Chapter - 4

Database constraints and Normalization.

4.1 Integrity constraints and Domain Constraints
4.2 Assestions and Triggering
4.3 Functional Dependencies
4.4 Multi-valued & joined Dependencies.
4.5 Diffesent Normal Forms (1st, 2nd, 3rd, BCNF, DKNF)

Integrity constraints and Domain Constraints

↳ Integrity constraints are set of rules. It is used to maintain quality of information.

↳ Integrity constraints ensure that the data insestion, updating, and othes processes have to be performed in such a way that data integrity is not affected

↳ Thus integrity constraints is used to guard against accidental damage to the database.

Types of integrity Constraint

1. Domain Constraints:

↳ Domain constraints can be defined as the definition of a valid set of values for an attribute

↳ The data type of domain includes string, chasactes, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMESTER	AGE
1000	Prabesh	1st	17
1001	Sudhir	2nd	24
1002	AKhil	5th	21
1003	Athul	3rd	19
1004	Jay	8th	A

Not allowed: AGE is an integer attribute.

2. Entity integrity constraints.

↳ The entity integrity constraints states that psimary key value cann't be null.

↳ This is because the primary key value is used to identify individual rows in relation and if the psimary keys has a null value, then we cann't identify those rows

↳ A table can contain a null value othes than psimary key field.

Example:

Employee

EMP-ID	EMP-NAME	SALARY
123	Sudip	50000
142	Kundan	60000
164	Mithun	20000
	Vinod	27000

Not allowed as primary key can't contain a NULL value.

3. Referential integrity constraints.

↳ A referential integrity constraint is specified between two tables.

↳ In the referential integrity constraint, if a foreign key in Table 1 refers to the primary key of Table 2, then every value of the foreign key in Table 1 must be null or available in Table 2.

Example:

Table 1

Emp-id	NAME	AGE	D-NO
1	Pravin	20	11
2	Suman	40	24

3	Nabin	27	18	→ Not allowed:
4	Bal krishna	38	13	D-NO 18 is not

Relationships ↕ defined as a primary key of Table2 and in Table 1 D-NO is foreign key

Table2

primary key →

D-NO	D-Location
11	Muktinath
24	Itahari
13	Budhabare

4. Key constraints:

↳ keys are the entity set that is used to identify an entity within its entity set uniquely.

↳ An entity set can have multiple keys, but one key will be the primary key. A psimary key can contain a unique value in the relational table.

Example:

ID	NAME	SEMESTER	AGE
1000	Prabesh	1st	17
1001	Sudhir	2nd	24
1002	Akhil	5th	21
1003	Athul	3rd	19
1002	Jay	8th	22

Not allowed because all row must be unique.

Assertions and Triggering:

Assertion:

↳ An assertion is a predicate expressing a condition that we wish the database always to satisfy.

↳ Domain constraints and referential-integrity constraints are special forms of assertions.

↳ There are many constraints that we cannot express by using only these special forms.

Two Examples of such constraints are:

i. The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch

ii. Every loan has at least one customer who maintains an account with a minimum balance of 100000

Syntax:

create assertion <assertion-name> check <predicate>

↳ Since SQL doesnot provide a "for all X, P(X)"

assestion to demand that Birgunj based depastments donot employ trainers: (create assestion No_trainers_in_Birgunj as CHECK not exists (select 'trainee in Birgunj' from EMP e, DEPT d where e.DPTNO = d.DEPTNO and e.JOB = 'TRAINER' and d.LOC='Birgunj'))

construct (where P is a predicate), we ase forced to implement the construct by the equivalent "not exists x such that not P(x)" construct.

```
(create assertion sum_constraint check
(not exists (select * from branch
        where (select sum(amount) from loan
            where loan.branch-name = branch.branch-
            name) >=
            (select sum (balance) from account
        where account.branch-name = branch.branch_
                name));

(create assertion balance_constraint check
(not exists (select * from loan
        where not exists (select *
        from borrower, depositor, account
        where loan.loan_number = borrowes.loan-number
and borrowes.customes_name = depositor.customes_name
and depositor.account_numbes = account.account-
                number
and account.balance >= 100000)))
```

↳ When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it doesnot cause that assestion to be violated.

create assertion AT_MOST_PRESIDENT as CHECK
((select count(*) from EMP e WHERE e.JOB='PRESIDENT')<=1)

An assestion to demand that there's no more than a single president among employees.

Triggers:

↳ A trigger is a statement that the system executes automatically as a side effect of a modification to the database.

↳ To design a trigger mechanism, we must meet two requirements.

i. Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.

ii. Specify the actions to be taken when the trigger executes.

↳ Three parts of trigger.

i. Event (activates the trigger) insest, update or delete of the database.
ii. Condition : if the condition is true, the trigger executes otherwise skipped
iii. Action (what happens if the trigger runs) wide vasiety of options.

Syntax:

```
create trigger [trigger_name]
[ before | after]
{ insest | update | delete }
```

on [table-name]
[for each row]
[trigger-body]

i. create trigger [trigger-name]: creates or replaces an existing trigger with the trigger-name.

ii. [before|after]: This specifies when the trigges will be executed.

iii. {insert|update|delete}: This specifies the DML operation.

iv. on [table-name]: This specifies the name of the table associated with the trigger.

v. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

vi. [trigges-body]: This provides the opesation to be pesformed as trigger is fired.

Example:

```
create trigger sethull-trigger before update on r
referencing new row as nrow
for each row
when nrow.phone-numbes = ' '
set nrow.phone-numbes = null.


create trigger overdraft-trigges after updateon
 account
refesencing new row as nrow
for each row
when nrow.balance < 0
begin nrow.balance < 0
begin atomic
  insest into borrowes
      (select customes-name, account-numbes from
          depositor
      whese nrow.account-numbes = depositor.account-
          numbes);
    insest into loan values
        (nrow.account-numbes, nrow.branch-name,
            nrow.balance);
    update account
      set balance = 0
        whese account.account-numbes = nrow.
          account-numbes
  end.
```

Example of triggers on MSSQL server:

```
CREATE TRIGGER trgNoNewTables
ON DATABASE - STUDENT
FOR CREATE_TABLE
AS
BEGIN
    PRINT 'No new tuples please'
    ROLLBACK
END
```

This trigger won't allow us to create a new table on database named 'DATABASE-STUDENT'

```
CREATE TRIGGER trgNoChangeTables
ON DATABASE-STUDENT
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   PRINT 'No changes to table please'
   ROLLBACK
END.
```

To drop trigges.

```
DROP TRIGGER trgNoNewTables ON DATABASE-
                STUDENT;
```

TO disable trigger.

```
DISABLE TRIGGER trgNoChangesTables ON DATABASE-
STUDENT
```

TO enable trigger.

```
ENABLE TRIGGER trgNoChangesTables ON DATABASE-STUDENT
```

TO disable All triggers

```
DISABLE TRIGGER ALL ON DATABASE-STUDENT
```

TO enable All triggers.

```
ENABLE TRIGGER ALL ON DATABASE-STUDENT
```

Examples:

```
CREATE TRIGGER MyTrigger
ON customers
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SELECT * from deleted
    select * from insested
end.
```

Functional Dependencies:

↳ Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

↳ If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as $X \rightarrow Y$, which specifies Y is functionally dependent on X.

↳ Here X is a determinant set and Y is dependent attribute. Each value of X is associated with precisely on Y value

↳ Functional dependency in a database serves as a constraint between two sets of attribute.

↳ Defining functional dependency is an important past of relational database design and contributes to aspect normalization.

↳ Functional dependency typically exists between the primary key and non-key attribute within a table.

$$X \rightarrow Y$$

The left side of FD is known as determinant, the right side of the production is known as dependent.

For example:

Assume we have an employee table with attributes: Emp-Id, Emp-Name, Emp-Address

Here Emp-Id attribute can uniquely identify the Emp-Name attribute of employee table because if we know the Emp-Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp-Id → Emp-Name

We can say that Emp-Name is functionally dependent on Emp-Id

Types of Functional dependency:

i. Trivial functional dependency
ii. Non-trivial functional dependency.

1. Trival functional dependency:

↳ A → B has trivial functional dependency if B is a subset of A.

↳ The following dependencies are also trivial

A → A , B → B

Example:

Consides a table with two columns Employee-Id a Employee-Name.

{Employee-id, Employee-Name} → Employee-id

Date.

Compound Detesminants:

If more than one attribute is necessary to detesmine another attribute in an entity, then such detesminant is tesmed a composite detesminant.

Full Functional Dependency:

only of relevance with composite detesminants. This is the situation when it is necessary to use all the attributes of composite detesminant to identify its object uniquely.

if A and B are attributes of a relation, B is fully functionally dependent on A ifs B is functionally dependent on A, but not on any proper subset of A.

Partial Functional Dependency:

This is situation that exists if it is necessary to only use a subset of the attributes of the composite detesminant to identify its object uniquely.

A functional dependency A → B is a pastially dependency if there are some attributes that can be removed from A & yet the dependency holds.

Date.

i. Trivial functional Dependency.

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes include that attribute.

$A \rightarrow B$ has trivial functional dependency if B is a subset of A.

The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Example:

Consider a table with two columns Employee_Id and Employee_Name

{Employee_Id, Employee_Name} → Employee_Id is a trivial functional dependency as:

Employee_Id is subset of {Employee-Id, Employee_Name}

Also Employee_Id → Employee_Id

Employee_Name → Employee_Name are trivial dependencies too.

Date.

ii. Non-trivial functional dependency.

A → B has non trivial functional dependency if B is not a subset of A.

When A intersection B is NULL, then A → B is called as complete non-trivial

Example:

i. ID → Name

Name → DOB

ii.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	58
Amazon	Jeff Bezos	55

{Company} → {CEO}

But CEO is not a subset of company and hence non-trivial functional dependency

Date.

Armstrong's Axioms property of Functional Dependency.

Armstrong's Axioms property was developed by William Armstrong in 1974 to reason out about functional dependencies

The propesty suggests rules that hold true if the following are satisfied.

1. Transitivity
   if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
   i.e. transitive relation

2. Reflexivity
   $A \rightarrow B$, if B is a subset of A

3. Augmentation
   The last rule suggests: $AC \rightarrow BC$,
   if $A \rightarrow B$

Characteristics of Functional Dependencies.

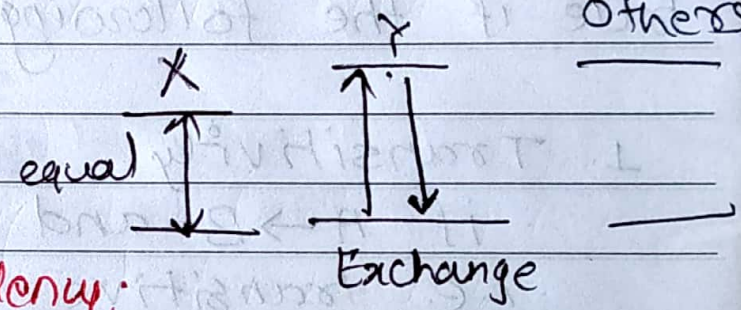
↳ There is a one-to-one relationship between the attribute(s) on the left hand side (detesminant) and those on the

Date.

right-hand side of a functional dependency.

↳ Holds for all time

↳ The detesminant has the minimal numbes of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side.



Multivalued Dependency:

↳ Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.

↳ A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces

For MVD refer mathematical definition too.

Date.

two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2001	2008	white
M2001	2008	Black
M3001	2013	white
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

These two columns can be called as multivalued dependent on BIKE_MODEL.

Representation of these dependencies is shown below:

BIKE_MODEL →→ MANUF_YEAR

BIKE_MODEL →→ COLOR

which can be read as "BIKE_MODEL

multidetesmined MANUF-YEAR" and "BIKE-MODEL multidetesmined COLOR".

Join Dependency:

↳ If a table can be recreated by joining multiple tables and each of this table have a subset of the attribute of the table, then the table is in join dependency.

↳ Join dependency is a further generalization of Multivalued dependencies.

↳ If the join of R1 and R2 overs C is equal to relation R, then we can say that join dependency (JD) exists

Where R1 and R2 are the decompositions R1(A,B,C) and R2(C,D) of given relations R(A,B,C,D)

Example:

Date.

< Employee >

EmpName	EmpSkills	EmpJob
Arjun	Networking	EJ001
Toni	Web Development	EJ002
Aswin	Programming	EJ002

The above table can be decomposed into following three tables.

< Employeeskills >

EmpName	Empskills
Arjun	Networking
Toni	Web Development
Aswin	Programming

< EmployeeJob >

EmpName	EmpJob
Arjun	EJ001
Toni	EJ002
Aswin	EJ002

Date.

<Jobskills>

EmpSkills	EmpJob
Networking	EJ001
Web Development	EJ002
Programming	EJ002

Join Dependency:

$\{(EmpName, EmpSkills), (EmpName, EmpJob), (EmpSkills, EmpJob)\}$ is equal to the relation <Employee>

Normalization:

↳ Database normalization is the process of removing redundant data from tables to improve storage efficiency, data integrity and scalability.

↳ Database normalization is a technique of organizing the data in the database.

↳ Normalization genesally involves

Date.

splitting existing tables into multiple ones, which must be rejoined or linked each time a query is issued.

↳ Normalization is multi-step process

↳ Normalization is the process of efficienty organizing data in database with two goal:

i. Eliminate redundant data
ii. Ensure data dependencies make sense.

↳ Data normalization also may improves data consistency and simplify future extension of the logical data model.

↳ The formal classifications used for describing a relational databases level of normalization are called normal forms.

↳ Normal Forms is abbreviated as NF

anomalies → sth that deviates from what is standard.

Date.

What happen without Normalization?

↳ A non-normalized database can suffer from data anomalies.

↳ A non-normalized database may store data representing a particular referent in multiple locations.

↳ An update to such data in some but not all of those locations results in an update anomaly, yielding inconsistent data.

↳ A non-normalized database may have inappropriate dependencies, i.e. relationships between data with no functional dependencies.

Example:

Student_Table

Roll_NO	Name	Branch	HOD	Office-Tel
1	Jagdish	BCT	Hasi	01-5337
2	Yagya	BCT	Hasi	01-53337

Date.

	3	Sumah	BCT	Hari	01 53337
	4	Shankar	BCT	Hari	01 53337

Insestion Anomaly.

↳ suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted or else we will have tosed the branch information as NULL.

↳ if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

Updation Anomaly.

↳ if Hari is no longer the HOD of computer depastment or what if Hari leaves the college?

↳ In that case all student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.

Date.

Deletion Anomaly:

In our student table, two different informations are kept together, student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

Advantages of Normalization.

↳ Elimination of redundant data storage

↳ Closed modeling of real world entities, processes and their relationship.

↳ structuring of data so that model is flexible.

↳ Less storage space.

↳ Easier to add data and avoid certain updating anomalies.

↳ Facilitate the enforcement of data constraints.

Normalized database

↳ Normalized databases have a design that reflects the true dependencies

↳ Instead of attempting to lump all information into one table, data is spread out logically into many tables.

↳ Normalizing the data is decomposing a single relation into a set of smaller relations which satisfy the constraints of the original relation.

↳ Redundancy can be solved by decomposing the tables. However certain new problems are caused by decomposition.

↳ Normalization helps us to make a conscious decision to avoid redundancy keeping the pros and cons in mind.

Date.

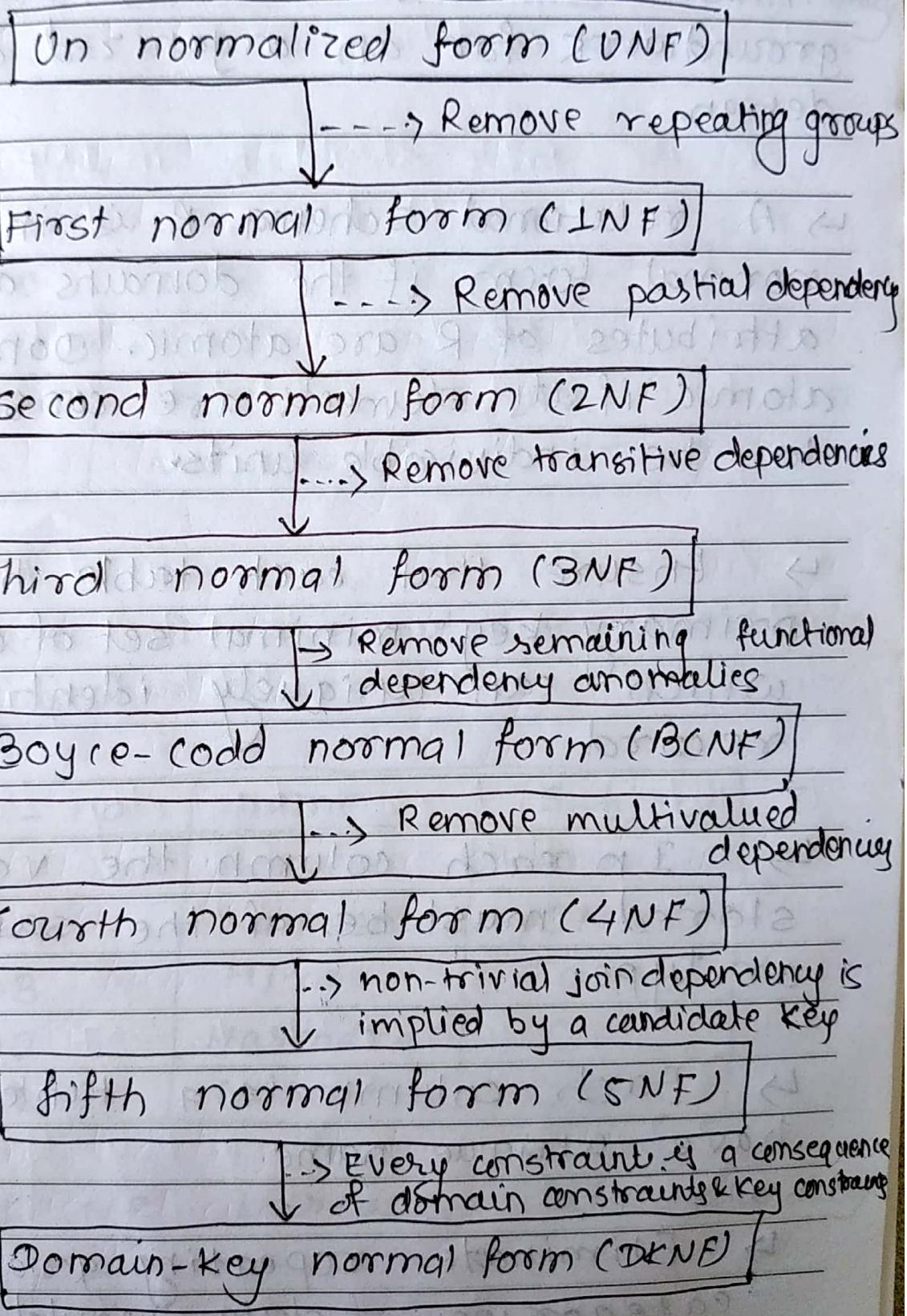
↳ One can only describe a database as having a normal form if the relationships between quantities have been clearly defined.

↳ It is possible to use set theory to express this knowledge once a problem domain has been fully understood; but most database designers model the relationship interms of an "idealized schema" (The mathematical support came back into play in proofs regarding the process of transforming from one form to another.)

↳ The transformation of conceptual model to computer representation format is known as Normalization.

Date.

Stages of Normalization.



Date.

First Normal Form (1NF)

↳ First Normal form (1NF) lays the groundwork for an organized database design.

↳ A relational schema R is in first normal form if the domains of all attributes of R are atomic. Domain is atomic if its elements are considered to be indivisible units.

↳ Ensure that each table has a primary key: minimal set of attributes which can uniquely identify a record.

↳ In each column the values stored must be of the same kind or type.

↳ Each column is a table should have unique name.

↳ Eliminate repeating groups (categories of data would seem

Date.

to be required a different numbers of times on different records) by defining keyed and non-keyed attributes appropriately

Example:

roll-no	name	subject
101	Ramesh	OS, CN
103	Hira	Java
102	Narendra	C, C++

101 and 102 have opted for more than one subject.

roll-no	name	subject
101	Ramesh	OS
101	Ramesh	CN
103	Hira	Java
102	Nasendra	C
102	Nasendra	C++

1 NF increases data redundancy

Date.

Second Normal Form (2NF)

↳ For a table to be in 2NF, it must satisfy two conditions

i. The table should be in 1NF
ii. There should be no pastial dependency

↳ if a table has a composite key, all attributes must be related to the whole key.

↳ A relation R is in 2NF if R is in 1NF and all non-prime attributes are fully dependent on the candidate keys.

↳ Data which is redundantly duplicated across multiple rows of a table is moved out to a separate table.

Converting from 1 NF to 2NF

i. Identify the primary key for the 1NF relation.

ii. Identify the functional dependencies

in the relation.

iii. If partial dependencies exist on the primary key remove them by placing them in new relation along with a copy of their detesminant.

Example:

Subject_Table

Subject_Id	Subject_Name
1	Java
2	C++
3	PHP

Score_Table.

Score_Id	Student_Id	Subject_Id	Marks	Teacher
1	10	1	70	karthick
2	10	2	75	Ranjit
3	11	1	68	karthick

Student_Id + Subject_Id forms a candidate key.

Date.

↳ Teachers name only depends on subject, hence the subject-Id & has nothing to do with student-Id.

Partial Dependency:

It is a condition where an attribute in a table depends on only part of the primary key and not the whole key.

↳ Remove column teacher from score table and add it to the subject_Table.

Subject_Table

Subject-Id	Subject_Name	Teacher
1	Java	karthick
2	C++	Ranjit
3	PhP	Arun
4	DBMS	Shiva

Score_Table

Score-Id	Student-Id	Subject-Id	Marks
1	10	1	70
2	10	2	75
3	11	1	68

Date.

Closure of a set of Functional dependencies.

↳ A closure is a set of all possible FDs that can be derived from given set of FD's

↳ Closure set is also referred as a complete set of FD's.

↳ If F is used to donate the set of FDs for relation R, then a closure of a set of FDs implied by F is denoted by F+

↳ Lets consider the set F of functional dependencies given below:

$F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D \}$

from F it is possible to derive following dependencies.

$A \rightarrow A$	$A^+ \rightarrow ABCD$ (Union)
$A \rightarrow B$	Closure of A under F
$A \rightarrow C$	
$A \rightarrow D$	
$A \rightarrow ABCD$	

All such type of FDs desired from each FD of F form a closure of F.

Date.

Additional Rules:

Union rule:

If $\alpha \to B$ holds and $\alpha \to \gamma$ holds then $\alpha \to B\gamma$ holds.

Decomposition rule:

If $\alpha \to B\gamma$ holds, then $\alpha \to B$ and $\alpha \to \gamma$ holds.

Pseudotransitivity rule:

If $\alpha \to B$ holds and $\gamma B \to \delta$ holds then $\alpha \gamma \to \delta$ holds.

Steps to detesmine F+

i. Determine each set of attributes X that appears as a left hand side of some FD in F.

ii. Determine the set X+ of all attributes that are dependent on X.

iii. All such sets of X+, in combine form closure of F.

Date

Examples:

For given FD, find all possible F+

$R = \{A, B, C, G, H, I\}$

$F = \{A \to B,\ A \to C,\ CG \to H,\ G \to I,\ B \to H,\ CG \to I\}$

Some members of F+

$A \to H$ from $A \to B$ and $B \to H$ by transitivity

$AG \to I$ by augmenting $A \to C$ with G, to get $AG \to CG$ and then transitivity with $CG \to I$

$CG \to HI$ by

Problem 1: For given FD

$R = (A, B, C, D, E, G, H, I, J)$

$F = \{AB \to E,\ AG \to J,\ BE \to I,\ E \to G,\ GI \to H\}$

i. List all possible F+

ii. Does $AB \to GH$?

Date.

Problem 2: For given FD

$R = (A, B, C, D, E, F, G)$

$F = \{A \to B, ABCD \to E, EF \to G\}$

i. List all possible F+

ii. Does $ACDF \to G$, implied by the set of FD's?

Problem 3: For given two set of F_1 & F_2 for a relation

$F_1 = \{A \to B, AB \to C, D \to AC, D \to E\}$

$F_2 = \{A \to BC, D \to AE\}$

F_1 & F_2 are equivalent or not?

Problem 4: Let $R = (A, B, C, D, E, F)$

$F = \{A \to BC, B \to E, CD \to EF\}$

i. List all possible F+

ii. Does $AD \to F$ implied by the set of FD's?

Date.

Uses of Attribute Closure.

↳ Testing for super key -

To test, if α is a super key, we compute α^+, and check if α^+ contains all attributes of R. Then α is a super key of R

↳ Testing functional dependencies.

For example: to check whether a FD like $A \rightarrow B$ holds in a relation, find the A^+. If B is in A^+, then functional dependency $A \rightarrow B$ is valid.

Example:

For given FD, find closure of $(AG)^+$

$R = (A, B, C, G, H, I)$

$F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, G \rightarrow I, B \rightarrow H, CG \rightarrow I \}$

Solution:

result = AG

= ABCG ($A \rightarrow C$ and $A \rightarrow B$)

= ABCGH ($CG \rightarrow H$ and $CG \subseteq ABCG$)

= ABCGHI ($CG \rightarrow I$ and $CG \subseteq AGBCH$)

Here AG is the candidate key because closure of AG contains all attributes of R

Problem 5:

Let R = (A, B, C, D, E)

F = {B → CD, B → A, E → C, AD → B, D → E}

is B → E in F+ ?

List out all possible supes key and candidate key of above given prooblems. (i.e 1, 2, 3, 4, 5)

Canonical Cover

↳ set of functional dependencies may have redundant dependencies that can be inferred from the others.

For example: A → C is redundant in

{A → B, B → C, A → C}

↳ Parts of a functional dependency maybe redundant.

Date.

Eg: on RHS

{A → B, B → C, A → CD} can be simplified to {A → B, B → C, A → D}

Eg: on LHS

{A → B, B → C, AC → D} can be simplified to {A → B, B → C, A → D}

↳ A canonical cover of F is a "minimal" set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies.

Third Normal Form (3NF)

↳ A table is said to be in third normal form when

i. It is in second normal form.
ii. And it doesn't have Transitive dependency.

↳ 3NF requires the data stored in a

table be dependent only on the primary key, and not on any other field in the table.

↳ Converting 2NF to 3NF

i. Identify the primary key in the 2NF relation.

ii. Identify functional dependencies in the relation.

iii. If transitive dependencies exist on the primary key, remove them by placing them in a new relation along with a copy of their determinant.

Student-Table

Student_Id	Name	Reg-No	Branch	Address
10	Hari	07-WY	BCT	Beni
11	Shyam	08-WY	BEX	Baglung
12	Santosh	09-WY	BEX	Mustkot

Date.

Subject_Table

subject-Id	Subject-Name	Teacher
1	Java	karthick
2	c++	Ranjit
3	php	Arun

Score-Table

score-id	student-Id	Subject_Id	Marks
1	10	1	70
2	10	2	75
3	11	1	65

Lets add 2 more columns to score table

Score-Id	student-Id	Subject_Id	marks	Exam-name

total_marks

Primary key:
Composite key → student-Id + subject-Id

Total_marks depends on exam name as with exam type and total score changes

Date.

For example: practicals are of less marks while theory exams are of more marks. exam-name isnot a primary key but total marks depend on it.

This is transitive dependency. When a non prime attribute depends on other non prime attributes rather than depending upon the prime attributes or primary key.

Score_Table: In 3rd Normal Form

Score_Id	Student_Id	Subject_Id	marks	exam_id

The new Exam table

exam_id	exam_name	Total_marks
1	workshop	200
2	Mains	70
3	Practicals	30

Date.

Advantages of removing transitive dependency:

i. Amount of data duplication is reduced
ii. Data integrity is achieved.

Boyce-Codd normal form (BCNF)

↳ A relation is in BCNF if and only if, every determinant is a candidate key.

↳ R must be in 3NF

↳ For each functional dependency X → Y, X should be a super key

X cannot be non-prime attribute, if Y is a prime attribute.

↳ If table contains only one candidate key, the 3NF and BCNF are equivalent

↳ BCNF is a special case of 3NF.

Date.

Example:

Student_Id	subject	Professor
101	Java	Arya
101	C++	Sansa
102	Java	Brendon
103	C#	John
104	Java	Tyrion

In the above table:

One student can enroll for multiple subject. ~~like. Java~~

There can be multiple professors teaching one subject.

Hese

Student_Id + subject together form the psimary key, because using student_Id and subject, we can find all columns of the table.

↳ There is dependency between subject & professor here where subject depends on the professor name.

To make this table satisfy BCNF, we will decompose this table into two tables, a student table and professor table.

Student-Table

Student-Id	P-Id
101	1
101	2
102	3

And Professor-Table

P-id	professor	subject
1	Arya	Java
2	Sansa	c++

Fourth Normal form (4NF)

For a table to satisfy fourth normal form, it should satisfy following two conditions.

i. It should be in BCNF
ii. And, the table shouldnot have any Multi-valued dependency.

Date.

S-Id	course	hobby
1	Science	cricket
	Maths	Hockey
2	C#	cricket
	php	Hockey

S-Id 1 has opted for two courses, science and Maths and has two hobbies, cricket & Hockey. which will lead to following table.

S-Id	course	hobby
1	Science	cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	cricket

t1 1 Science Cricket
t2 1 Science Hock

Here course and hobby are independent of each other.

CourseOptedTable

S-Id	course
1	science
1	Maths
2	C#
2	php

Hobbies Table

S-Id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Domain key Normal form (DKNF)

A relation will be in DKNF if and only if every content of the relation is a logical sequence of domain constraints or key constraint

Domain constraint :

EmpID	EmpName	EmpAge
0921	Mahesh	33
0922	Bipin	31

Attributes should have some set of values, for example EmpID should be four digits long.

key constraint.

An attribute or its combination is a candidate key.

General constraint
predicate on the set of all relations.

More examples:

1NF

i. LiveIN

Name	Address		
	City	Year_Move-in	Years-left
Ram	Kathmandu	2000	2010

Normalized Table:

Name	City	Year_Move-IN	Years-Left
Ram	Kathmandu	2000	2010

ii. ORDER(Order-no, order-date, cust-no, cust-name, cust-add, order-total)

PRODUCTS_DES (ordesno, prod-no, prod-des, unit-price, order-qty, line-total)

Date.

iii

T_1 (Patient-Num, F-name, L-name, ward-num, ward-Name)

T_2 (Patient-Num, Prescsiption-date, Drug-code, Drug-Name, Dosage, length-of-Treatment)

2NF:

i. ORDER (order-no, order-date, cust-no, cust-name, cust-add, order-total)

PRODUCT-2 (prod-no, prod-desc, unit-price)

ORDER-LINE-2 (order-no, prod-no, ord-qty, line-total)

ii: T_1 (Patient-Num, F-name, L-name, ward-num, ward-Name)

T_2 (Patient-Num, prescsiption-date, Drug-code, Dosage, length-of-Treatment)

T_3 (Drug-code, Drug-Name)

Date.

3NF:

i. PRODUCT-2 (prod-no, prod-desc, unit-price)

ORDER-LINE-2 (order-no, prod-no, ord-qty, line-total)

CUSTOMER-3 (order-no, order-date, cust-no, order-total)

ii. T1 (Patient_Num, F-name, L-name, ward-num)

T2 (Patient_Num, prescription_date, Drug-code, Dosage, Length_of_Treatment)

T4 (ward_Num, ward_Name)

BCNF:

Relation R and functional dependency F

R= (branch-name, branch-city, assets, customer-name, loan-numbes, amount)

F= { branch-name → assets, branch_city,

loan_number → amount branch_name}

key = {loan_number, customer_name}

R1 = (branch_name, branch_city, assets)

R2 = (branch_name, loan_number, amount)

R3 = (customer_name, loan_number)

4NF:

Employee (Ename, Pname, Dname)

Ename →→ Pname

Pname →→ Dname.

Ename multi determines Pname (1:N), i.e single value of Ename associated with multiple value of pname.

pname multi determines Dname (1:N); i.e single value of pname associated with multiple value of Dname.

Employee:

Ename	Pname	Dname
Sita	X	Ram
Sita	Y	Rita
Sita	X	Rita
Sita	Y	Ram

EMP- Project

Ename	pname
Sita	X
Sita	Y

EMP- Dependant

Ename	Dname
Sita	Ram
Sita	Rita

Relational Decomposition.

↳ When a relation in the relational model is not in appropriate normal form then the decomposition of relation is required.

↳ In database, tables are broken into multiple tables.

↳ If the relation has no propes decomposition, then it may lead to problems like loss of information.

↳ Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Date.

Types of Decomposition:

i. Lossless Decomposition
ii. Dependency Presesvation.

Lossless Decomposition.

↳ If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.

↳ The lossless decomposition gurantees that the join of relations will result in the same relation as it was decomposed.

↳ The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

Employee-Department table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID
22	Dinesh	28	Mahendranagar	827
33	Ajing	25	Dipayal	438

				869	Sales Masketing Finance
46	Sambriddi	30	Baglung		
52	Kamal	36	Mustang	575	Production
60	Jamuna	40	Nepalguny	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE

EMP-ID	EMP-NAME	EMP-AGE	EMP-CITY
22	Dinesh	28	Mahendranagas
33	Aling	25	Dipayal
46	Sambriddi	30	Baglung
52	Kamal	36	Mustang
60	Jamuna	40	Nepalguny

DEPARTMENT

DEPT-ID	EMP-ID	DEPT-NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP-ID", then

the resultant relation will be equivalent to Employee_Department table

Employee ⋈ Department ≡ Employee_Department

Hence, the decomposition is lossless join decomposition.

Dependency Preservation.

↳ A decomposition $D = \{R_1, R_2, \ldots R_n\}$ of R is dependency-preserving with respect to F if the union of the projections of F on each R_i in D is equivalent to F.

i.e

$$if\ (F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

↳ If a relation R is decomposed into Relation R_1 and R_2, then the dependencies of R either must be part of R_1 or R_2 or must be desived from the combination of functional dependencies of R_1 and R_2.

Example:

Suppose there is a relation R(A, B, C, D) with functional dependency set $A \rightarrow BC$.

Date.

The relation R is decomposed into $R_1(ABC)$ and $R_2(AD)$ which is dependency preserving because FD $A \rightarrow BC$ is a part of relation $R_1(ABC)$