Chapter 2: Normalization

Chapter 2: Normalization

- Purpose of Normalization
- Data Redundancy and Update Anomalies
- Functional Dependencies
- Identifying Functional Dependencies
- Identifying the Primary key using Functional Dependencies
- The Process of Normalization
- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

Relational Database Design

- The goal of a relational-database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily.
- One approach is to design schemas that are in an appropriate *normal form*, known as ***normalization***.
  - Begins by examining the relationships called FDs between attributes
  - Uses series of tests (Normal forms) to help identify grouping of attributes

Purpose of Normalization

The purpose of normalization is **to identify a suitable set of relations** that support the data requirements of an enterprise.

- The characteristics of a suitable set of relations include the following:
  - **the *minimal number of attributes*** *necessary to support the data requirements of the* enterprise;
  - ***minimal redundancy*** *with each attribute.*
  - **attributes with a close logical relationship** (described as functional dependency) are found in the same relation;
- *Can be used at any stage of DB design*

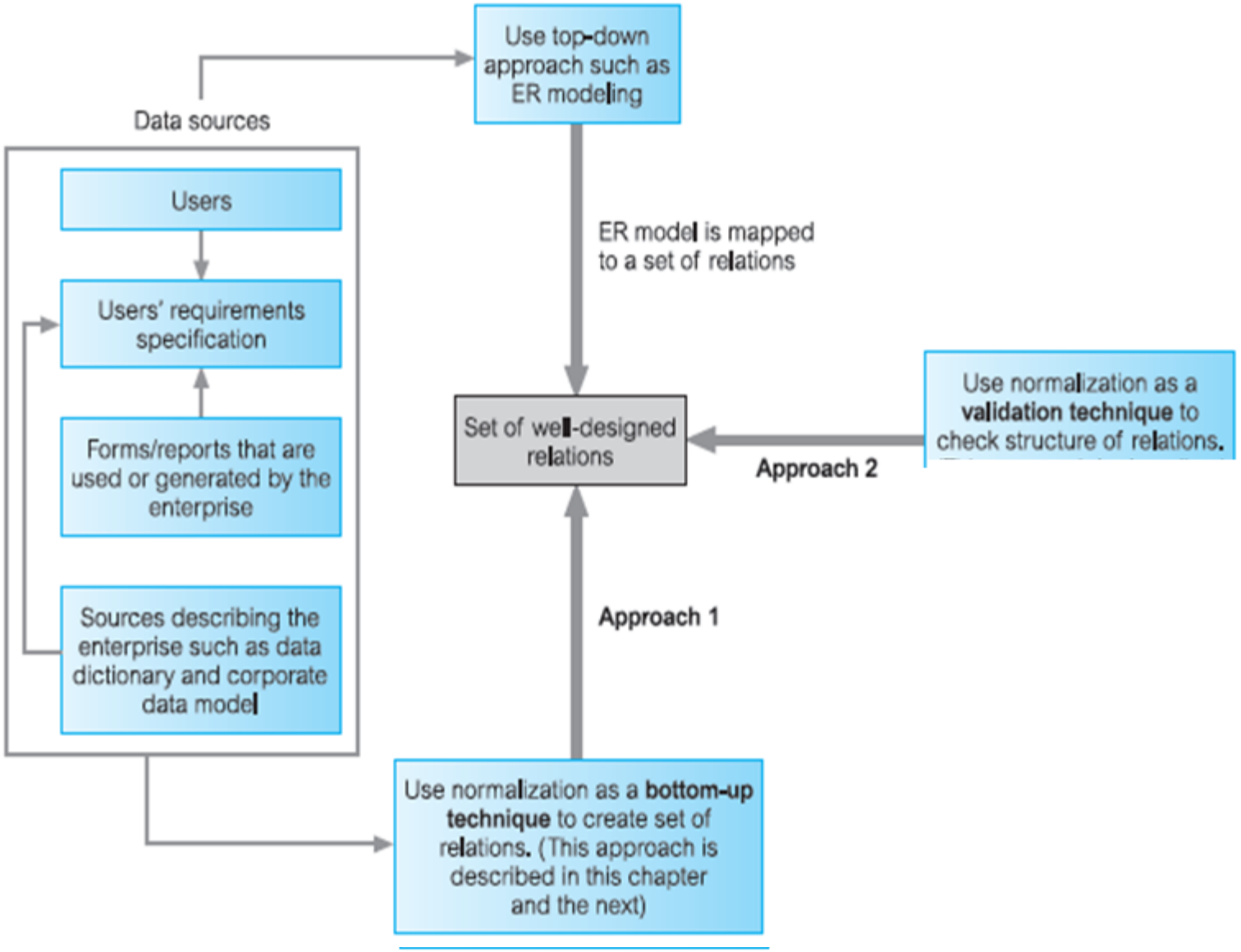
Purpose of Normalization

- A large database defined as a single relation may result in data duplication. This repetition of data may result in:
  - Making relations very large.
  - It isn't easy to maintain and update data as it would involve searching many records in relation.
  - Wastage and poor utilization of disk space and resources.
  - The likelihood of errors and inconsistencies increases.
- So to handle these problems, **we should analyze and decompose the relations with redundant data into smaller**, simpler, and well-structured relations that are satisfy desirable properties.
- **Normalization is a process of decomposing the relations into relations with fewer attributes.**

What is Normalization

- Normalization is the process of organizing the data in the database.
- is used to minimize the redundancy from a relation or set of relations.
- It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

How Normalization Supports Database Design



Property Database

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner	(ownerNo, fName, lName, address, telNo)
Viewing	(clientNo, propertyNo, viewDate, comment)
Registration	(clientNo, branchNo, staffNo, dateJoined)

Data Redundancy and Update Anomalies

- A major aim of relational database design is to group attributes into relations to minimize data redundancy.
- If this goal is achieved, then
  - updates to the data stored in the database are achieved with a minimal number of operations thus reducing the opportunities for data inconsistencies occurring in the database;
  - reduction in the file storage space required by the base relations thus minimizing costs.
- Relations that have redundant data may have problems called update anomalies

Data Redundancy and Update Anomalies

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

StaffBranch (staffNo, sName, position, salary, branchNo, bAddress)

Data Redundancy and Update Anomalies

- The relations have the form:

  StaffBranch (staffNo, sName, position, salary, branchNo, bAddress)

- Redundant data in StaffBranch
- Relations that have redundant data may have problems called **update anomalies,** which are classified as
  - insertion,
  - deletion, or
  - modification anomalies.

update anomalies

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Insertion Anomalies

- There are two main types of insertion anomaly. Consider StaffBranch relation:

1. To insert the details of new members of staff into the StaffBranch relation, we must include the details of the branch at which the staff are to be located.
2. To insert details of a new branch that currently has no members of staff into the StaffBranch relation, it is necessary to enter nulls into the attributes for staff, such as staffNo.

   However, as staffNo is the primary key for the StaffBranch relation, attempting to enter nulls for staffNo violates entity integrity, and is not allowed. We therefore cannot enter a tuple for a new branch into the StaffBranch relation with a null for the staffNo.

Deletion Anomalies

- If we delete a tuple from the StaffBranch relation that represents the last member of staff located at a branch, the details about that branch are also lost from the database.

Modification Anomalies

- If we want to change the value of one of the attributes of a particular branch in the StaffBranch relation, for example the address for branch number B003, we must update the tuples of all staff located at that branch.
- If this modification is not carried out on all the appropriate tuples of the StaffBranch relation, the database will become inconsistent.

Decomposition

- The StaffBranch relation is subject to update anomalies, we can avoid these anomalies by decomposing the original relation into the Staff and Branch relations.
- The Staff and Branch relations have more desirable properties than the StaffBranch relation.
- There are two important properties associated with decomposition of a larger relation into smaller relations:

1. The **lossless-join** property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.
2. The **dependency preservation** property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations.

Data Redundancy and Update Anomalies

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Staff (staffNo, sName, position, salary, branchNo)
Branch (branchNo, bAddress)

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Functional Dependencies

- An important concept associated with normalization is **functional dependency,** which describes the relationship between attributes.
- Functional dependencies play a key role in differentiating good database design from bad database design.
- It's a constraints on the set of legal relations.
- Functional dependencies requires that the value for a certain set of attributes uniquely determines the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key.*

Functional Dependencies (Cont.)

- Let R be a relation schema

  $\alpha \subseteq R$ *and* $\beta \subseteq R$

- The functional dependency

  $\alpha \rightarrow \beta$

  holds on R if and only if, for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

  $t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$

  or α uniquely indentify β

- Example: Consider $r(A,B)$ with the following instance of r.

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Functional Dependencies (Cont.)

A	B
1	4
1	4
2	4
3	7

On this instance, $A \rightarrow B$ does hold, but $B \rightarrow A$ does **NOT** hold

A	C
a_1 c_1 a_1 c_1	
a_2 c_2	
a_2 a_3 c_2	c_2

On this instance, $A \rightarrow C$ does hold, but $C \rightarrow A$ does **NOT** hold

Sample Relation *r*

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Satisfied:

$A \rightarrow C$

$D \rightarrow$ B

AB $\rightarrow D$

Not Satisfied:

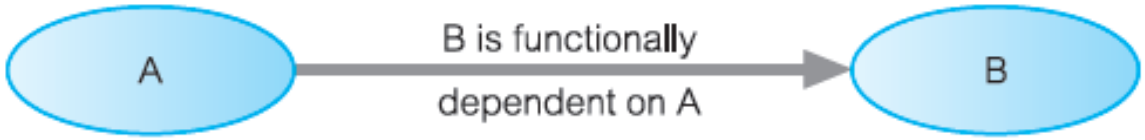
$C \rightarrow A$

B $\rightarrow D$

A $\rightarrow D$

Functional Dependencies (Cont.)

- **Determinant:** When a functional dependency exists, the attribute or group of attributes on the left hand side of the arrow is called the **determinant.**
- For example, following figure A is the determinant of B.



Example of a functional dependency

Staff

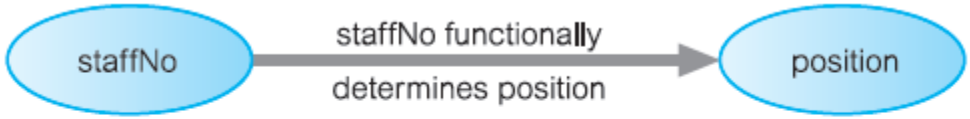
staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Example of a functional dependency

- Consider the attributes staffNo and position of the Staff relation.

(a) staffNo functionally determines position

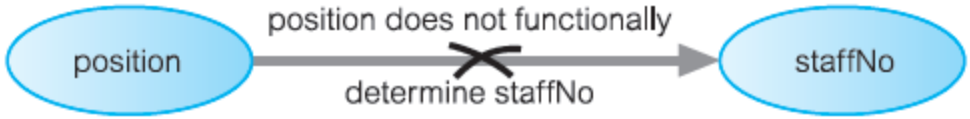
(staffNo → position)



(a)

(b) position does *not functionally* determine staffNo

(position → staffNo).



(b)

Example of a functional dependency

- In this example, staffNo is the determinant of this functional dependency.
- For the purposes of normalization we are interested in identifying functional dependencies between attributes of a relation that have a one-to-one relationship between the attribute(s) that makes up the determinant on the left-hand side and the attribute(s) on the right-hand side of a dependency.

Example of a functional dependency that holds for all time

- Consider the values staffNo and sName attributes of the Staff relation.
- Initially, the staffNo attribute functionally determines the sName attribute and/or that the sName attribute functionally determines the staffNo attribute :

  staffNo →sName

  sName →staffNo

- But it is possible for the sName attribute to hold duplicate values for members of staff with the same name, then for some members of staff in this category we would not be able to determine their staff number (staffNo).
- we want to identify functional dependencies that hold for all possible values for attributes of a relation as these represent the types of integrity constraints that we need to identify

Example of a functional dependency

- The relationship between staffNo and sName is one-to-one (1:1): for each staff number there is only one name.
- On the other hand, the relationship between sName and staffNo is one-to-many (1:*): there can be several staff numbers associated with a given name.
- The functional dependency that remains true after consideration of all possible values for the staffNo and sName attributes of the Staff relation is:
    - staffNo →sName

Full Functional Dependency

- An additional characteristic of functional dependencies that is useful for normalization is that their determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side. This requirement is called **full functional dependency.**
- A functional dependency A → B is a ***full functional dependency*** if removal of any attribute from A results in the dependency no longer existing.
- Indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A, but not **on any proper subset of A.**
- A functional dependency A → B is a **partially dependency** if there is some attributes (extraneous attributes) that can be removed from A and yet the dependency still holds.

- Example:
  - staffNo, sName →branchNo : partial dependency
  - staffNo →branchNo : full functional dependency
- It is correct to say that each value of (staffNo, sName) is associated with a single value of branchNo.
- However, it is not a full functional dependency because branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo.
- In other words, the functional dependency shown above is an example of a partial dependency.

Characteristics of functional dependencies

- The functional dependencies that we use in normalization have the following characteristics:

1. There is a *one-to-one* relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency.
2. They hold for *all time.*
3. The determinant has the *minimal number* of attributes necessary to maintain *the* dependency with the attribute(s) on the right-hand side. In other words, there must be a full functional dependency between the attribute(s) on the left- and right-hand sides of the dependency.

Transitive Functional Dependency

- A condition where A, B, and C are attributes of a relation such that if **A → B** and **B → C,** then C is transitively dependent on A via B (**A → C).**
- Example:
  - staffNo ⟶sName, position, salary, branchNo, bAddress
  - branchNo ⟶bAddress
  - staffNo → bAddress

The staffNo attribute functionally determines the bAddress via the branchNo attribute.

Trivial and Nontrivial Functional dependency

- Functional dependency A->B is a **trivial** dependency if B is a subset of A
  - Eg: staffNo sName → sName
- Functional dependency A->B is a **nontrivial** dependency if B is not a subset of A
  - Eg: staffNo → sName

Identifying Functional Dependencies

- Identifying all functional dependencies between a set of attributes should be quite simple **if the meaning of each attribute and the relationships between the attributes are well understood.**

Identifying Functional Dependencies

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

- Example: Consider attributes in StaffBranch relation.
- The functional dependencies based on our understanding of the attributes in the relation as:
    - staffNo →sName, position, salary, branchNo, bAddress
    - branchNo →bAddress
    - bAddress →branchNo
    - branchNo, position →salary
    - bAddress, position →salary
- With staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position) as determinants.

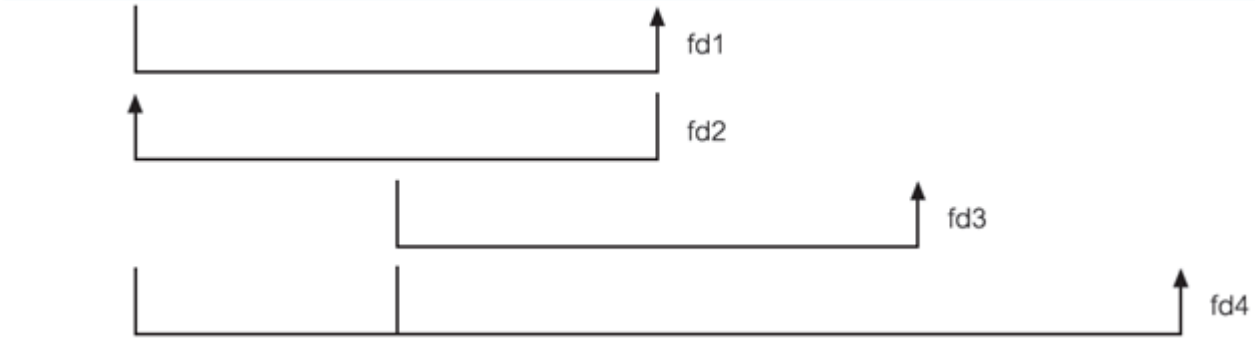
Identifying Functional Dependencies

- However, if the users are unavailable for consultation and/or the documentation is incomplete, then, depending on the database application, it may be necessary for the database designer to use their common sense and/or experience to provide the missing information.
- Example: Consider the data for attributes denoted A, B, C, D, and E in the Sample relation of Figure next.

Identifying Functional Dependencies

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



- A → C (fd1)
- C → A (fd2)
- B → D (fd3)
- A, B → E (fd4)

Identifying the primary key for the StaffBranch relation

- To identify the candidate key(s) for the StaffBranch relation, we must identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation.
- If a relation has more than one candidate key, we identify the candidate key that is to act as the primary key for the relation (see Section 3.2.5).
- All attributes that are not part of the primary key (non-primary-key attributes) should be functionally dependent on the key.
- The only candidate key of the StaffBranch relation, and therefore the primary key, isstaffNo, as *all other attributes of the relation are functionally dependent on staffNo.*

Identifying the primary key for the Sample relation

- The determinants in the Sample relation are A, B, C, and (A, B).
- A → C (fd1), C → A (fd2),
- B → D (fd3), A, B → E (fd4)
- However, the only determinant that functionally determines all the other attributes of the relation is (A, B). In particular, A functionally determines C, B functionally determines D, and (A, B) functionally determines E.
- In other words, the attributes that make up the determinant (A, B) can determine all the other attributes in the relation either separately as A or B or together as (A, B).
- As there are no other candidate keys for the Sample relation (A, B) is identified as the primary key for this relation.

Inference Rules for Functional Dependencies

- From given set of functional dependencies we can derive another set of functional dependencies.
- The set of all functional dependencies that are implied by a given set of functional dependencies X is called the closure of X, written X^{+}.
- A set of inference rules, called Armstrong's axioms are used to compute X^{+} from X.
- Let A, B, and C be subsets of the attributes of the relation R. Armstrong's axioms are as follows:

  (1) Reflexivity: If B is a subset of A, then A → B (**trivial FD**)

  (2) Augmentation: If A → B, then A,C → B,C

  (3) Transitivity: If A → B and B → C, then A → C

- These rules are
  - sound (do not generate any incorrect functional dependencies) and
  - complete (generate all functional dependencies that hold).

Inference Rules for Functional Dependencies

- Several further rules can be derived from the three given above that simplify the practical task of computing X^+.
- In the following rules, let D be another subset of the attributes of relation R, then:

  (4) Self-determination: A → A (**trivial FD**)

  (5) Decomposition: If A → B,C, then A → B and A → C

  (6) Union: If A → B and A → C, then A → B,C

  (7) Composition: If A → B and C → D then A,C → B,D

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$ *:* "union rule" can be inferred from
      - definition of functional dependencies, or
  - Similarly, $A \rightarrow BC$ by union rule.

Closure of attribute set

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.
- Closure of attribute set {A} is denoted as $\{A\}^+$.

1: Add the attributes contained in the attribute set for which closure is being calculated to the result set.

2: Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Closure of attribute set

- Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-
  - $A \rightarrow BC$
  - $BC \rightarrow DE$
  - $D \rightarrow F$
  - $CF \rightarrow G$
- $A^+ = \{ A \}$

  $= \{ A , B , C \}$ (Using $A \rightarrow BC$)

  $= \{ A , B , C , D , E \}$ (Using $BC \rightarrow DE$)

  $= \{ A , B , C , D , E , F \}$ (Using $D \rightarrow F$)

  $= \{ A , B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)

  Thus,

  $A^+ = \{ A , B , C , D , E , F , G \}$

Closure of attribute set

- $\{ B , C \}^{+} = \{ B , C \}$

  $= \{ B , C , D , E \}$ (Using $BC \rightarrow DE$)

  $= \{ B , C , D , E , F \}$ (Using $D \rightarrow F$)

  $= \{ B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)

  Thus,

  $\{ B , C \}^{+} = \{ B , C , D , E , F , G \}$

Identifying Keys for a Relation using Functional Dependencies

- The main purpose of identifying a set of functional dependencies for a relation is to specify the **set of integrity constraints** that must hold on a relation.
- An important integrity constraint to consider **first is the identification of candidate keys, one of which is selected to be the primary key for the relation.**

Identifying the Keys Using Closure

- If the closure result of an attribute set contains **all the attributes** of the relation, then that attribute set is called as a **super key** of that relation.
  - **A^+ = { A , B , C , D , E , F , G }**
  - The closure of attribute A is the entire relation schema.
  - Thus, attribute A is a super key for that relation.
- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.
  - No subset of attribute A contains all the attributes of the relation.
  - Thus, attribute A is also a candidate key for that relation.

Identifying the Keys Using Closure

- Consider a relation R (A , B , C , D , E) with the functional dependencies-
  - $A \rightarrow BC$
  - $CD \rightarrow E$
  - $B \rightarrow D$
  - $E \rightarrow A$
- $A^+ = \{ A \}$

  $= \{ A , B , C \}$ (Using $A \rightarrow BC$)

  $= \{ A , B , C , D \}$ (Using $B \rightarrow D$)

  $= \{ A , B , C , D , E \}$ (Using $CD \rightarrow E$)

  Thus,

  $A^+ = \{ A , B , C , D , E \}$
  - The closure of attribute A is the entire relation schema. Thus, attribute A is a super key for that relation.
  - No subset of attribute A contains all the attributes of the relation. Thus, attribute A is also a candidate key for that relation.

Identifying the Keys Using Closure

- $CD^+ = \{ C, D \}$

  $= \{ C , D , E \}$ (Using $CD \rightarrow E$)

  $= \{ A , C , D , E \}$ (Using $E \rightarrow A$)

  $= \{ A , B , C , D , E \}$ (Using $A \rightarrow BC$)

  Thus,

  $CD^+ = \{ A , B , C , D , E \}$

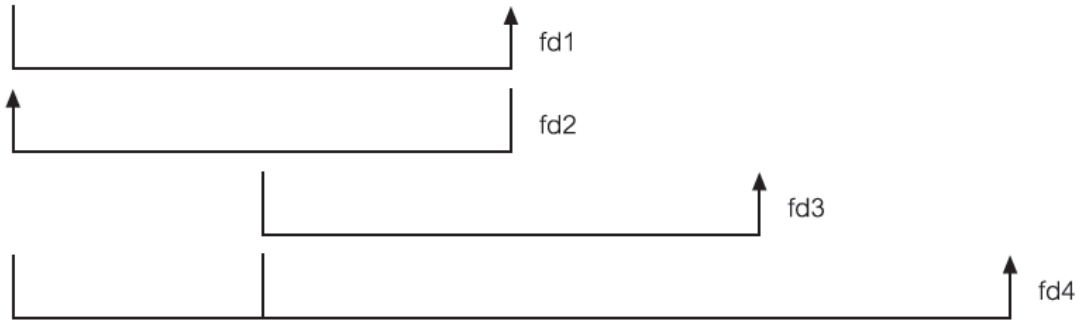
  - The closure of attribute set CD is the entire relation schema. Thus, attribute set CD is a super key for that relation.
  - No subset of attribute set CD contains all the attributes of the relation. Thus, attribute set CD is also a candidate key for that relation.
  - $C^+ = \{ C \}$
  - $D^+ = \{ D \}$

  As both A and CD are candidate keys, any one is chosen as primary key

Identifying the Keys Using Closure

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



- A → C (fd1)
- C → A (fd2)
- B → D (fd3)
- A, B → E (fd4)

Identifying the Keys Using Closure

- $A^+ = \{ A \}$
  $= \{ A , C \}$ (Using $A \rightarrow C$)
- $C^+ = \{ C \}$
  $= \{ A , C \}$ (Using $C \rightarrow A$)
- $B^+ = \{ B \}$
  $= \{ B , D \}$ (Using $B \rightarrow D$)
- $AB^+ = \{ A, B \}$
  $= \{ A, B , E \}$ (Using $AB \rightarrow E$)
  $= \{ A, B , C , E \}$ (Using $A \rightarrow C$)
  $= \{ A , B , C , D , E \}$ (Using $B \rightarrow D$)
  Thus,
  $AB^+ = \{ A , B , C , D , E \}$
  - The closure of attribute set AB is the entire relation schema. Thus, attribute set AB is a super key for that relation.
  - No subset of attribute set AB contains all the attributes of the relation. Thus, attribute set AB is also a candidate key for that relation.
  - As AB is a single candidate key of relation R, it becomes primary key.

Identifying the primary key for the Sample relation

- The determinants in the Sample relation are A, B, C, and (A, B). However, the only determinant that functionally determines all the other attributes of the relation is (A,B).
- A functionally determines C, B functionally determines D, and (A, B) functionally determines E.
- In other words, the attributes that make up the determinant (A, B) can determine all the other attributes in the relation either separately as A or B or together as (A, B).
- An essential characteristic for a candidate key of a relation is that the attributes of a determinant either individually or working together must be able to functionally determine *all the other attributes in the relation.*
- As there are no other candidate keys for the Sample relation (A, B) is identified as the primary key for this relation.

Identifying the primary key for the StaffBranch relation

- Five functional dependencies for the StaffBranch relation are:

  staffNo →sName, position, salary, branchNo, bAddress

  branchNo →bAddress

  bAddress →branchNo

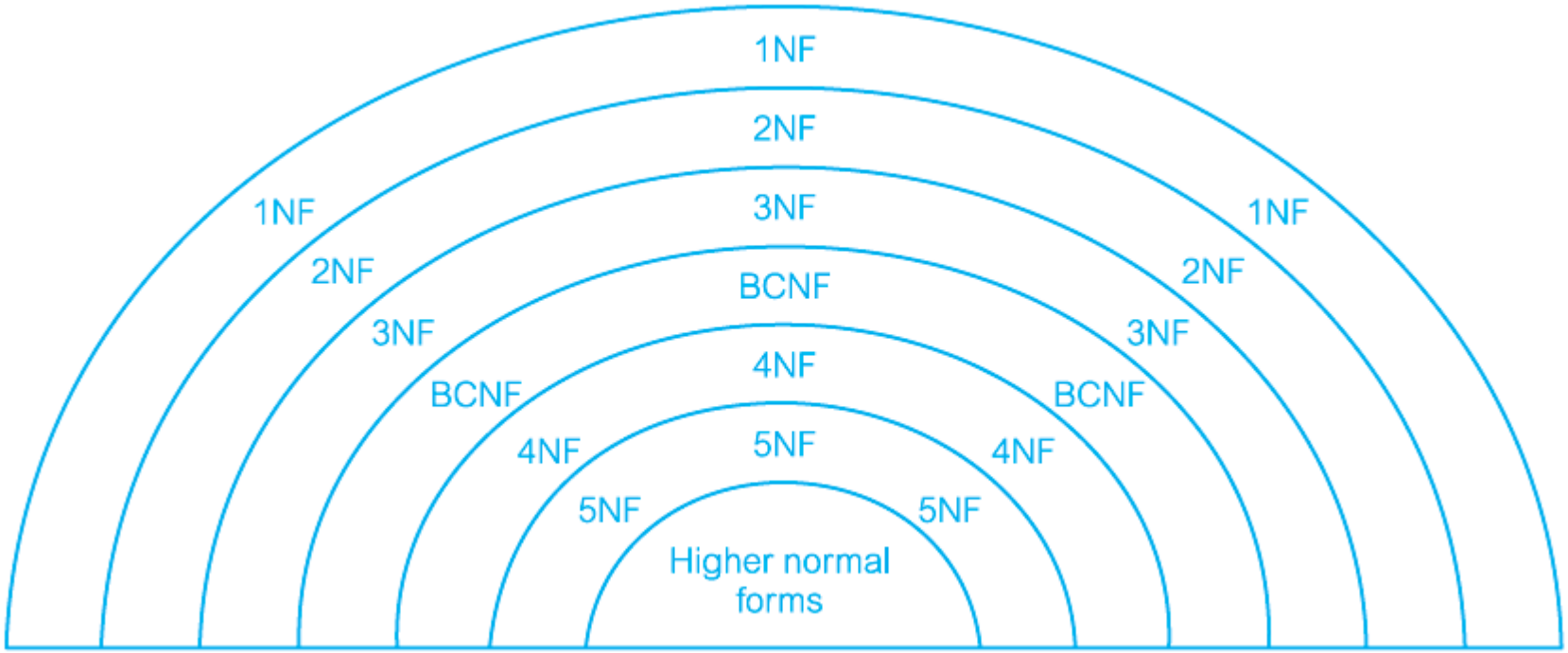
  branchNo, position →salary

  bAddress, position →salary

- With staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position) as determinants.
- staffNo attribute uniquely identifies each tuple in this relation.
- All attributes that are not part of the primary key (non-primary-key attributes) should be functionally dependent on the key.
- The only candidate key of the StaffBranch relation, and therefore the primary key, is staffNo, as *all other attributes of the relation are functionally dependent on staffNo.*

The Process of Normalization

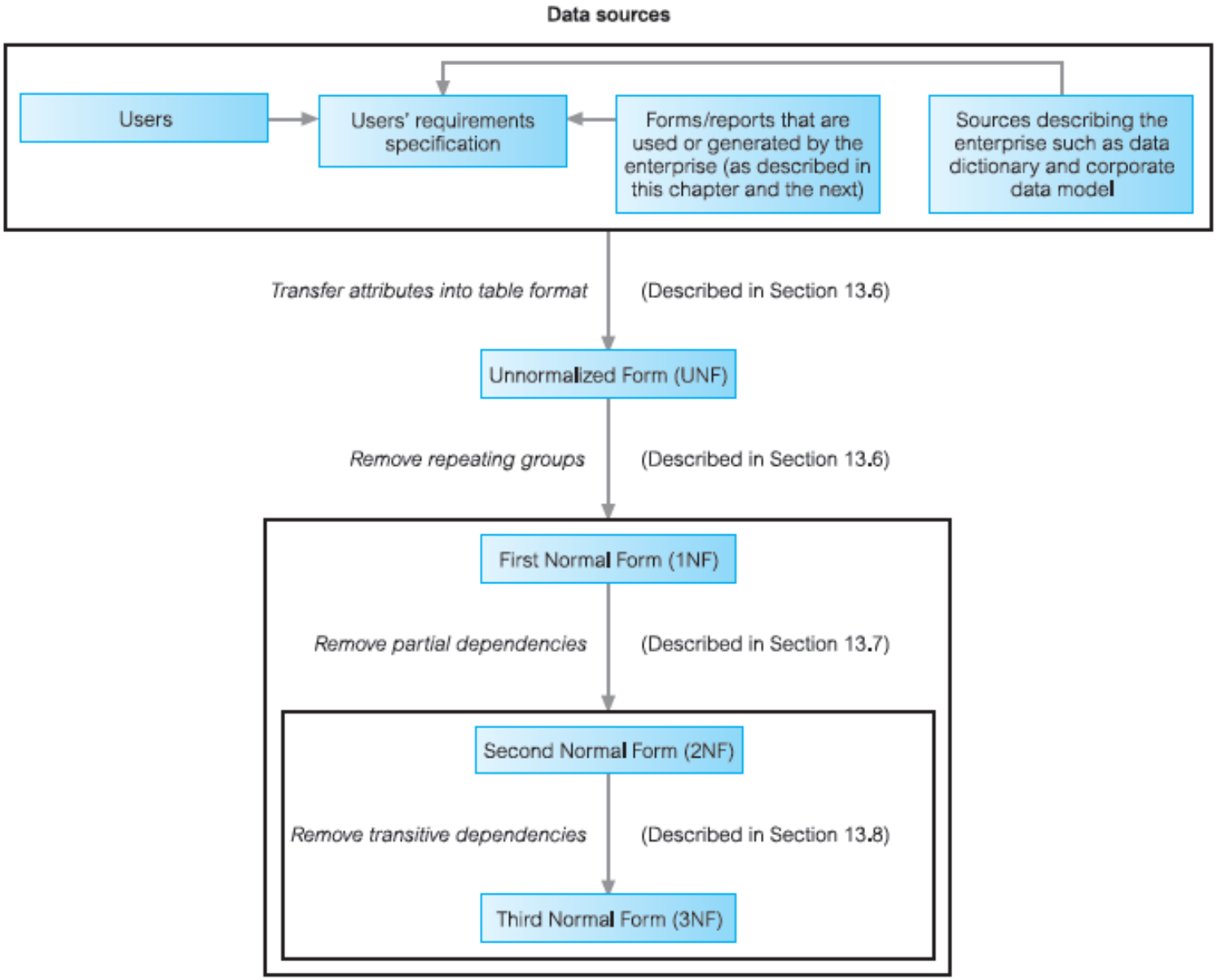
- **Normalization** is a formal technique for analyzing relations based on their primary key (or candidate keys) and functional dependencies.
- Illustration of the relationship between the normal forms:



The Process of Normalization

- Three normal forms were initially proposed called First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).
- Subsequently, R. Boyce and E.F. Codd introduced a stronger definition of third normal form called Boyce–Codd Normal Form (BCNF) (Codd, 1974).
- With the exception of 1NF, all these normal forms are based on functional dependencies among the attributes of a relation..
- Higher normal forms that go beyond BCNF were introduced later such as Fourth Normal Form (4NF) and Fifth Normal Form (5NF) (Fagin, 1977, 1979).
- However, these later normal forms deal with situations that are very rare.

The Process of Normalization



First Normal Form (1NF)

- **Unnormalized Form (UNF):** A table that contains one or more repeating groups.
- A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table.

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

ClientRental unnormalized table with clientNo as a key attribute.

The structure of the repeating group is:
Repeating Group = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

First Normal Form (1NF)

- To transform the unnormalized table to First Normal Form we identify and remove repeating groups within the table.
- There are two common approaches to removing repeating groups from unnormalized tables:

(1) *By entering appropriate data in the empty columns of rows containing the repeating data.* Commonly referred to as 'flattening' the table.

The resulting first normal form ClientRental relation is shown in Figure:

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

First Normal Form (1NF): Example

(2) With the second approach, we remove the repeating group by placing the repeating data along with a copy of the original key attribute (clientNo) in a separate relation

- The format of the resulting 1NF relations are as follows:

Client (clientNo, cName)

PropertyRentalOwner (clientNo, propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

First Normal Form (1NF)

- (1) *By entering appropriate data in the empty columns of rows containing the repeating data.* Commonly referred to as 'flattening' the table.
- (2) *By placing the repeating data, along with a copy of the original key attribute(s), in a separate relation.*
- **1NF:** A relation in which the intersection of each row and column contains one and only one value (atomic value).
  - A relation will be 1NF if it contains an atomic value.
  - It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
  - First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- **2NF:** In the 2NF, relational must be in 1NF.
- A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key.
- Second Normal Form (2NF) is based on the concept of full functional dependency.
- A relation with a single-attribute primary key is automatically in at least 2NF.
- Second Normal Form applies to relations with a primary key composed of two or more attributes.
- A relation that is not in 2NF may suffer from the update anomalies.

Fd1: clientNo → cName (**Primary key**)

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Second Normal Form (2NF)

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

- The PropertyRentalOwner relation has the following functional dependencies:

  Fd1: clientNo, propertyNo → rentStart, rentFinish (**Primary key**)

  Fd2: propertyNo → pAddress, rent, ownerNo, oName (**Partial dependency)**

  Fd3: ownerNo → oName (**Transitive dependency**)

  Fd4: clientNo, rentStart → propertyNo, pAddress, rentFinish, rent, ownerNo, oName (**Candidate key**)

  Fd5: propertyNo, rentStart → clientNo, rentFinish (**Candidate key)**

Second Normal Form (2NF): Example

- In fd1, the property rented attributes (rentStart and rentFinish) are fully dependent on the whole primary key; that is the clientNo and propertyNo attributes.
- In fd2, the property attributes (pAddress, rent, ownerNo, oName) are partially dependent on the primary key, that is, on only the propertyNo attribute.
- The identification of partial dependencies within the PropertyRentalOwner relation indicates that the relation is not in 2NF.
- To transform the PropertyRentalOwner relation into 2NF requires the creation of new relations so that every nonprimary- key attribute is fully functionally dependent on the primary key of the relation.

Second Normal Form (2NF): Example

- The normalization of 1NF relations to 2NF involves the removal of partial dependencies.
- If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them in a new relation along with a copy of their determinant.
- PropertyRentalOwner (clientNo, propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)
- **Decompose into:**

  Rental (clientNo, propertyNo, rentStart, rentFinish)

  PropertyOwner (propertyNo, pAddress, rent, ownerNo, oName)

Second Normal Form (2NF): Example

- This results in the creation of two new relations called Rental, and PropertyOwner.

  Rental (clientNo, propertyNo, rentStart, rentFinish)

  PropertyOwner (propertyNo, pAddress, rent, ownerNo, oName)

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

: clientNo → cName (**Primary key**)

Second Normal Form (2NF): Example

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

Fd1: clientNo, propertyNo → rentStart, rentFinish (**Primary key**)

Fd2: clientNo, rentStart → propertyNo, rentFinish (**Candidate key**)

Fd3: propertyNo, rentStart → clientNo, rentFinish (**Candidate key**)

Second Normal Form (2NF): Example

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Fd3: propertyNo → pAddress, rent, ownerNo, oName (**Primary key)**

Fd4: ownerNo → oName (**Transitive dependency**)

Example 2 : 2NF

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

2NF form

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- Although 2NF relations have less redundancy than those in 1NF, they may still suffer from update anomalies.
- **3NF:** A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key.
- The normalization of 2NF relations to 3NF involves the removal of transitive dependencies.
- If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity

3NF

- A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency X → Y.
  - X is a super key.
  - Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:3NF

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Transitive dependency in relation

- **Super key in the table above:**
- {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on
- **Candidate key:** {EMP_ID}
- **Non-prime attributes:** In the given table, all attributes except EMP_ID are non-prime.
- Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.
- That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

3NF form

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Third Normal Form (3NF): Example

- All the non-primary-key attributes within the Client and Rental relations are functionally dependent on only their primary keys. The Client and Rental relations have no transitive dependencies and are therefore already in 3NF.
- All the non-primary-key attributes within the PropertyOwner relation are functionally dependent on the primary key, with the exception of oName, which is transitively dependent on ownerNo (represented as fd4).
- To transform the PropertyOwner relation into 3NF we must first remove this transitive dependency by creating two new relations called PropertyForRent and Owner:
  - PropertyForRent (propertyNo, pAddress, rent, ownerNo)
  - Owner (ownerNo, oName)

Third Normal Form (3NF): Example

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Normalization

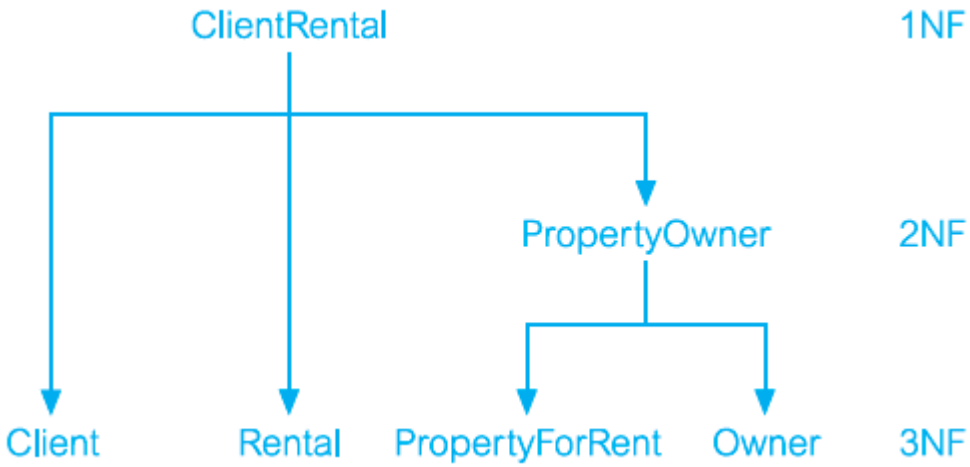
- The ClientRental relation has been transformed by the process of normalization into four relations in 3NF.
- The resulting 3NF relations have the form:

  Client (clientNo, cName)

  Rental (clientNo, propertyNo, rentStart, rentFinish)

  PropertyForRent (propertyNo, pAddress, rent, ownerNo)

  Owner (ownerNo, oName)



Normalization

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

A summary of the 3NF relations derived from the ClientRental relation.

Minimal Sets of Functional Dependencies

- A set of functional dependencies Y is covered by a set of functional dependencies X, if every functional dependency in Y is also in X^+; that is, every dependency in Y can be inferred from X.
- A set of functional dependencies X is minimal if it satisfies the following conditions:
  - Every dependency in X has a single attribute on its right-hand side.
  - We cannot replace any dependency $A \rightarrow B$ in X with dependency $C \rightarrow B$, where C is a proper subset of A, and still have a set of dependencies that is equivalent to X.
  - We cannot remove any dependency from X and still have a set of dependencies that is equivalent to X.
- A minimal set of dependencies should be in a standard form with no redundancies.
- A minimal cover of a set of functional dependencies X is a minimal set of dependencies X_{min} that is equivalent to X.

Minimal Sets of Functional Dependencies: Example

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

The functional dependencies based on our understanding of the attributes in the relation as:

staffNo →sName, position, salary, branchNo, bAddress
branchNo →bAddress
bAddress →branchNo
branchNo, position →salary
bAddress, position →salary

Minimal Sets of Functional Dependencies: Example

- We apply these three conditions on the set of functional dependencies for the StaffBranch to produce the following functional dependencies:
  - staffNo → sName
  - staffNo → position
  - staffNo → salary
  - staffNo → branchNo
  - staffNo → bAddress
  - branchNo → bAddress
  - bAddress → branchNo
  - branchNo, position → salary
  - bAddress, position → salary

Boyce–Codd Normal Form (BCNF)

- 2NF and 3NF disallow partial and transitive dependencies on the *primary key* of a relation, respectively.
- Relations that have these types of dependencies may suffer from the update anomalies.
- However, the definition of 2NF and 3NF do not consider whether such dependencies remain on other candidate keys of a relation.
- This weakness in 3NF, resulted in the presentation of a stronger normal form called Boyce–Codd Normal Form.
- Boyce–Codd Normal Form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation.
- A relation is in BCNF, if and only if, every determinant is a candidate key.

Boyce–Codd Normal Form (BCNF)

- A determinant is an attribute, or a group of attributes, on which some other attribute is fully functionally dependent.
- To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.
- Boyce–Codd Normal Form is a stronger form of 3NF, such that every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.
- The functional dependencies for the Client, Rental, and PropertyOwner relations are as follows:

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

fd1: clientNo → cName (**Primary key**)

BCNF

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

EXAMPLE

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

BCNF conversion

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
Candidate keys:
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Boyce–Codd Normal Form (BCNF)

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

Fd1: clientNo, propertyNo → rentStart, rentFinish (**Primary key**)

Fd2: clientNo, rentStart → propertyNo, rentFinish (**Candidate key**)

Fd3: propertyNo, rentStart → clientNo, rentFinish (**Candidate key**)

Boyce–Codd Normal Form (BCNF):Example

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

- PropertyForRent:

  Fd3: propertyNo → pAddress, rent, ownerNo, oName (**Primary key)**

- Owner:

  Fd4: ownerNo → oName (**Primary key**)

- The Client, PropertyForRent, and Owner relations are all in BCNF, as each relation only has a single determinant.
- As the three determinants of the Rental relation are also candidate keys, the Rental relation is also already in BCNF.

Fourth Normal Form (4NF): Multi-Valued Dependency

- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:
  - It should be in the **Boyce-Codd Normal Form**.
  - And, the table should not have any **Multi-valued Dependency**.
- A table is said to have multi-valued dependency, if the following conditions are true,
  1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
  2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
  3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, and A and C, then B and C should be independent of each other.

  A —>> B

  A —>> C
- If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Fourth Normal Form (4NF): Multi-Valued Dependency

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

- The given STUDENT table is in 3NF: In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.
- The COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

Fourth Normal Form (4NF): Multi-Valued Dependency

- So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Nath
34	Chemistry
74	Elology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth Normal Form (5NF): Lossless-Join Dependency

- **Joint dependency –** Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R then we can say that a join dependency (JD) exists, where R1 and R2 are the decomposition R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- Let R is a relation schema R1, R2, R3……..Rn be the decomposition of R. r(R) is said to satisfy join dependency if and only if

$$\bowtie_{i=1}^{n} \Pi_{R_i}(r) = r.$$

Fifth Normal Form (5NF): Lossless-Join Dependency

- A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R.
- A relation decomposed into two relations must have loss-less join property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.
- A relation R is in 5NF if and only if it satisfies following conditions:
  - R should be already in 4NF.
  - not contains any join dependency and joining should be lossless.
  - 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- Fifth Normal Form (5NF) also called *Project-Join Normal Form (PJNF).*

Fifth Normal Form (5NF)

- Consider example : “if a company makes a product and an agent is an agent for that company, then he always sells that product for the company”. The ACP table is shown as:

Table – ACP

AGENT	COMPANY	PRODUCT
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

Fifth Normal Form (5NF)

- The relation ACP is again decompose into 3 relations.

AGENT	COMPANY
A1	PQR
A1	XYZ
A2	PQR

Table – R2

AGENT	PRODUCT
A1	Nut
A1	Bolt
A2	Nut

Table – R3

COMPANY	PRODUCT
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

Fifth Normal Form (5NF)

- Result of Natural Join of R1 and R3 over ‘Company’ and then Natural Join of R3 and R2 over ‘Agent’and ‘Product’ will be table **ACP**.
- Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation is in 5NF as it does not violate the property of lossless join.

Example2: 5NF

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

Decomposition : 5NF

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

Decomposition : 5NF

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Lossless Join and Dependency Preserving Decomposition

- Decomposition of a relation is done when a relation in relational model is not in appropriate normal form.
- There are two important properties associated with decomposition of a larger relation into smaller relations:

1. The **lossless-join** property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.
2. The **dependency preservation** property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations.

Lossless Join Decomposition

- If we decompose a relation R into relations R1 and R2,
  - Decomposition is lossy if $R1 \bowtie R2 \supset R$
  - Decomposition is lossless if $R1 \bowtie R2 = R$
- To check for lossless join decomposition using FD set, following conditions must hold:
  1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

     $Att(R1) \cup Att(R2) = Att(R)$
  2. Intersection of Attributes of R1 and R2 must not be NULL.

     $Att(R1) \cap Att(R2) \neq \Phi$
  3. Common attribute must be a key for at least one relation (R1 or R2)

     $Att(R1) \cap Att(R2) \rightarrow Att(R1)$ or

     $Att(R1) \cap Att(R2) \rightarrow Att(R2)$

Lossless Join Decomposition

- For Example, A relation R (A, B, C, D) with FD set{A → BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:
  1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
  2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
  3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A → BC is given.

Dependency Preserving Decomposition

- If we decompose a relation R into relations R1 and R2, all dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.
- For Example, A relation R (A, B, C, D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).

Lossless Join and Dependency Preserving Decomposition

- **Consider a schema R(A,B,C,D) and functional dependencies A → B and C → D. Then the decomposition of R into R1(AB) and R2(CD) is**

  A. dependency preserving and lossless join

  B. lossless join but not dependency preserving

  C. dependency preserving but not lossless join

  D. not dependency preserving and not lossless join

- For lossless join decomposition, three conditions must hold true:
  - Att(R1) U Att(R2) = ABCD = Att(R)
  - Att(R1) ∩ Att(R2) = Φ, which violates the condition of lossless join decomposition.

  Hence the decomposition is not lossless.

- For dependency preserving decomposition,
  - A → B can be ensured in R1(AB) and
  - C → D can be ensured in R2(CD).

  Hence it is dependency preserving decomposition.

  So, the correct option is C.

Lossless Join and Dependency Preserving Decomposition

- Let a relation R(A,B,C,D) and set a FDs F = { A → B , A → C , C → D} are given. A relation R is decomposed into - R_1 = (A, B, C) with FDs F_1 = {A → B, A → C}, and R_2 = (C, D) with FDs F_2 = {C → D}
- For lossless join decomposition, three conditions must hold true:
  - Att(R1) U Att(R2) = ABCD = Att(R)
  - Att(R1) ∩ Att(R2) = C,
  - C is primary key of R2 {C → D}

  Hence the decomposition is **lossless join**
- For dependency preserving decomposition,
  - A → B and A → C can be ensured in R1(ABC) and
  - C → D can be ensured in R2(CD).

  Hence it is **dependency preserving** decomposition.

End of Unit 2