Relational Database Design

COMPILED BY: RITURAJ JAIN

The Banking Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*,
  customer_city)
- *account* = (*account_number*, *balance*)
- *depositor* = (*customer_id*, *account_number*)
- *loan* = (*loan_number*, *amount*)
- *borrower* = (*customer_id*, *loan_number*)

Pitfalls in Relational Database Design

- Relational database design requires that we find a “good” collection of relation schemas. A bad design may lead to
  - **Repetition of Information.**
  - **Inability to represent certain information.**
- Design Goals:
  - **Avoid redundant data**
  - **Ensure that relationships among attributes are represented**
  - **Facilitate the checking of updates for violation of database integrity constraints.**

Example

- Consider the relation schema for loan:

  ***Lending-schema* = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)**

B_name	B_city	assets	Cust_name	L_no	Amount
Coll_Road	**Nadiad**	**9000000**	Ajay	L 21	21000
Coll_Road	**Nadiad**	**9000000**	Suresh	L 23	26500
C.G. Road	Ahmedabad	2574000	Suresh	L 43	2300
Raj Marg	**Surat**	**2563000**	Ajay	L 100	74500
Raj Marg	**Surat**	**2563000**	Rakshita	L 45	100000

- Redundancy:
  - Data for *branch-name, branch-city,* assets are repeated for each loan that a branch makes
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of *assets* value
- Null values
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle.

Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in “good” form.
- In the case that a relation R is not in “good” form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is **a lossless-join decomposition**

Decomposition

- Decompose the relation schema *Lending-schema* into:

Branch-schema = (branch-name, branch-city,assets)

B_name	B_city	assets	Cust_name
Coll_Road	Nadiad	9000000	**Ajay**
Coll_Road	Nadiad	9000000	**Suresh**
C.G. Road	Ahmedabad	2574000	**Suresh**
Raj Marg	Surat	2563000	**Ajay**
Raj Marg	Surat	2563000	**Rakshita**

Loan-info-schema = (customer-name, loan-number, branch-name, amount)

Cust_name	L_no	Amount
Ajay	L 21	21000
Suresh	L 23	26500
Suresh	L 43	2300
Ajay	L 100	74500
Rakshita	L 45	100000

Decomposition

- Sometimes it is required to reconstruct loan relation from the ***Branch-schema*** *and* ***Loan-info-schema:*** *so we can do this by*

Branch-schema $\bowtie$ ***Loan-info-schema***

B_name	B_city	assets	Cust_name	L_no	Amount
Coll_Road	Nadiad	9000000	**Ajay**	L 21	21000
Coll_Road	Nadiad	9000000	**Ajay**	L 100	74500
Coll_Road	Nadiad	9000000	**Suresh**	L 23	26500
Coll_Road	Nadiad	9000000	**Suresh**	L 43	2300
C.G. Road	Ahmedabad	2574000	**Suresh**	L 23	26500
C.G. Road	Ahmedabad	2574000	**Suresh**	L 43	2300
Raj Marg	Surat	2563000	**Ajay**	L 21	21000
Raj Marg	Surat	2563000	**Ajay**	L 100	74500
Raj Marg	Surat	2563000	**Rakshita**	L 45	100000

- ***Which customer are borrowers of from which branch? (lost information)***

Decomposition

- **In the last example we are not able to identify which customers are borrower from which branch.**
- because of this loss of information
- This type of decomposition is called **lossy decomposition**.
- A decomposition that is not a lossy-join decomposition is called **lossless join decomposition.**
- **So lossy join decomposition is a bad database design.**

Decomposition

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2, R_3, …… R_n):

$$R = R_1 \cup R_2 \cup R_3 \ldots\ldots\ldots\ldots \cup R_n$$

- Lossless-join decomposition.

  For all possible relations *r* on schema *R*

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r) \bowtie \Pi_{R3}(r) \bowtie \ldots\ldots\ldots \bowtie \Pi_{Rn}(r)$$

What is Normalization?

- Database designed based on the E-R model may have some amount of
  - Inconsistency
  - Uncertainty
  - Redundancy

To eliminate these draw backs some **refinement** has to be done on the database.

- **Refinement** process is called **Normalization**
- Defined as a step-by-step process of decomposing a complex relation into a simple and stable data structure.
- The formal process that can be followed to achieve a good database design
- Also used to check that an existing design is of good quality
- The different stages of normalization are known as “normal forms”
- To accomplish normalization we need to understand the concept of Functional Dependencies.

Source: Infosys Campus Connect Study Material

Need for Normalization

Student_Course_Result Table

Student_Details			Course_Details				Result_Details		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

- **Data Duplication**
- **Delete Anomaly**
- **Insert Anomaly**
- **Update Anomaly**

Source: Infosys Campus Connect Study Material

Need for Normalization

- **Duplication of Data** – The same data is listed in multiple lines of the database
- **Insert Anomaly** – A record about an entity cannot be inserted into the table without first inserting information about another entity – Cannot enter a student details without a course details
- **Delete Anomaly** – A record cannot be deleted without deleting a record about a related entity. Cannot delete a course details without deleting all of the students' information.
- **Update Anomaly** – Cannot update information without changing information in many places. To update student information, it must be updated for each course the student has placed

Desirable Properties of Decomposition

1. We'll take another look at the schema

Lending-schema = (B_name, assets, B_city, L_no, cust_name,amount)

which we saw was a bad design.

2. The set of functional dependencies we required to hold on this schema was:

B_name → assets B_city

L_no → amount B_name

3. If we decompose it into

Branch-schema = (B_name, assets, B_city)

Loan-info-schema = (B_name, L_no, amount)

Borrow-schema = (cust_name, L_no)

we claim this decomposition has several desirable properties.

Desirable Properties of Decomposition

a) Lossless Decomposition

b) Dependency Preservation

c) Repetition of information

Desirable Properties of Decomposition

a) Lossless Decomposition

How can we decide whether a decomposition is lossless?

- Let **R** be a relation schema.
- Let **F** be a set of functional dependencies on **R.**
- Let **R1** and **R2** form a decomposition of **R**.
- **The decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies are in F^+:**

(a) R1 ∩ R2 → R1

(b) R1 ∩ R2 → R2

Example

a) Lossless Decomposition

- *R = (A, B, C)*

  F = {A → B, B → C)

  - Can be decomposed in two different ways

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

  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$

    (a) R1 ∩ R2 → R1
    (b) R1 ∩ R2 → R2

  - Dependency preserving

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

  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

    (a) R1 ∩ R2 → R1
    (b) R1 ∩ R2 → R2

  - **Not dependency preserving**

    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Desirable Properties of Decomposition

a) Lossless Decomposition

Example:

- First we decompose ***Lending-schema*** *into* ***Branch-schema and Loan-info-schema***

Lending-schema = (B_name, assets, B_city, L_no, cust_name,amount)

Branch-schema = (B_name, B_city, assets)

Loan-info-schema = (B_name, cust_name, L_no, amount)

- ***B_name → assets B_city***, the augmentation rule for functional dependencies implies that ***B_name*** → ***B_name assets B_city***

- Since **Branch-schema ∩ Loan-info-schema = B_name**, **our decomposition is lossless join.**

Desirable Properties of Decomposition

a) Lossless Decomposition

Example Continue:

- Next we decompose ***Loan-info-schema*** into *Loan-schema and Borrow-schema*

Loan-info-schema = (B_name, cust_name, L_no, amount)

Loan-schema = (B_name, L_no, amount)

Borrow-schema = (cust_name, L_no)

- As ***L_no*** is the common attribute, and

  L_no → ***L_no amount B_name***

- **This is also a lossless-join decomposition.**

Desirable Properties of Decomposition

b) Dependency Preservation

- Check that updates to the database do not result in illegal relations
- Better to check updates without having to compute natural joins.
- To know whether joins must be computed, we need to determine what functional dependencies may be tested by checking each relation individually.
- Let **F** be a set of functional dependencies on schema **R**. Let $\{R_1, R_2, \ldots, R_n\}$ be a decomposition of **R**.
- The **restriction** of **F** to R_i is the set of all functional dependencies**(denoted as** F_i**)** in F^+ that include only attributes of R_i.

Desirable Properties of Decomposition

b) Dependency Preservation

- **$F_1, F_2, \ldots, F_n$** is the set of dependencies of decomposed relations.
- **$F' = F_1 \cup F_2 \cup \ldots \cup F_n$**
- When a relational schema **R** defined by functional dependency **F** is decomposed into **$\{R_1, R_2, \ldots, R_n\}$**, each functional dependency should be testable by at least one of R_i.
- Formally, let **F^+** be the closure **F** and let **F'^+** be the closure of dependencies covered by R_i.
- **$F^+ == F'^+$ for dependency preservation.**

Testing for Dependency Preservation

b) Dependency Preservation

compute F^+

for each schema R_i in **D** do

begin

F_i := the restriction of F^+ to R_i ;

end

$F' := \varphi$

for each restriction F_i do

begin

$F' = F' \cup F_i$

end

compute F'^+ ;

if $(F'^+ = F^+)$ then return (true)

else return (false);

Testing for Dependency Preservation

b) Dependency Preservation

Lending-schema = (B_name, assets, B_city, L_no, cust_name,amount)

Decomposed into these schemas:

Branch-schema = (B_name, assets, B_city)

Loan-info-schema = (B_name, L_no, amount)

Borrow-schema = (cust_name, L_no)

Decomposition of Lending-schema is dependency preserving.

B_name → assets B_city

L_no → amount B_name

Desirable Properties of Decomposition

c) Repetition of Information

- Our decomposition does not suffer from the repetition of information problem.
- Branch and loan data are separated into distinct relations.
- Thus we do not have to repeat branch data for each loan.
- If a single loan is made to several customers, we do not have to repeat the loan amount for each customer.
- This lack of redundancy is obviously desirable.
- **We will see how this may be achieved through the use of normal forms.**

Functional dependency

- In a given relation R, X and Y are attributes. Attribute Y is **functionally dependent** on attribute X if each value of X determines **EXACTLY ONE** value of Y, which is represented as X -> Y (X can be composite in nature).

- We say here "x determines y" *or* "y is functionally dependent on x"

  $X \rightarrow Y$ does not imply $Y \rightarrow X$

- If the value of an attribute "Marks" is known then the value of an attribute "Grade" is determined since Marks$\rightarrow$ Grade

- Types of functional dependencies:

  – Full Functional dependency
  – Partial Functional dependency
  – Transitive dependency

Source: Infosys Campus Connect Study Material

Functional dependency

Consider the following Relation

REPORT (<u>STUDENT#,COURSE#</u>, CourseName, IName, Room#, Marks, Grade)

- **STUDENT#** - Student Number
- **COURSE#** - Course Number
- **CourseName** - Course Name
- **IName** - Name of the Instructor who delivered the course
- **Room#** - Room number which is assigned to respective Instructor
- **Marks** - Scored in Course COURSE# by Student STUDENT#
- **Grade** - obtained by Student STUDENT# in Course COURSE#

Source: Infosys Campus Connect Study Material

Functional dependency

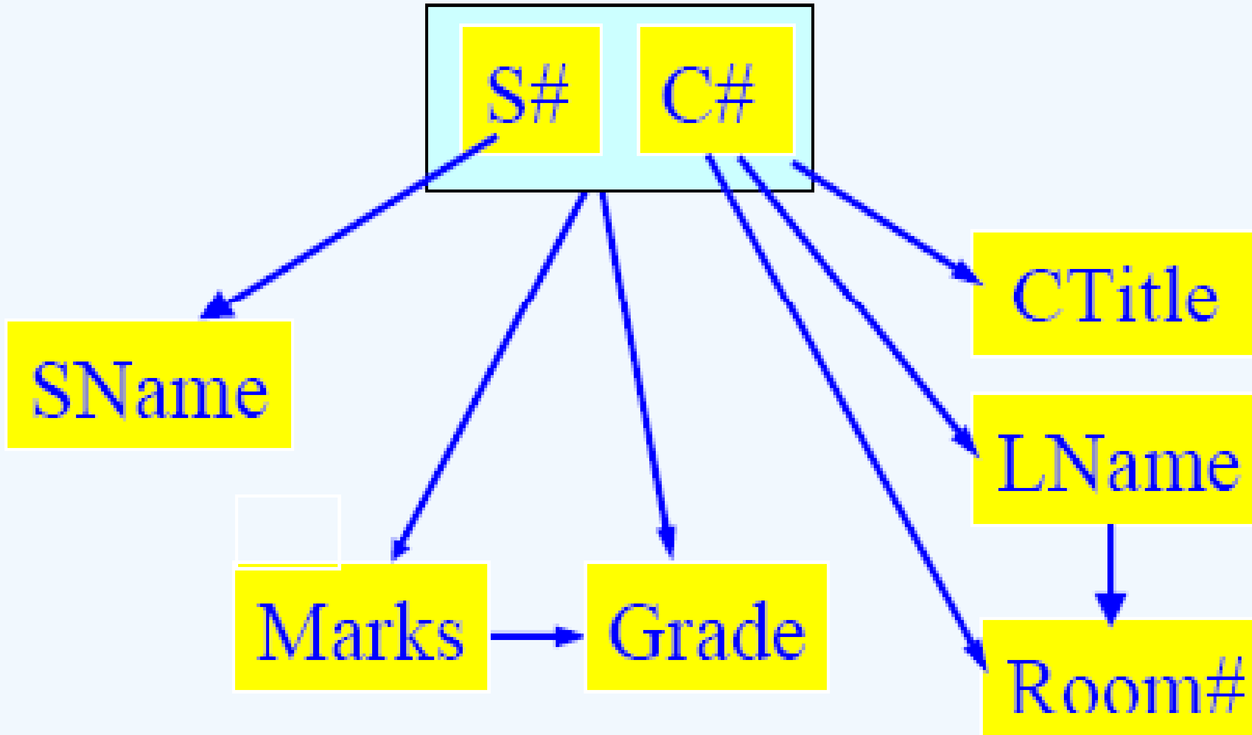
- STUDENT# COURSE# → Marks
- COURSE# → CourseName,
- COURSE# → IName (Assuming one course is taught by one and only one Instructor)
- IName → Room# (Assuming each Instructor has his/her own and non-shared room)
- Marks → Grade

Source: Infosys Campus Connect Study Material

Dependency Diagram

Report(S#,C#,SName,CTitle,LName,Room#,Marks,Grade)

- S# → SName
- C# → CTitle,
- C# → LName
- LName → Room#
- C# → Room#
- S# C# → Marks
- Marks → Grade
- S# C# → Grade



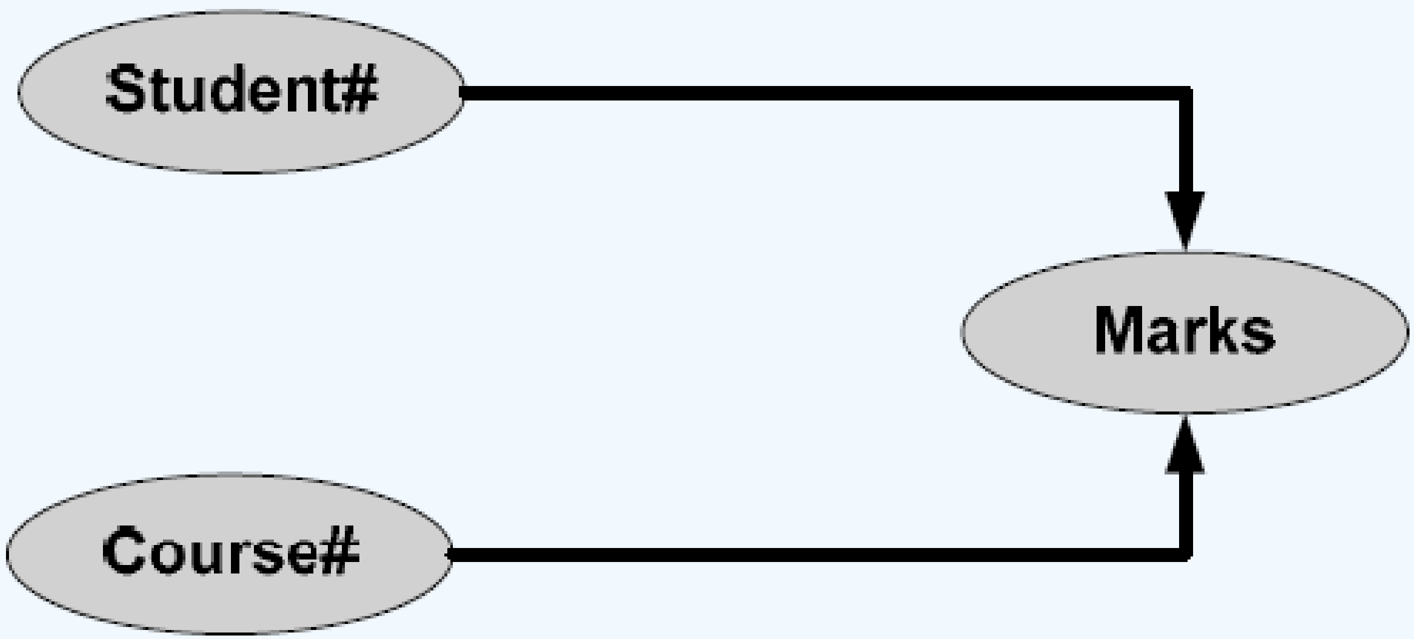
Assumptions:

- Each course has only one lecturer and each lecturer has a room.
- Grade is determined from Marks.

Source: Infosys Campus Connect Study Material

Full Dependency

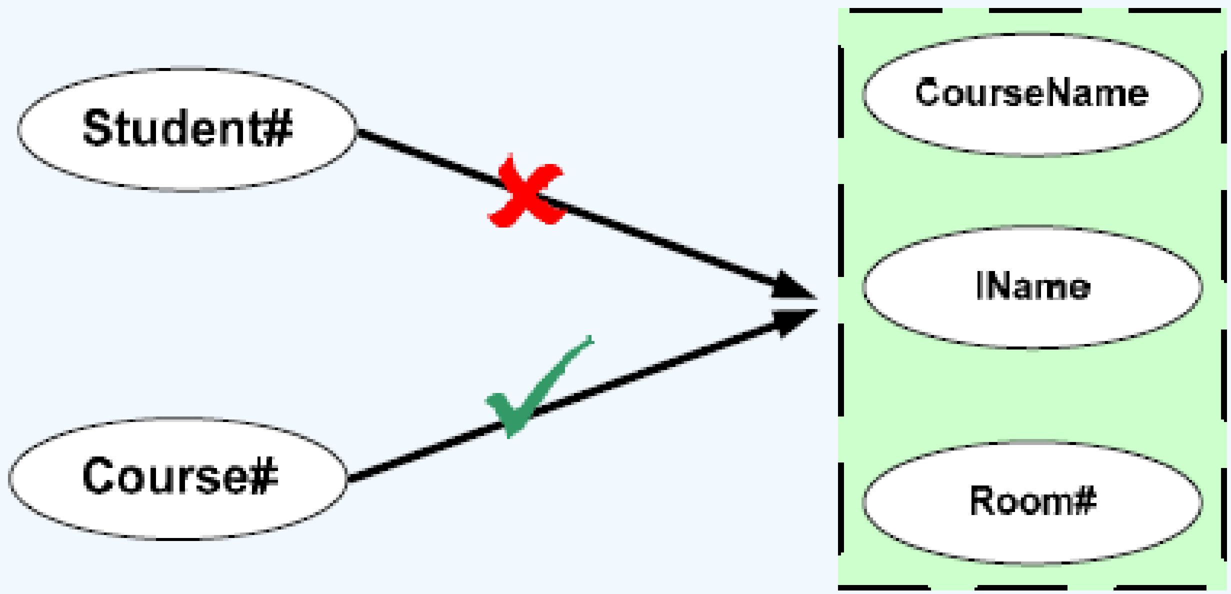
X and Y are attributes.
X Functionally determines Y
Note: Subset of X should not functionally determine Y



Source: Infosys Campus Connect Study Material

Partial Dependency

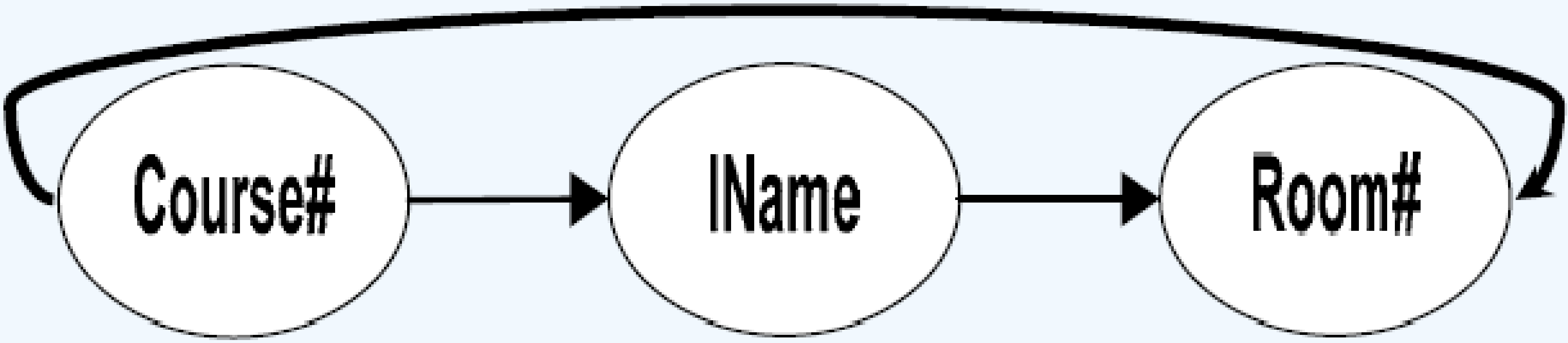
X and Y are attributes.
Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.



Source: Infosys Campus Connect Study Material

Transitive Dependency

X Y and Z are three attributes.
X -> Y
Y-> Z
=> X -> Z



Source: Infosys Campus Connect Study Material

First Normal Form

- Domain is **atomic** if its elements are considered to be **indivisible units**
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data

First Normal Form (Cont'd)

- **A relation schema is in 1NF :**
  - if and only if all the attributes of the relation R are atomic in nature.
  - **Atomic:** the smallest level to which data may be broken down and remain meaningful

Source: Infosys Campus Connect Study Material

Example … Without Normalization

Student_Course_Result Table

Student_Details			Course_Details				Result_Details		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

Source: Infosys Campus Connect Study Material

Table in 1NF

Source: Infosys Campus Connect Study Material

Student_Course_Result Table

Student#	Student Name	Dateof Birth	Course#	CourseName	Pre Requisite	Duration InDays	DateOf Exam	Marks	Grade
101	Davis	04-Nov-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	82	A
102	Daniel	06-Nov-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	62	C
101	Davis	04-Nov-1986	H6	American History		4	22-Nov-2004	79	B
103	Sandra	02-Oct-1988	C3	Bio Chemistry	Basic Chemistry	11	16-Nov-2004	65	B
104	Evelyn	22-Feb-1986	B3	Botany		8	26-Nov-2004	77	B
102	Daniel	06-Nov-1986	P3	Nuclear Physics	Basic Physics	13	12-Nov-2004	68	B
105	Susan	31-Aug-1985	P3	Nuclear Physics	Basic Physics	13	12-Nov-2004	89	A
103	Sandra	02-Oct-1988	B4	Zoology		5	27-Nov-2004	54	D
105	Susan	31-Aug-1985	H6	American History		4	22-Nov-2004	87	A
104	Evelyn	22-Feb-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	65	B

First Normal Form Example

Course_Pref_Table			
Dept	Prof	Course Pref	
		Course	Course_dept
CE	Rajiv	101	CS
		102	CS
		103	EC
	Mahesh	101	CS
		102	CS
		103	EC
		104	EC
CL	Ruchika	101	CS
		103	EC
		106	EE
IT	Rajesh	103	EC
		104	EC
		106	EE
		102	CS
		105	EE

First Normal Form Example

Course_Pref_Table			
Dept	Prof	Course	Course_dept
CE	Rajiv	101	CS
CE	Rajiv	102	CS
CE	Rajiv	103	EC
CE	Mahesh	101	CS
CE	Mahesh	102	CS
CE	Mahesh	103	EC
CE	Mahesh	104	EC
CL	Ruchika	101	CS
CL	Ruchika	103	EC
CL	Ruchika	106	EE
IT	Rajesh	103	EC
IT	Rajesh	104	EC
IT	Rajesh	106	EE
IT	Rajesh	102	CS
IT	Rajesh	105	EE

Second normal form: 2NF

- ***A Relation is said to be in Second Normal Form if and only if :***
  - ***It is in the First normal form, and***
  - ***No partial dependency exists between non-key attributes and key attributes.***

- An attribute of a relation R that belongs to any key of R is said to be a prime attribute and that which doesn't is a **non-prime attribute**

To make a table 2NF compliant, we have to remove all the partial dependencies

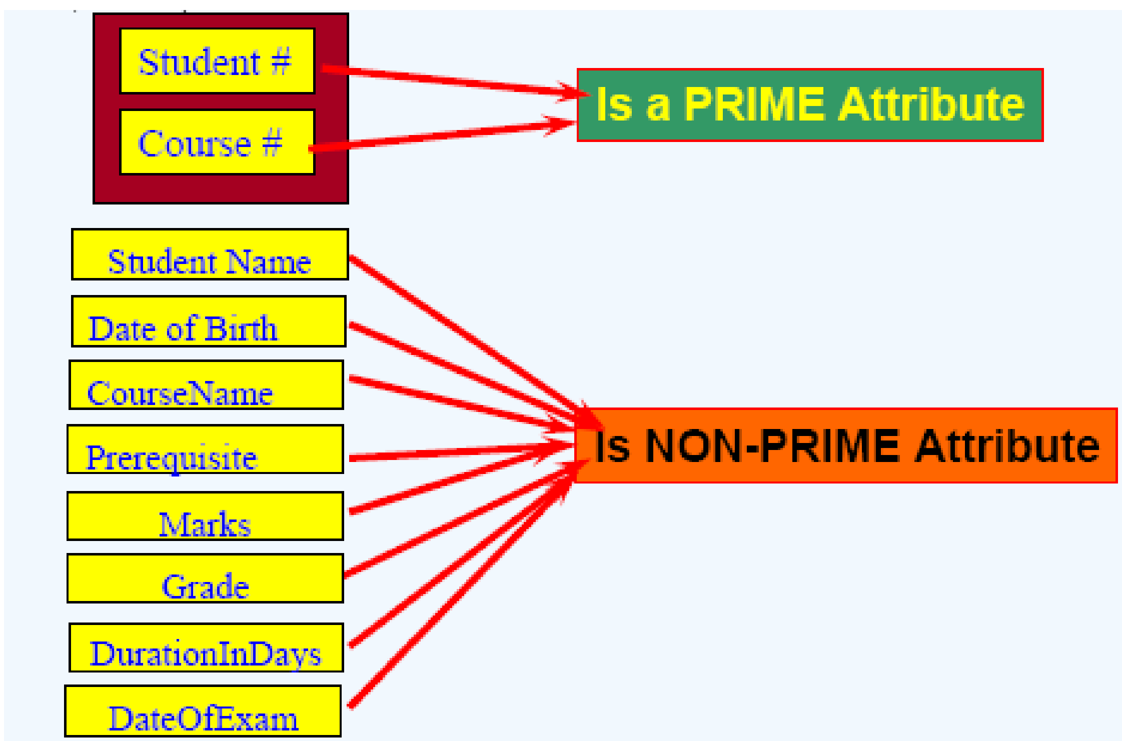
Note : - All partial dependencies are eliminated

Source: Infosys Campus Connect Study Material

Prime Vs Non-Prime Attributes

- An attribute of a relation R that belongs to any key of R is said to be a **prime** attribute and that which doesn't is a **non-prime attribute**

Report(S#, C#, StudentName, DateOfBirth, CourseName, PreRequisite, DurationInDays, DateOfExam, Marks, Grade)



Source: Infosys Campus Connect Study Material

Second normal form: 2NF

- STUDENT# is key attribute for Student,
- COURSE# is key attribute for Course
- STUDENT# COURSE# together form the composite key attributes for Results relationship.
- Other attributes like StudentName (Student Name), DateofBirth, CourseName, PreRequisite, DurationInDays, DateofExam, Marks and Grade are non-key attributes.

To make this table 2NF compliant, we have to remove all the partial dependencies.

Student #, Course# -> Marks, Grade

Student# -> StudentName, DOB,

Course# -> CourseName, Prerequiste, DurationInDays

Course# -> Date of Exam

Source: Infosys Campus Connect Study Material

Second normal form: 2NF

- S#,C# → Marks
- S#,C# → Grade

Fully Functionally dependent on composite Candidate key

- S# → StudentName
- S# → DOB

Partial Dependency

- C# → CourseName
- C# → Prerequisite
- C# → Duration

Partial Dependency

- C# → DateOfExam

Partial Dependency

Source: Infosys Campus Connect Study Material

Second normal form: Table in 2NF

STUDENT TABLE

Student#	StudentName	DateofBirth
101	Davis	04-Nov-1986
102	Daniel	06-Nov-1987
103	Sandra	02-Oct-1988
104	Evelyn	22-Feb-1986
105	Susan	31-Aug-1985
106	Mike	04-Feb-1987
107	Juliet	09-Nov-1986
108	Tom	07-Oct-1986
109	Catherine	06-Jun-1984

COURSE TABLE

Course#	Course Name	Pre Requisite	Duration InDays
M1	Basic Mathematics		11
M4	Applied Mathematics	M1	7
H6	American History		4
C1	Basic Chemistry		5
C3	Bio Chemistry	C1	11
B3	Botany		8
P1	Basic Physics		8
P3	Nuclear Physics	P1	13
B4	Zoology		5

Source: Infosys Campus Connect Study Material

Second normal form: Table in 2NF

Student#	Course#	Marks	Grade
101	M4	82	A
102	M4	62	C
101	H6	79	B
103	C3	65	B
104	B3	77	B
102	P3	68	B
105	P3	89	A
103	B4	54	D
105	H6	87	A
104	M4	65	B

Exam_Date Table

Course#	DateOfExam
M4	11-Nov-04
H6	22-Nov-04
C3	16-Nov-04
B3	26-Nov-04
P3	12-Nov-04
B4	27-Nov-04

Source: Infosys Campus Connect Study Material

Second normal form … Example

Example: The following relation is in First Normal Form, but not Second Normal Form:

Cust_Order_table

OrderNo	Customer	ContactPerson	Total
1	Acme Widgets	John Doe	$134.23
2	ABC Corporation	Fred Flintstone	$521.24
3	Acme Widgets	John Doe	$1042.42
4	Acme Widgets	John Doe	$928.53

OrderNo Customer → Total

Customer → ContactPerson

Second normal form … Example

Customer table

Customer	ContactPerson
Acme Widgets	John Doe
ABC Corporation	Fred Flintstone

Customer → ContactPerson

Order_table

OrderNo	Customer	Total
1	Acme Widgets	$134.23
2	ABC Corporation	$521.24
3	Acme Widgets	$1042.42
4	Acme Widgets	$928.53

OrderNo Customer → Total

Boyce-Codd Normal Form

A relation schema ***R*** is in **BCNF** with respect to a set ***F*** of functional dependencies if for all functional dependencies in F^+ of the form

$\alpha \rightarrow \beta$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- **$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)**
- **α is a superkey for *R***

Example schema *not* in BCNF:

***bor_loan* = (*customer_id, loan_number, amount*)**

because ***loan_number*** $\rightarrow$ ***amount*** holds on ***bor_loan*** but ***loan_number*** is not a superkey

Decomposing a Schema into BCNF

- Suppose we have a schema ***R*** and a **non-trivial dependency** $\alpha \rightarrow \beta$ causes a violation of **BCNF**.

  We decompose ***R*** into:

  - **(α U β)**
  - **(*R* - (β - α))**

- In our example,

  - α **= *loan_number***
  - β **= *amount***

  and ***bor_loan*** is replaced by

  - **(α U β) = (*loan_number, amount*)**
  - **(*R* - (β - α)) = (*customer_id, loan_number*)**

Decomposing a Schema into BCNF

- *Lending-schema = (B_name, assets, B_city, L_no, cust_name,amount)*

  B_name → assets B_city (not trivial and B_name is not a super key)

  L_no → amount B_name (not trivial and L_no is not a super key)

Candidate key for this Schema is { L_no, cust_name}. This Schema is not in BCNF form. So decompose this schema into below given two schemas

Branch-schema = (B_name, B_city, assets)

Loan-info-schema = (B_name, cust_name, L_no, amount)

- *B_name → assets B_city*, the augmentation rule for functional dependencies implies that *B_name → B_name assets B_city*

- **B_name is super key in Branch_schema.**

Decomposing a Schema into BCNF

Loan-info-schema = (B_name, cust_name, L_no, amount)

L_no → amount B_name (not trivial and L_no is not a super key)

- *This Schema is not in BCNF form. So decompose this schema into below given two schemas*

Loan-schema = (B_name, L_no, amount)

Borrow-schema = (cust_name, L_no)

- *Both of these two schemas are in BCNF.*
- *Decomposition of Lending-schema to all these three schema Branch-schema, Loan-schema and Borrow-schema having dependency preservation and lossless decomposition.*

BCNF and Dependency Loss…Example

- banker-schema = (branch-name, customer-name, banker-name)

  banker-name → branch-name

  branch-name customer-name → banker-name

- Banker-schema is not in BCNF -- Why?
- banker-name is not a super key. So decompose banker-schema…..

  banker-branch-schema = (banker-name, branch-name)

  customer-banker-schema = (customer-name, banker-name)

- New schema in BCNF but only one dependency is preserves

  banker-name → branch-name

- While other dependency is not preserve.

Testing for BCNF

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

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction of F (i.e. F_i)** to R_i (that is, all FDs in F^+ that contain only attributes from R_i)

Third Normal Form

- A relation schema *R* is in third normal form (3NF) if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:

  - **$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)**
  - **α is a superkey for *R***
  - **Each attribute *A* in ($\beta - \alpha$) is contained in a candidate key for *R.***

    (NOTE*:* each attribute may be in a different candidate key)

- **If a relation is in BCNF it is in 3NF** (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation.

Third Normal Form

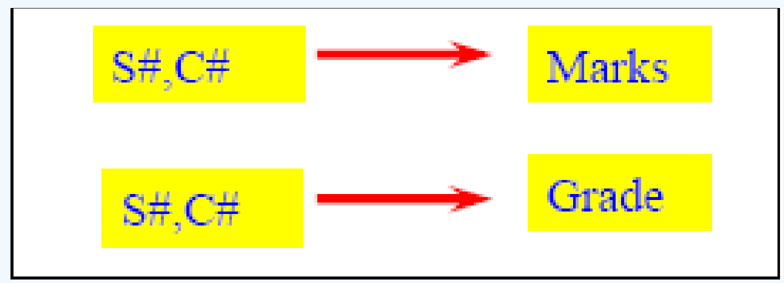
A relation R is said to be in the Third Normal Form (3NF) if and only if

— ***It is in 2NF and***

— ***No transitive dependency exists between non-key attributes and key attributes.***

- STUDENT# and COURSE# are the key attributes.
- All other attributes, except grade are non-partially, non-transitively

  dependent on key attributes.



- **Student#, Course# - >** Marks
- **Marks ->** Grade

Note : - All transitive dependencies are eliminated

Source: Infosys Campus Connect Study Material

Third Normal Form

Note that 3NF is concerned with transitive dependencies which do not involve candidate keys. A 3NF relation with more than one candidate key will clearly have transitive dependencies of the form:

primary_key → other_candidate_key → any_non-key_column

Third Normal Form

Student#	Course#	Marks	Grade
101	M4	82	A
102	M4	62	C
101	H6	79	B
103	C3	65	B
104	B3	77	B
102	P3	68	B
105	P3	89	A
103	B4	54	D
105	H6	87	A
104	M4	65	B

Source: Infosys Campus Connect Study Material

Third Normal Form

Student#	Course#	Marks
101	M4	82
102	M4	62
101	H6	79
103	C3	65
104	B3	77
102	P3	68
105	P3	89
103	B4	54
105	H6	87
104	M4	65

MARKSGRADE TABLE

UpperBound	LowerBound	Grade
100	95	A+
94	85	A
84	70	B
69	65	B-
64	55	C
54	45	D
44	0	E

Source: Infosys Campus Connect Study Material

Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*

3NF Decomposition Algorithm

Let F_c be a canonical cover for *F;*

i := 0;

for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**

if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$

then begin

$i := i + 1;$

$R_i := \alpha\ \beta$

end

if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R

then begin

$i := i + 1;$

R_i := any candidate key for *R;*

end

return $(R_1, R_2, ..., R_i)$

3NF Decomposition Algorithm (Cont.)

- Above algorithm ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join

3NF Decomposition: An Example

- Relation schema:

  ***cust_banker_branch* = (<u>*customer_id, employee_id*</u>*, branch_name, type*)**

- The functional dependencies for this relation schema are:

  1. ***customer_id, employee_id* → *branch_name, type***
  2. ***employee_id* → *branch_name***
  3. ***customer_id, branch_name* → *employee_id***

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id* → *type

    employee_id* → *branch_name

    customer_id, branch_name* → *employee_id

3NF Decompsition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*<u>employee_id</u>, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added
- If the FDs were considered in a different order, with the 2nd one considered after the 3rd,

  (*<u>employee_id</u>, branch_name*)

  would not be included in the decomposition because it is a subset of

  (*customer_id, branch_name, employee_id)*
- Minor extension of the 3NF decomposition algorithm: at end of for loop, detect and delete schemas, such as (*<u>employee_id</u>, branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Comparison of BCNF and 3NF

- Relations in BCNF and 3NF
  - Relations in BCNF: no repetition of information
  - Relations in 3NF: problem of repetition of information
- Decomposition in BCNF and in 3NF
  - It is always possible to decompose a relation into relations in 3NF and
    - the decomposition is lossless
    - **dependencies are preserved**
  - It is always possible to decompose a relation into relations in BCNF and
    - the decomposition is lossless
    - **May some of the dependencies are not preserved.**

Multivalued Dependencies (MVDs)

- **Functional dependencies** rule out certain tuples from appearing in a relation. If *A B*, then we cannot have two tuples with the same *A* value but different *B* values.
- **Multivalued dependencies** do not rule out the existence of certain tuples. Instead, they **require** that other tuples of a certain form be present in the relation.
- Every functional dependency is also a multivalued dependency

$\twoheadrightarrow$

Multivalued Dependencies (MVDs)

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The ***multivalued dependency***

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_3[\beta] = t_1[\beta]$$

$$t_3[R - \beta] = t_2[R - \beta]$$

$$t_4[\beta] = t_2[\beta]$$

$$t_4[R - \beta] = t_1[R - \beta]$$

$\rightarrow\!\!\rightarrow$

MVD (Cont.)

- Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

MVD (Cont.)

Employee (E-name P-name D-name)

E-name	P-name	D-name
Smith	X	John
Smith	Y	Ann
Smith	X	Ann
Smith	Y	John

MVDs E-name $\rightarrow\rightarrow$ P-name and E-name $\rightarrow\rightarrow$ D-name hold in the relation:

The employee named Smith works on projects X and Y, and has two dependents John and Ann.

- **Trivial MVD**

If MVD X →→ Y is satisfied by all relations whose schemas include X and Y, it is called *trivial MVD.* X →→Y is trivial whenever $Y \subseteq X$ or $X \cup Y = R$

Inference Rules for Computing D^+

D: a set of FDs and MVDs

D^+ : the closure of D, the set of all FDs and MVDs logically implied by D

Sound and complete rules

1. reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$
2. augmentation: if $X \rightarrow Y$ then $WX \rightarrow Y$
3. transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
4. complementation: if $X \rightarrow\rightarrow Y$ then $X \rightarrow\rightarrow R - XY$

Inference Rules for Computing D^+

5. MV augmentation: if $X \rightarrow\rightarrow Y$ and $W \subseteq R$, $V \subseteq W$,
   then $WX \rightarrow\rightarrow VY$

6. MV transitivity: if $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$ then $X \rightarrow\rightarrow Z - Y$

7. replication: if $X \rightarrow Y$ then $X \rightarrow\rightarrow Y$

8. coalescence: if $X \rightarrow\rightarrow Y$ and $Z \subseteq Y$, $W \subseteq R$, $W \cap Y = \emptyset$, $W \rightarrow Z$,
   then $X \rightarrow Z$

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:

  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies

  2. To specify constraints on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.

- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations r' that does satisfy the multivalued dependency by adding tuples to *r.*

Merits of Normalization

- Normalization is based on a mathematical foundation.
- Removes the redundancy to a greater extent. After 3NF, data redundancy is minimized to the extent of foreign keys.
- Removes the anomalies present in INSERTs, UPDATEs and DELETEs.

Demerits of Normalization

- Data retrieval or SELECT operation performance will be severely affected.
- Normalization might not always represent real world scenarios.

Summary of Normal Forms

Input	Operation	Output
Un-normalized Table	Create separate rows or columns for every combination of multivalued columns	Table in 1 NF
Table in 1 NF	Eliminate Partial dependencies	Tables in 2NF
Tables in 2 NF	Eliminate Transitive dependencies	Tables in 3 NF
Tables in 3 NF	Eliminate Overlapping candidate key columns	Tables in BCNF

Source: Infosys Campus Connect Study Material

Points to Remember:

Normal Form	Test	Remedy (Normalization)
1NF	Relation should have atomic attributes. The domain of an attribute must include only atomic (simple, indivisible) values.	Form new relations for each non-atomic attribute
2NF	For relations where primary key contains multiple attributes (composite primary key), non-key attribute should not be functionally dependent on a part of the primary key.	Decompose and form a new relation for each partial key with its dependent attribute(s). Retain the relation with the original primary key and any attributes that are fully functionally dependent on it.
3NF	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes). In other words there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and form a relation that includes the non-key attribute(s) that functionally determine(s) other non-key attribute(s).

Source: Infosys Campus Connect Study Material

Database System Concepts, 5th Ed.