in the definition of BCNF
- We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.



- In practice, most relation schemas that are in 3NF are also in BCNF
- Only if *X*→*A* holds in a relation schema *R* with *X* not being a superkey *and A* being a prime attribute will *R* be in 3NF but not in BCNF
- Example: consider the relation TEACH with the following dependencies:

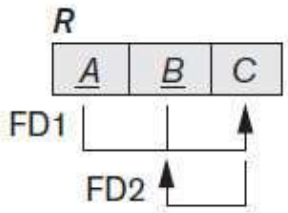
TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

FD1: {Student, Course} → Instructor

FD2: Instructor → Course -- means that each instructor teaches one course

- {Student, Course} is a candidate key for this relation
- The dependencies shown follow the pattern in Figure below with Student as *A*, Course as *B*, and Instructor as *C*



- Hence this relation is in 3NF but not BCNF
- 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. R1(Student, Instructor) and R2(Student, Course)

  2. R1(Course, Instructor) and R2(Course, Student)

  3. R1(Instructor, Course)and R2(Instructor, Student)
- It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF
- Rather, 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.

- We are not able to meet the functional dependency preservation ,but we must meet the non additive join property
- **Nonadditive Join Test for Binary Decomposition:**

  A decomposition D={R_1, R_2} of R has the lossless join property with respect to a set of functional dependencies F on R if and only if either

  - The FD (($R_1 \cap R_2$) →(R_1-R_2)is in F^+ or
  - The FD (($R_1 \cap R_2$) →(R_2-R_1)is in F^+
- The third decomposition meets the test

  $R_1 \cap R_2$ is Instructor

  R_1-R_2 is Course
- Hence, the proper decomposition of TEACH into BCNF relations is:

  TEACH1(Instructor,Course) and TEACH2(Instructor,Student)
- In general, a relation R not in BCNF can be decomposed so as to meet the nonadditive join prorperty by the following procedure. It decomposes R successively into set of relations that are in BCNF:

  Let R be the relation not in BCNF, let X R, and let X → A be the FD that causes violation of BCNF. R may be decomposed into two relations:

  R – A

  XA

  If either R-A or XA is not in BCNF, repeat the process

4.7 Multivalued Dependency and Fourth Normal Form

- For example, consider the relation EMP shown in Figure below:

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

- A tuple in this EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname
- An employee may work on several projects and may have several dependents
- The employee's projects and dependents are independent of one another

- To keep the relation state consistent, and to avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project
- In the relation state shown in the EMP, the employee Smith works on two projects 'X' and 'Y' and has two dependents 'John' and 'Anna' and therefore there are 4 tuples to represent these facts together
- The relation EMP is an **all-key relation** (with key made up of all attributes) and therefore no f.d's and as such qualifies to be a BCNF relation
- There is a redundancy in the relation EMP-the dependent information is repeated for every project and project information is repeated for every dependent
- To address this situation, the concept of multivalued dependency(MVD) was proposed and based on this dependency, the fourth normal form was defined
- **Multivalued dependencies** are a consequence of 1NF which disallows an attribute in a tuple to have a set of values, and the accompanying process of converting an unnormalized relation into 1NF
- Informally, whenever two independent 1:N relationships are mixed in the same relation, R(A, B, C), an MVD may arise

4.7.1 Formal Definition of Multivalued Dependency

Definition. A multivalued dependency X→→Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties where we use Z to denote (R – (X ∪ Y))

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Let X= Ename, Y=Pname
t1[Ename]=t2[ename]=Smith
Z= (EMP-(Ename ∪ Pname))
= Dname

- t3(Ename)=t4(Ename)=t1(Ename)=t2(Ename)=Smith

- t3(Pname)=t1(Pname)=X and t4(Pname)=t2(Pname)=Y
- t3(Dname)=t2(Dname)=Anna and t4(Dname)=t1(Dname)=John
- Whenever $X \rightarrow\rightarrow Y$ holds, we say that X **multidetermines** Y. Because of the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in R, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$, and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.
- An MVD X →→ Y in R is called a **trivial MVD** if
  (a) Y is a subset of X, or
  (b) X ∪ Y = R

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

- For example, the relation EMP_PROJECTS has the trivial MVD
  Ename →→ Pname
- An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**
- If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples
- In the EMP relation the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname)
- This redundancy is clearly undesirable.
- We now present the definition of **fourth normal form (4NF)**, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations
- **Definition: A relation schema *R* is in 4NF with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \rightarrow\rightarrow Y$ in F^+ *X* is a superkey for *R***
- The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

- We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS
- Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs Ename →→ Pname in EMP_PROJECTS and Ename →→ Dname in EMP_DEPENDENTS are trivial MVDs
- No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDENTS. No FDs hold in these relation schemas either

- We can state the following points:
  - An all-key relation is always in BCNF since it has no FDs
  - An all-key relation such as the EMP, which has no FDs but has the MVD Ename→→ Pname | Dname, is not in 4NF
  - A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF
  - The decomposition removes the redundancy caused by the MVD

4.8 Join Dependencies and Fifth Normal Form

- A **join dependency** (**JD**), denoted by JD(R1, R2, ..., Rn), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a nonadditive join decomposition into R1, R2, ..., Rn. Hence, for every such r we have

$$*(\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$$

- A join dependency JD(R1, R2, ..., Rn), specified on relation schema R, is a **trivial** JD if one of the relation schemas Ri in JD(R1, R2, ..., Rn) is equal to R.

Fifth normal form (project-join normal form)

- A relation schema *R* is in **fifth normal form (5NF)** (or **project-join normal form (PJNF)**) with respect to a set *F* of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(*R*1, *R*2, ..., *Rn*) in F^+ every R_i is a superkey of *R*.
- A database is said to be in 5NF, if and only if,
  - It's in 4NF

- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Fig: The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(*R*1, *R*2, *R*3)

R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Fig: Decomposing the relation SUPPLY into the 5NF relations *R*1, *R*2, *R*3.

Chapter 2: Normalization Algorithms

4.9 Inference Rules for Functional Dependencies

- Let F be the set of functional dependencies that are specified on relation schema R
- The schema designer specifies the functional dependencies that are semantically obvious
- Numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F
- Those other dependencies can be inferred or deduced from the FDs in F.
- **For example:**
    - If each department has one manager, so that Dept_no uniquely determines Mgr_ssn (Dept_no → Mgr_ssn), and a manager has a unique phone number called Mgr_phone (Mgr_ssn→Mgr_phone),
    - Then these two dependencies together imply that
      Dept_no → Mgr_phone
    - This is an **inferred FD** and need *not* be explicitly stated in addition to the two given FDs.

- **Definition.** Formally, the set of all dependencies that include *F* as well as all dependencies that can be inferred from *F* is called the **closure** of *F*; it is denoted by F^+.
- For example, suppose that we specify the following set *F* of obvious functional dependencies on the relation schema EMP_DEPT

EMP_DEPT

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN

- F = {
  Ssn → {Ename, Bdate, Address, Dnumber},
  Dnumber → {Dname, Dmgr_ssn}
  }
- Some of the additional functional dependencies that we can *infer* from *F* are the following:
    - Ssn → {Dname, Dmgr_ssn}
    - Ssn → Ssn

- Dnumber → Dname

- An FD X → Y is **inferred from** a set of dependencies F specified on R if X → Y holds in every legal relation state r of R
- The closure F^+ of F is the set of all functional dependencies that can be inferred from F
- Set of **inference rules** can be used to infer new dependencies from a given set of dependencies
- We use the notation $F \models X \rightarrow Y$ to denote that the functional dependency $X \rightarrow Y$ is inferred from the set of functional dependencies F
- we use an abbreviated notation when discussing functional dependencies. We concatenate attribute variables and drop the commas for convenience
- The FD $\{X,Y\} \rightarrow Z$ is abbreviated to $XY \rightarrow Z$, and the FD $\{X, Y,Z\} \rightarrow \{U, V\}$ is abbreviated to $XYZ \rightarrow UV$.
- Three rules IR1 through IR3 are well-known inference rules for functional dependencies.
- They are proposed by Armstrong and hence known as **Armstrong's axioms**
  - IR1 (reflexive rule): If $X \supseteq Y$, then $X \rightarrow Y$.
  - IR2 (augmentation rule): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.
  - IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
- The reflexive rule (IR1) states that a set of attributes always determines itself or any of its subsets, which is obvious.
- Because IR1 generates dependencies that are always true, such dependencies are called *trivial*.
- Formally, a functional dependency $X \rightarrow Y$ is **trivial** if $X \supseteq Y$; otherwise, it is **nontrivial**.
- The augmentation rule (IR2) says that adding the same set of attributes to both the left- and right-hand sides of a dependency results in another valid dependency
- According to IR3, functional dependencies are transitive
- There are three other inference rules that follow from IR1,IR2 and IR3. They are:
  - IR4 (decomposition, or projective, rule): $\{X \rightarrow YZ\} \models X \rightarrow Y$
  - IR5 (union, or additive, rule): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$
  - IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$
- The decomposition rule (IR4) says that we can remove attributes from the right-hand side of a dependency; applying this rule repeatedly can decompose the FD $X \rightarrow \{A1, A2, ..., An\}$ into the set of dependencies $\{X \rightarrow A1, X \rightarrow A2, ..., X \rightarrow An\}$.
- The union rule (IR5) allows us to do the opposite; we can combine a set of dependencies $\{X \rightarrow A1, X \rightarrow A2, ..., X \rightarrow An\}$ into the single FD $X \rightarrow \{A1, A2, ..., An\}$.
- The pseudotransitive rule (IR6) allows us to replace a set of attributes Y on the left hand side of a dependency with another set X that functionally determines Y, and can be

derived from IR2 and IR3 if we augment the first functional dependency $X \rightarrow Y$ with W (the augmentation rule) and then apply the transitive rule.

- In other words, the set of dependencies F+, which we called the **closure** of F, can be determined from F by using only inference rules IR1 through IR3.
- A systematic way to determine these additional functional dependencies is first to determine each set of attributes X that appears as a left-hand side of some functional dependency in F and then to determine the set of *all attributes* that are dependent on X.
- **Definition.** For each such set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F; X^+ is called the **closure of X under F**.
- Algorithm 16.1 can be used to calculate X^+.

Algorithm 16.1. Determining X^+, the Closure of X under F

Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

```
X+ := X;
repeat
    oldX+ := X+;
    for each functional dependency Y → Z in F do
        if X+ ⊇ Y then X+ := X+ ∪ Z;
until (X+ = oldX+);
```

- Algorithm 16.1 starts by setting X^+ to all the attributes in X.
- By IR1, we know that all these attributes are functionally dependent on X.
- Using inference rules IR3 and IR4, we add attributes to X^+, using each functional dependency in F.
- We keep going through all the dependencies in F (the repeat loop) until no more attributes are added to X^+ during a complete cycle (of the for loop) through the dependencies in F.
- For example, consider the relation schema EMP_PRO. From the semantics of the attributes, we specify the following set F of functional dependencies that should hold on EMP_PROJ:

  F = { Ssn → Ename,
  Pnumber → {Pname, Plocation},
  {Ssn, Pnumber} → Hours}
- Using Algorithm 16.1, we calculate the following closure sets with respect to F:
  - $\{Ssn\}^+ = \{Ssn, Ename\}$

- {Pnumber} $^{+}$ = {Pnumber, Pname, Plocation}
- {Ssn, Pnumber} $^{+}$ = {Ssn, Pnumber, Ename, Pname, Plocation, Hours}

4.10Equivalence 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.

4.11Sets of Functional Dependencies

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*.

5 We cannot remove any dependency from *F* and still have a set of dependencies that is equivalent to *F*.

Algorithm 16.2. Finding a Minimal Cover *F* for a Set of Functional Dependencies *E*

Input: A set of functional dependencies E.

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in *F* by the *n* functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
3. For each functional dependency $X \rightarrow A$ in *F*
   for each attribute *B* that is an element of *X*
   if $\{ \{F - \{X \rightarrow A\}\} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to *F*
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in *F*.
4. For each remaining functional dependency $X \rightarrow A$ in *F*
   if $\{F - \{X \rightarrow A\}\}$ is equivalent to *F*,
   then remove $X \rightarrow A$ from *F*.

- Step 2 places FDs in a canonical form for subsequent testing
- Step 3 constitutes removal of an extraneous attribute B contained in the left-hand side X of a functional dependency X->A from F when possible
- Step 4 constitutes removal of a redundant functional dependency x->A from F when possible
- Example 1: Let the given set of FDs be E : {$B \rightarrow A$, $D \rightarrow A$, $AB \rightarrow D$}.We have to find the minimal cover of E.
  - All above dependencies are in canonical form (that is, they have only one attribute on the right-hand side), so we have completed step 1 of Algorithm and can proceed to step 2
  - In step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side; that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$?
  - Since B →A, by augmenting with B on both sides (IR2), we have $BB \rightarrow AB$, or $B \rightarrow AB$ (i). However, $AB \rightarrow D$ as given (ii).
  - Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$. Thus $AB \rightarrow D$ may be replaced by $B \rightarrow D$.
  - We now have a set equivalent to original E, say E: {$B \rightarrow A$, $D \rightarrow A$, $B \rightarrow D$}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
  - In step 3 we look for a redundant FD in E. By using the transitive rule on $B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$. Hence $B \rightarrow A$ is redundant in E and can be eliminated.

Algorithm 16.2(a). Finding a Key K for R Given a set F of Functional Dependencies

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

1. Set $K := R$.
2. For each attribute A in K
   {compute $(K - A)^+$ with respect to F;
   if $(K - A)^+$ contains all the attributes in R, then set $K := K - \{A\}$ };

  - Therefore, the minimal cover of E is {$B \rightarrow D$, $D \rightarrow A$}.

We start by setting K to all the attributes of R; we then remove one attribute at a time and check whether the remaining attributes still form a superkey.

- Algorithm 16.2(a) determines only *one key* out of the possible candidate keys for R; the key returned depends on the order in which attributes are removed from R in step 2.

4.12 Properties of Relational Decompositions

Universal relation schema

- **Universal relation schema** $R = \{A1, A2, ..., An\}$ includes *all* the attributes of the database
- **universal relation assumption**: every attribute name is unique
- The set F of functional dependencies that should hold on the attributes of R is specified by the database designers
- Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas
  $D = \{R1, R2, ..., Rm\}$ that will become the relational database schema ; D is called a **decomposition** of R.

Attribute Preservation condition of a Decomposition

- Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are *lost*; formally, we have

$$\bigcup_{i=1}^{m} R_i = R$$

- Another goal of decomposition is to have each individual relation R_i in the decomposition D be in BCNF or 3NF
- Additional properties of decomposition are needed to prevent from generating spurious tuples

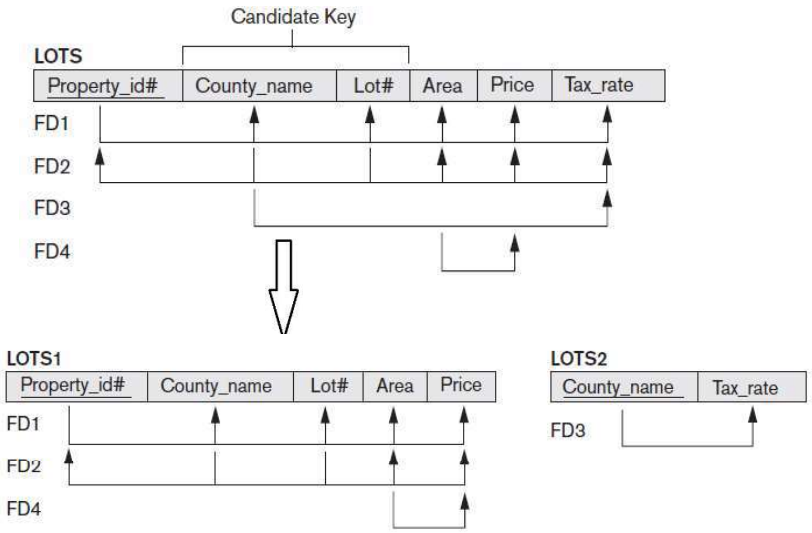
Desirable Properties of Decompositions

- Not all decomposition of a schema are useful
- We require two properties to be satisfied:
  i) Dependency Preservation Property
  ii) Nonadditive (Lossless) Join Property

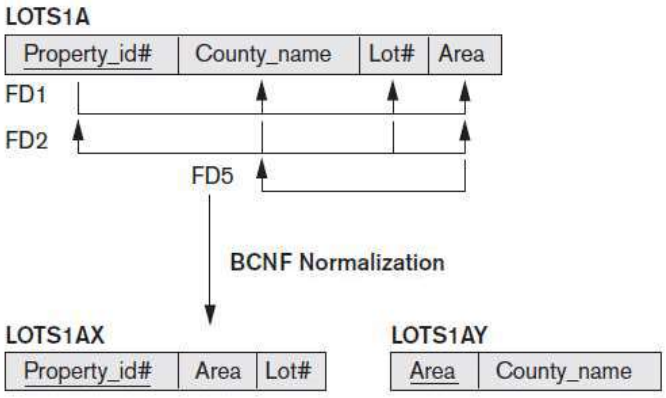
Dependency Preservation Property

- Each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relation schemas R_i in the decomposition D or could be inferred from the dependencies that appear in some R_i

- We want to preserve the dependencies because each dependency in *F* represents a constraint on the database
- If one of the dependencies is not represented in some individual relation *Ri* of the decomposition, we cannot enforce this constraint by dealing with an individual relation
- It is not necessary that the exact dependencies specified in *F* appear themselves in individual relations of the decomposition *D*.
- It is sufficient that the union of the dependencies that hold on the individual relations in *D* be equivalent to *F*
- **Example:** Dependency Preserving Decomposition



- **Example:** Decomposition that does not Preserve Dependency



Nonadditive (Lossless) Join Property

- The nonadditive join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations
- The term **lossy design** refer to a design that represents a loss of information
- 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

Algorithm 16.3. Testing for Nonadditive Join Property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, ..., R_m\}$ 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

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

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

Testing Binary Decompositions for the Nonadditive Join Property

Property NJB (Nonadditive Join Test for Binary Decompositions). A decomposition $D = \{R_1, R_2\}$ of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R *if and only if* either

- The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or
- The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+

4.13 Algorithms for Relational Database Schema Design

Two algorithms for creating a relational decomposition from universal relation

1. The first algorithm decomposes a universal relation into dependency preserving 3NF relations that also possess the nonadditive join property
2. The second algorithm decomposes a universal relation schema into BCNF schemas that possess the nonadditive join property

4.13.1 Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

Algorithm 16.4. Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property

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

1. Find a minimal cover *G* for *F* (use Algorithm 16.2).

2. For each left-hand-side *X* of a functional dependency that appears in *G*, create a relation schema in *D* with attributes {*X* ∪ {*A*1} ∪ {*A*2} ... ∪ {*Ak*}}, where *X*→*A*1, *X*→*A*2, ..., *X*→*Ak* are the only dependencies in *G* with *X* as left-hand-side (*X* is the key of this relation)

5 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*

6 Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation *R* is considered redundant if *R* is a projection of another relation *S* in the schema; alternately, *R* is subsumed by *S*

- **Example:** Consider the following universal relation:

  U(Emp_ssn, Pno, Esal, Ephone, Dno, Pname, Plocation)

- Emp_ssn, Esal, Ephone refer to the Social Security number, salary, and phone number of the employee. Pno, Pname, and Plocation refer to the number, name, and location of the project. Dno is department number.
- 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 universal relation.
- Hence *F*, the set of given FDs includes {Emp_ssn → Esal, Ephone, Dno; Pno→Pname, Plocation; Emp_ssn, Pno→Esal, Ephone, Dno, Pname, Plocation}.
- By applying the minimal cover , in step 3 we see that Pno is a redundant attribute in Emp_ssn, Pno → Esal, Ephone, Dno. Moreover, Emp_ssn is redundant in Emp_ssn, Pno→Pname, Plocation.
- Hence the minimal cover consists of FD1 and FD2 only
- Minimal cover *G*: {Emp_ssn → Esal, Ephone, Dno; Pno → Pname, Plocation}
- By applying Algorithm 16.4 to the above Minimal cover *G*, we get a 3NF design consisting of two relations with keys Emp_ssn and Pno as follows:

  *R*1 (Emp_ssn, Esal, Ephone, Dno)

  *R*2 (Pno, Pname, Plocation)
- In step 3, we generate a relation corresponding to the key(Emp_ssn,Pno) of U. Hence, the resulting design contains:

  *R*1 (Emp_ssn, Esal, Ephone, Dno)

  *R*2 (Pno, Pname, Plocation)

  R3(Emp_ssn, Pno)

This design achieves both the desirablle properties of dependency preservation and non additive join

4.13.2 Nonadditive Join Decomposition into BCNF Schemas

Algorithm 16.5. Relational Decomposition into BCNF with Nonadditive Join Property

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

1. Set *D* := {*R*} ;

2. While there is a relation schema *Q* in *D* that is not in BCNF do

{

choose a relation schema *Q* in *D* that is not in BCNF;

find a functional dependency $X \rightarrow Y$ in Q that violates BCNF;
replace Q in D by two relation schemas ($Q - Y$) and ($X \cup Y$);
} ;

- Each time through the loop in Algorithm 16.5, we decompose one relation schema Q that is not in BCNF into two relation schemas.
- According to Property NJB for binary decompositions and Claim 2, the decomposition D has the nonadditive join property
- At the end of the algorithm, all relation schemas in D will be in BCNF
- Example:TEACH relation schema decomposed into TEACH1(Instructor, Student) and TEACH2(Instructor, Course) because the dependency FD2 Instructor→Course violates BCNF.
- In step 2 of Algorithm 16.5, it is necessary to determine whether a relation schema Q is in BCNF or not.
- whenever a relation schema Q has a BCNF violation, there exists a pair of attributes A and B in Q such that $\{Q - \{A, B\}\} \rightarrow A$; by computing the closure $\{Q - \{A, B\}\}+$ for each pair of attributes $\{A, B\}$ of Q, and checking whether the closure includes A (or B), we can determine whether Q is in BCNF.

4.13.3 Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

- It is *not possible to have all three of the following:*
  (1) guaranteed nonlossy design,
  (2) guaranteed dependency preservation, and
  (3) all relations in BCNF
- The first condition is a must and cannot be compromised.
- The second condition is desirable, but not a must, and may have to be relaxed if we insist on achieving BCNF.
- Now we give an alternative algorithm where we achieve conditions 1 and 2 and only guarantee 3NF.
- A simple modification to Algorithm 16.4, shown as Algorithm 16.6, yields a decomposition D of R that does the following:
  - Preserves dependencies
  - Has the nonadditive join property

- Is such that each resulting relation schema in the decomposition is in 3NF

Algorithm 16.6. Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property

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

1. Find a minimal cover G for F (use Algorithm 16.2).
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k$ 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.[7] (Algorithm 16.2(a) may be used to find a key.)
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S.

- Step 3 involves identifying a key K of R. Algorithm 16.2(a) can be used to identify a key K of R based on the set of given functional dependencies F.

Example 1 of Algorithm 16.6. Let us revisit the example given earlier at the end of Algorithm 16.4. The minimal cover G holds as before. The second step produces relations R_1 and R_2 as before. However, now in step 3, we will generate a relation corresponding to the key {Emp_ssn, Pno}. Hence, the resulting design contains:

R_1 (Emp_ssn , Esal, Ephone, Dno)
R_2 (Pno, Pname, Plocation)
R_3 (Emp_ssn, Pno)

This design achieves both the desirable properties of dependency preservation and nonadditive join.

4.14 About Nulls, Dangling Tuples, and Alternative Relational Designs

4.14.1 Problems with NULL Values and Dangling Tuples

- Whenever a relational database schema is designed in which two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential NULL values in foreign keys.
- This can cause unexpected loss of information in queries that involve joins on that foreign key.

- If NULLs occur in other attributes, such as Salary, their effect on built-in functions such as SUM and AVERAGE must be carefully evaluated.

Dangling tuples may occur if we carry a decomposition too far. Suppose that we decompose the EMPLOYEE relation in Figure 16.2(a) further into EMPLOYEE_1 and EMPLOYEE_2, shown in Figure 16.3(a) and 16.3(b)

(a)

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
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	291 Berry, 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

DEPARTMENT

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

(b)

Ename	Ssn	Bdate	Address	Dnum	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 Fire Oak, 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

(c)

Ename	Ssn	Bdate	Address	Dnum	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 Fire Oak, 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
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL	NULL	NULL
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX	NULL	NULL	NULL

Figure 16.2: Issues with NULL-value joins. (a) Some EMPLOYEE tuples have NULL for the join attribute Dnum (b) Result of applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT relations. (c) Result of applying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT.

(a) **EMPLOYEE_1**

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

(b) **EMPLOYEE_2**

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) **EMPLOYEE_3**

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure 16.3: The dangling tuple problem. (a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 16.2(a) except Dnum). (b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values). (c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

- If we apply the NATURAL JOIN operation to EMPLOYEE_1 and EMPLOYEE_2, we get the original EMPLOYEE relation.
- we may use the alternative representation, shown in Figure 16.3(c), where we *do not include a tuple*in EMPLOYEE_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE_2).
- If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result; these are called **dangling tuples** in EMPLOYEE_1 because they are represented in only one of the two relations that represent employees, and hence are lost if we apply an (INNER) JOIN operation.

4.15Other Dependencies and Normal Forms

4.15.1 Inclusion Dependencies

Inclusion dependencies were defined in order to formalize two types of interrelational constraints:

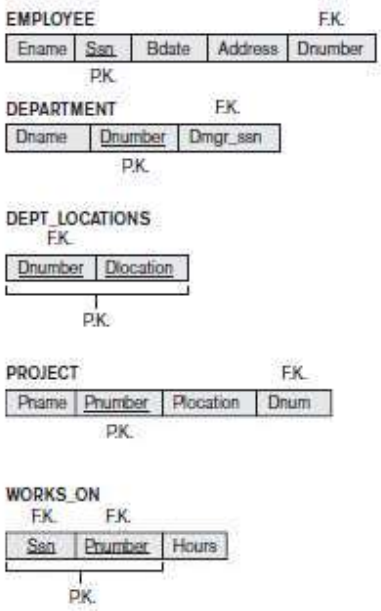
- The foreign key (or referential integrity) constraint cannot be specified as a functional or multivalued dependency because it relates attributes across relations.
- The constraint between two relations that represent a class/subclass relationship also has no formal definition in terms of the functional,multivalued, and join dependencies.

Definition. An **inclusion dependency** $R.X < S.Y$ between two sets of attributes—X of relation schema R, and Y of relation schema S—specifies the constraint that, at any specific time when r is a relation state of R and s a relation state of S, we must have

$$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

- The subset relationship does not necessarily have to be a proper subset. Obviously, the sets of attributes on which the inclusion dependency is specified—X of R and Y of S—must have the same number of attributes.
- In addition, the domains for each pair of corresponding attributes should be compatible.
- For example, we can specify the following inclusion dependencies on the relationalvschema in Figure 15.1:

Figure 15.1
A simplified COMPANY relational database schema.



- DEPARTMENT.Dmgr_ssn < EMPLOYEE.Ssn
- WORKS_ON.Ssn < EMPLOYEE.Ssn
- EMPLOYEE.Dnumber < DEPARTMENT.Dnumber
- PROJECT.Dnum < DEPARTMENT.Dnumber
- WORKS_ON.Pnumber < PROJECT.Pnumber
- DEPT_LOCATIONS.Dnumber < DEPARTMENT.Dnumber

- All the preceding inclusion dependencies represent **referential integrity constraints**.

- We can also use inclusion dependencies to represent **class/subclass.**For example, in the relational schema of Figure 9.6, we can specify the following inclusion dependencies:
  - EMPLOYEE.Ssn < PERSON.Ssn
  - ALUMNUS.Ssn < PERSON.Ssn
  - STUDENT.Ssn < PERSON.Ssn

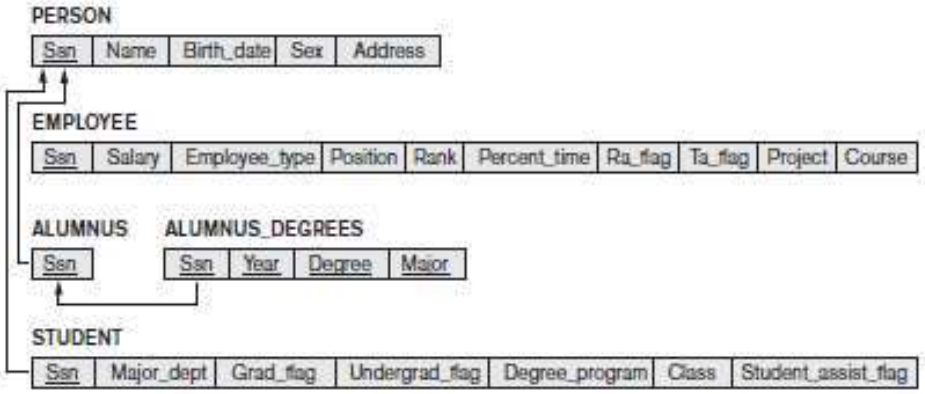


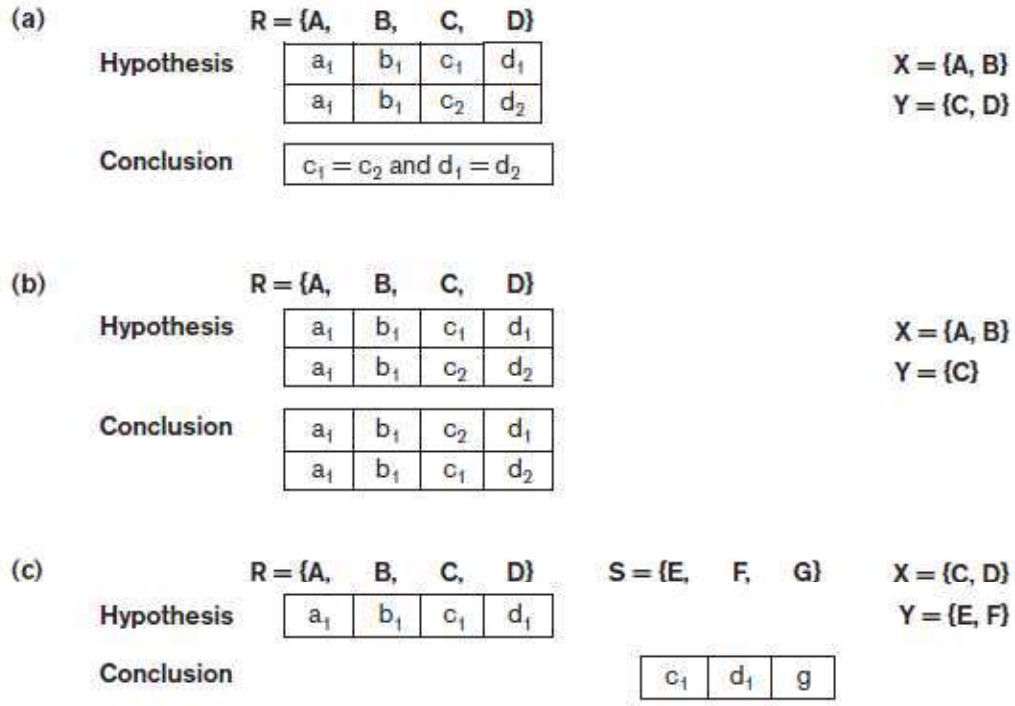
Figure 9.6
Mapping the EER specialization lattice in Figure 8.8 using multiple options.

4.15.2 Template Dependencies

- Template dependencies provide a technique for representing constraints in relations that typically have no easy and formal definitions.
  - There are two types of templates:
    - tuple-generating templates and
    - constraint generating templates.
  - A template consists of a number of **hypothesis tuples** that are meant to show an example of the tuples that may appear in one or more relations.
  - The other part of the template is the **template conclusion**.
  - For tuple-generating templates, the conclusion is a *set of tuples* that must also exist in the relations if the hypothesis tuples are there.

- For constraint-generating templates, the template conclusion is a *condition* that must hold on the hypothesis tuples.
- Using constraint generating templates, we are able to define **semantic constraints—**those that are beyond the scope of the relational model in terms of its data definition language and notation.
- Figure 16.5 shows how we may define functional, multivalued, and inclusion dependencies by templates.

Figure 16.5
Templates for some common type of dependencies.
(a) Template for functional dependency X → Y.
(b) Template for the multivalued dependency X →→ Y.
(c) Template for the inclusion dependency R.X < S.Y.



- Figure 16.6 shows how we may specify the constraint that an employee's salary cannot be higher than the salary of his or her direct supervisor on the relation schema EMPLOYEE

Figure 16.6
Templates for the constraint that an employee's salary must be less than the supervisor's salary.

EMPLOYEE = {Name, Ssn, . . . , Salary, Supervisor_ssn}

	a	b	c	d
Hypothesis	e	d	f	g
Conclusion			c < f	

4.15.3 Functional Dependencies Based on Arithmetic Functions and Procedures

- Sometimes some attributes in a relation may be related via some arithmetic function or a more complicated functional relationship.
- As long as a unique value of *Y* is associated with every *X*, we can still consider that the FD *X*→*Y* exists.
- For example, in the relation
  ORDER_LINE (Order#, Item#, Quantity, Unit_price, Extended_price, Discounted_price)
- each tuple represents an item from an order with a particular quantity, and the price per unit for that item.
- In this relation, (Quantity, Unit_price)→Extended_price by the formula
  Extended_price = Unit_price * Quantity.
- Hence, there is a unique value for Extended_price for every pair (Quantity, Unit_price), and thus it conforms to the definition of functional dependency.
- Moreover, there may be a procedure that takes into account the quantity discounts, the type of item, and so on and computes a discounted price for the total quantity ordered for that item.
- Therefore, we can say
  (Item#, Quantity, Unit_price) → Discounted_price, or
  (Item#, Quantity, Extended_price) → Discounted_price.

4.15.4 Domain-Key Normal Form

- The idea behind **domain-key normal form** (**DKNF**) is to specify the *ultimate normal form* that takes into account all possible types of dependencies and constraints.

- A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- For example, consider a relation CAR(Make, Vin#) (where Vin# is the vehicle identification number) and another relation MANUFACTURE(Vin#,Country) (where Country is the country of manufacture).
- A general constraint may be of the following form: *If the Make is either 'Toyota' or 'Lexus,' then the first character of the Vin# is a 'J' if the country of manufacture is 'Japan'; if the Make is 'Honda' or 'Acura,' the second character of the Vin# is a 'J' if the country of manufacture is 'Japan.'*
- There is no simplified way to represent such constraints short of writing a procedure (or general assertions) to test them.
- The procedure COMPUTE_TOTAL_PRICE above is an example of such procedures needed to enforce an appropriate integrity constraint.

Problem 1

Consider the following relation for published books:

BOOK(BookTitle, AuthorName, BookType, ListPrice, AuthorAffiliation, Publisher)

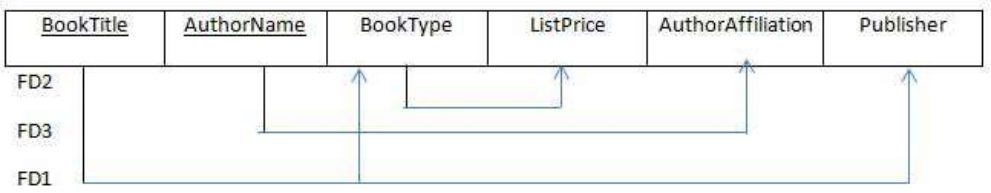
Suppose the following dependencies exist:

- **BookTitle → BookType, Publisher**
- **BookType → ListPrice**
- **AuthorName → AuthorAffiliation**

What normal form is the relation in? explain your answer. Apply normalization until you cannot decompose the relations further. State the reasons behind each decomposition.

Solution:

The relation is in 1NF and not in 2NF as no attributes are fully functionally dependent on the key (BookTitle and AuthorName). It is also not in 3NF.



• The relation is not in 2NF because the partial Dependencies exist

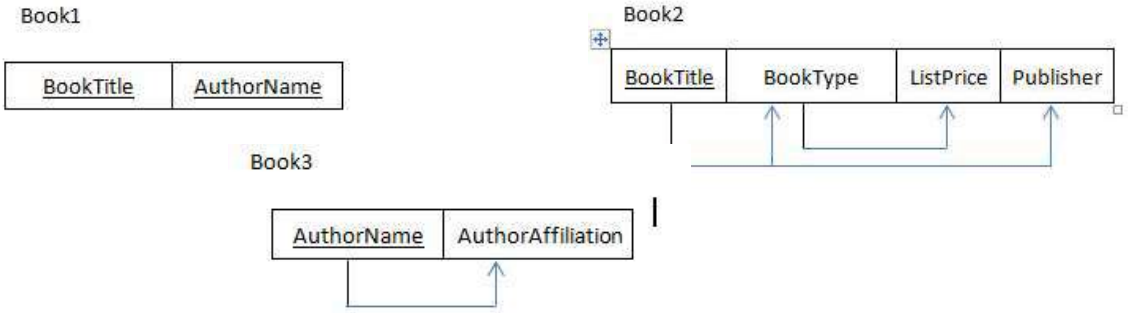
{BookTitle,AuthorName} → {Publisher, BookType}

{BookdTitle,AuthorName} → AuthorAffiliation

•Thus, these attributes are not fully functionally dependent on the primary key The 2NF decomposition will eliminate the partial dependencies.

•2NF decomposition:

- Book1(BookTitle, AuthorName)
- Book2(BookTitle, BookType, ListPrice, Publisher)
- Book3(AuthorName, AuthorAffiliation)

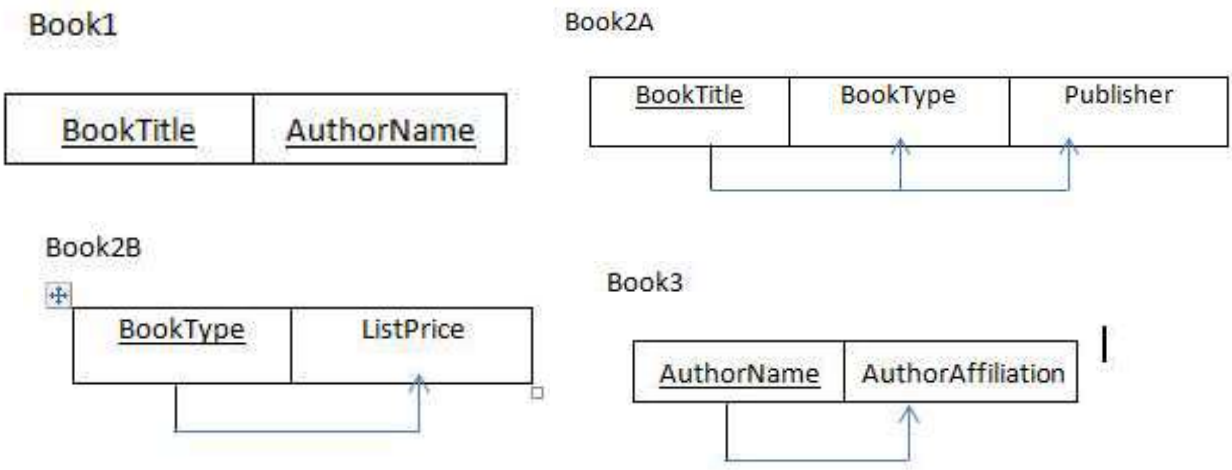


- The relations are not in 3NF because:

• BookTitle → BookType → ListPrice

BookType is neither a key itself nor a subset of a key and ListPrice is not a prime attribute

- The 3NF decomposition will eliminate the transitive dependency of Listprice. 3NF decomposition:
  - Book1(BookTitle, AuthorName)
  - Book2A(BookTitle, BookType, Publisher)
  - Book2B(BookType, ListPrice)
  - Book3(AuthorName, AuthorAffiliation)



Problem 2

Consider the following relation:

CAR_SALE(Car#, DateSold, Salesman#, Commission%, DiscountAmount)

Assume that a car may be sold by multiple salesmen, and hence

{Car#, Salesman#} is the primary key.

Additional dependencies are:

Car# → DateSold

Car# → DiscountAmount

DateSold → DiscountAmount

Salesman# → Commission%

Based on the given primary key, is the relation in 1NF, 2NF, 3NF?

Why or why not?

How would you successively normalize it completely?

Solution:

- The relation is in 1NF because all attribute values are single atomic values.
- The relation is not in 2NF because:
  - Car# → DateSold
  - Car# → DiscountAmount
  - Salesman# → Commission%

Thus, these attributes are not fully functionally dependent on the primary key.

- 2NF decomposition:
  - CAR_SALE1(Car#, DateSold, DiscountAmount)
  - CAR_SALE2(Car#, Salesman#)
  - CAR_SALE3(Salesman#, Commission%)
- The relations are not in 3NF because:
  - Car# → DateSold → DiscountAmount

DateSold is neither a key itself nor a subset of a key and DiscountAmount is not a prime attribute.

- 3NF decomposition:
  - CAR_SALES1A(Car#, DateSold)
  - CAR_SALES1B(DateSold, DiscountAmount)
  - CAR_SALE2(Car#, Salesman#)
  - CAR_SALE3(Salesman#, Commission%)

4.16 Assignment Questions

1. Consider the following relation for published books:

   BOOK(BookTitle, AuthorName, BookType, ListPrice, AuthorAffiliation, Publisher)

   Suppose the following dependencies exist:

   BookTitle → BookType, Publisher

   BookType → ListPrice

   AuthorName → AuthorAffiliation

   What normal form is the relation in? Explain your answer.

2. Consider the following relation:

   CAR_SALE(Car#, DateSold, Salesman#, Commission%, DiscountAmount)

   Assume that a car may be sold by multiple salesmen, and hence

   {Car#, Salesman#} is the primary key.

   Additional dependencies are:

   Car# → DateSold

Car# → DiscountAmount

DateSold → DiscountAmount

Salesman# → Commission%

Based on the given primary key, is the relation in 1NF, 2NF, 3NF?

Why or why not?

How would you successively normalize it completely?

3. Let R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} and 0 = {RI, R2, R3} where
   RI = EMP = {Ssn, Ename}
   R2 = PRO] = {Pnumber, Pname, Plocation}
   R3 = WORKS-ON = {Ssn, Pnumber, Hours}
   The following functional dependencies hold on relation R.
   F = {Ssn ->Ename; Pnumber -> {Pname, Plocation};
   {Ssn, Pnumber} -> Hours}
   Prove that the above decomposition of relation R has the loss less join property.
4. Consider R={A B C D E F} FDS {AB-> B->E A->DF}
   Check whether decomposition is lossless.
5. What is a set of functional dependencies F said to be minimal? Give an algorithm for finding a minimal cover G for F.

4.17 Expected Outcome

- To design a database which will have minimum redundancy
- To apply normalization to the designed database.
- To decompose the tables and normalize the design upto 4NF and 5 NFthe tables upto 4NF and 5NF
- To apply lossless and lossy join operations
- To apply inference rules and deduce other rules from the given set.

4.18 Further Reading

1. https://www.smartdraw.com/entity-relationship-diagram/
2. https://en.wikipedia.org/wiki/Database_normalization
3. www.databasteknik.se/webbkursen/relalg-lecture
4. https://technet.microsoft.com/en-us/library/bb264565(v=sql.90).aspx
5. pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/.../Ch16_Overview_Xacts.pdf