MODULE -4

Schema Refinement and Normal Forms

Schema Refinement and Normal Forms: Introduction to Schema Refinement, Functional Dependencies, Reasoning about Functional Dependencies. Normal forms, Properties of decomposition, Normalization, Different types of dependencies.

Schema Refinement: Database Schema of a Database system is its structure described in a formal Language supported by the DBMS.

Schema: It is a blueprint of how the database is constructed. And IC imposed on a database and IC's can be used to refine the conceptual schema which is produced by translating on ER MODEL Design into a collection of Relations.

An important class of constraints called Functional Dependencies and other kinds of IC's for Eg – Multivalued dependencies and Join Dependencies provide useful information. Schema Refinement is intended to address and a refinement approach based on Decompositions. And Decomposition can eliminate redundancy, it leads to problems of its own.

Decomposition:

Decomposition in DBMS reduces the redundancy, eliminates anomalies and inconsistencies from a database by dividing the table into multiple tables.

Problems caused by Redundancy: Redundancy means storing the same information in multiple locations i.e more than once in a data base, that is not needed normally, because that can be used in the event of failure of disk to retrieve the loss of information. This redundancy can lead to several problems.

1. **Redundant storage:** Same information is stored repeatedly.
2. **Update anomalies:** If one copy of among such repeated data is updated an inconsistency is created until all the copies are simultaneously updated.
3. **Insertion anomalies:** it may not be possible to insert certain information unless and until some other information should be added with it.
4. **Deletion anomalies:** Sometimes it may not be possible to delete some information unless and until some other useful information should be deleted along with it.

Ex: Consider a relation obtained by translating the variants of an Hourly_employess entity set

Hourly_employees(ssn,name,lot,rating,hourly_wages,hourly_worked)

The attributes are abbreviated cto a single letter and refers to a relation schema by a string of letters.

S N L R W H

ssn name lot rating hourly_wages hourly_worked

ssn	name	lot	rating	hourly_wages	hourly_worked
1	Alice	48	8	10	40
2	Bob	22	8	10	30
3	Smith	35	5	7	30
4	Ram	35	5	7	32
5	Rahul	35	8	10	40

Now consider this table, the same value appears in ratings column of two tuples, the IC's tells us that same value must appear in hourly_wages column as well. This IC is an example of functional dependency and now it leads to possible redundancy in a relation Hourly-employees.

Redundant storage: The rating value 8 corresponds to the hourly_wages 10, and this association is repeated 3 times.

Insertion anomalies: We cannot insert a tuple for an employee unless; we know the hourly wages for the employee's rating value.

Deletion anomalies: If we delete all tuples with a given rating value, we loose the association between the rating value and its hourly_wages value.

Update anomalies: Updating the hourly_wages in the first tuple without making a change in the 2nd ,5th tuple will lead to update anolamies.An inconsistency will be created unless all the similar copies are not updated.

Finally, the schema should not permit redundancy, but it can allow redundancy by accepting the drawbacks.

Decomposition:

Redundancy arises when a relation schema forces an association between attributes. Functional dependencies can be used to identify such situations and suggests refinement to the schema. Refinement means redefining the schema. Due to the problems of redundancy, for eliminating redundancy, decomposition is used in DBMS.

A decomposition of a relation schema R consists of two or more relational schema that each contains a subset of the attributes of R and together includes all the attributes in R.

Example: decomposing the hourly_employees relation into two sub relations,

Relation 1:

Hourly_employees2(ssn, name, lot, rating, hourly_worked)

Ssn	name	lot	rating
1	Alice	48	8
2	Bob	22	8
3	Smith	35	5
4	Ram	35	5
5	Rahul	35	8

Relation 2:

Wages(rating, hourly_wages)

rating	hourly_wages
8	10
5	7

Note: Now we can easily record the hourly_wages for any rating by adding a tuple to wages instead of updating several tuples, and also it eliminates the inconsistency.

Problems related to decomposition:

Decomposing a relation schema can create more problems. Two important questions must be asked repeatedly.

1. Do we need to decompose a relation?
2. What problems do give decomposition cause?

1) Do we need to decompose a relation?

Several normal forms have been proposed for relations (1NF to 5NF). If the relation schema is in one of these normal forms, the problems cannot arise. By considering the normal form of a given relation schema can help us to decide whether or not to decompose it. In case, if we

decide that a relation schema must be decomposed further, we must choose a particular decomposition (i.e., a particular collection of smaller relation to replace the given relation).

2) What problems do given decomposition cause?

Properties of decomposition:

a) Lossless-join property
b) Dependency-preservation property.

Lossless Join Decomposition:

Let R be a relational schema and set F be a set of FDs over R

A decomposition of R into 2 schemas with attribute sets X and Y is said to be a lossless join decomposition with respect to F if for every instance r of R that satisfies the dependency in F i.e,

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

3 conditions need to be satisfied for lossless join decomposition:

i) attribute(R1) U attribute(R2)=attribute(R). i.e after joining the decomposed relations the number of attributes should be the same as attributes of R before decomposition.

ii) attribute(R1) ∩ attribute(R2) $\neq \Phi$: **i.e at least one attribute has to be there in the decomposed relations.**

iii) attribute(R1) U attribute(R2)=attribute(R1)

or

attribute(R1) U attribute(R2)=attribute(R2)

Or BOTH

i.e. the common attribute should be either a Super key or a candidate key in either R1 or R2 or in both R1 and R2.

Can we recover the original relation from decomposed relations

Supervisor id	Project id	Department id
S1	P1	D1
S2	P2	D2
S3	P1	D3

After Decomposition

Supervisor id	Department id
S1	D1
S2	D2
S3	D3

Project id	Department Id
P1	D1
P2	D2
P1	D3

Lossy Decomposition:

If R is a subset of (R1 JOIN R2) then Lossy Decomposition.

Main Table:

SSN	NAME	ADDRESS
1	JOE	CHICAGO
2	BOB	UK
3	BOB	US

After decomposition the combination is Lossless Join.

Decomposed table:

LET R1 =(SSN, NAME)

R2 =(NAME, ADDRESS)

R1 (SSN, NAME)

SSN	NAME
1	JOE
2	BOB
3	BOB

R2 = (NAME, ADDRESS)

NAME	ADDRESS
JOE	CHICAGO
BOB	UK
BOB	US

Now, JOIN the R1 and R2

SSN	NAME	ADDRESS
1	JOE	CHICAGO
2	BOB	UK
2	BOB	US
3	BOB	UK
3	BOB	US

After decomposed the Relations and perform Join Operation, Here, the information lost in this Example, is the address for person 2 and 3. In the original relation R person 2 lives at UK and 3 at US. After joining the ables R1 and R2, the Person 2 either lives at UK or US.

This is how Extra information after joining decomposed relations can result in lossy decomposition. The records were not lost but we lost the information about which records were in the Original Table.

Dependency Preservation:

It is used to allow us to check the integrity constraints efficiently.

Let R be a relation schema that is decomposed into 2 schemas with attributes X and Y. Let F be a set of FD's over R.

A decomposition D={R1,R2,R3…Rn} of R is dependency preserving with respect to a set F of functional dependency if (F1 U F2 U … Fn)+ = F+

Consider a relation R

R— >F (with some FD)

R is decomposed or divided into R1 with FD{F1} and R2 with {F2} , then there can be 2 cases:

1. F1U F2= F decomposition is dependency preserving
2. F1 U F2 is a subset of F, not dependency preserving

Example:

Problem: Let a relation R(A,B,C,D) and functional dependency { AB—> C , C—>D, D—>A}. Relation R is decomposed into R1 (A ,B,C), R2(C,D) check whether decomposition is dependency preserving or not.

Solution:

R1(A,B,C) and R2(C,D)

Let us find closure of F1 and F2 to find closure of F1, consider all combination of ABC, i.e, find closure of A,B,C,AB,BC,AC.

Closure (A). = {A} // trivial

Closure (B). = {B} // trivial

Closure (C). = {C,A,D} // but D cannot be in closure as D is not present in R1

= {C,A}.

C—>A // removing C from right side as it is trivial attribute

Closure (AB). = {A,B,C,D} // D should be removed

{A,B,C}

AB—>C // removing AB from right side as these are trivial attributes

Closure (BC). = {B,C,D,A}

{B,C,A}

BC—> A // removing BC from right side as these are trivial attribute

F1= {C—>A, AB—>C, BC—> A}

Similarly F2= {C—> D}

Functional Dependencies(FD):

The FD is a kind of integrity constraint that generalises the concept of a key. Let R be a relation schema and let X and Y be non empty of attributes in R we say that an instance r of R satisfies the FD X—> Y if the following holds for every pair of tuples t1 and t2 in R.

If t1.X = t2.X Then t1.Y=t2.Y

The notation t1.X to refer to the projection of tuples t1 onto the attributes in X. [from TRC: t.a for referring to attribute 'a' of tuple t]

An FD **X—>Y** essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attribute Y.

X is called the **determinant** (a set of attributes).

Y is called the **dependent** (a set of attributes).

This means that the value of the attributes in Y is uniquely determined by the value of the attributes in X.

Eg: FD: AB—>C by showing an instance that satisfies dependency.

First 2 tuples show an FD is not same as key constraint although the FD is not violated, AB is clearly not a key for the relation.

A	B	C	D
A1	B1	C1	D1
A1	B1	C1	D2
A1	B2	C2	D1
A2	B1	C3	D1

An instance that satisfies AB—>C

The 3rd and 4th tuples shows that if 2 tuples differ in either the A field or B field, they can differ in the C field without violating FD.

If we add a tuple < A1,B1,C2,D1>: the resulting instance would violate the FD. [

note: X—>Y read as X functionally determines Y, or simply as X determines Y].

Reasoning about Functional Dependencies:

Reasoning about functional dependencies is essential for database design, especially when normalizing databases to reduce redundancy and improve data integrity.

A set of FD's over a relation schema R, there are several additional FD's that hold over R.

Example: Consider

Workers (ssn, name,lot,did,since)

SSN—>did Where ssn is the key

Did—>lot

If the 2 tuples have the same ssn value, they must have the same did value. From 1st FD and because they have the same did value, they must also have the same lot value from 2nd FD.

The FD ssn—>lot also holds on Workers.

An FD F is implied by a given set F of FD's if F holds on every relation instance that satisfies all dependencies in F.

Note: it is not sufficient for F to hold on some instance that satisfies all dependencies in F, rather F must hold on every instance that satisfies all dependencies in F.

Trivial Functional Dependency: A functional dependency X→Y is trivial if Y is a subset of X. For example, {A,B}→A is trivial because A is a part of {A,B}.

Non-Trivial Functional Dependency: A functional dependency X→Y is non-trivial if Y is not a subset of X. For example, A→B is non-trivial if B is not a part of A.

Fully functional dependency:

In a relation, there exists full functional dependency between any two attributes X and Y, when X is functionally dependent on Y and is not functionally dependent on any proper subset of Y.

For example: $ABC \rightarrow D$, (Here D is fully functionally dependent on ABC).

Fully functional dependent means, the D value determined by combination of attributes but not on any subset of attributes A, B, C.

For example: Consider a relation ($StudentID, sname, proof\ ID, grade$)

ID	$sname$	$proof\ ID$	$grade$
S1	John	P2	8
S2	Smith	P1	9

Combination of two attributes, $ID, proof\ ID$

$ID, proof\ ID \rightarrow grade$, when one attribute depends on two or more attributes , then it is called as FDD.

Closure of a set of FD's:

The set of all functional dependencies implied by a given set F of functional dependencies is called the closure of F, denoted by F^+. By using inference rules or Armstrong axioms we can compute the closure.

- F^+=superset of FDs F
- F^+=set of all given FDs F + additionally inferred FDs.

Armstrong Axioms:

1. **Reflexivity rule** (Trivial functional dependency): Consider the attributes X, Y, Z, W, if X is a set of attributes and $Y \subseteq X$, then $X \rightarrow Y$.
2. **Union rule:** $if\ X \rightarrow Y\ and\ X \rightarrow Z\ holds, then\ X \rightarrow YZ$
3. **Decomposition rule:** $if\ X \rightarrow YZ\ holds, then\ X \rightarrow Y\ and\ X \rightarrow Z\ also\ holds.$
   Example: $Eid \rightarrow name, age, then\ Eid \rightarrow name\ and\ Eid \rightarrow age.$
4. **Augmentation rule:** Consider $X \rightarrow Y$, we can add same attribute name on both sides. $if\ X \rightarrow Y, then\ XZ \rightarrow YZ\ for\ any\ Z.$
5. **Transitivity rule:** $if\ X \rightarrow Y\ and\ X \rightarrow Z\ holds, then\ X \rightarrow Z$

Example 1:

Given a relation R, R(A,B,C,D,E,F)

FDs,F

FD1:A->BC

FD2: B->E

FD3: CD->EF

Show That the FD, AD->F hold good for the relation R,

Solution: Use Armstrong Axioms

1. A->BC--given
2. Decomposition on 1 : 2.1 A->B, 2.2 A->C
3. Augmentation of D on 2.2(A->C) , 3.1 AD-> CD
4. Transitivity of step 3 FD 3.1(AD->CD) with the FD3(CD->EF) 4.1 AD->EF
5. Decomposition of step 4 FD 4.1(AD->EF): AD->F

From Step 5, we have got AD->F, hence we can say that AD->F hold good for the relation R.

Example 2:

Contracts relation:

Contracts(contractId, supplierid, projectid, deptid, partid,qty, values)

Using domain variables

ContractID – C

Supplierid – S

Projectid – J

Deptid- D

Partid – P

Qty- Q

Values- V

Schema contracts —>CSJDPQV

Following **IC**'s are known to hold:

The contractId C is a key: a C—>CSJDPQV

A project purchases a given part using a single contract JP—>C

A department purchases at most one part from a supplier SD—> P

Additional FD's hold in closure of set of FD's:

From JP—>C

Using transitivity: C—>CSJDPQV

Infer: JP—> CSJDPQV

From SD—>P

Using augmentation

Infer: SDJ—>JP

From SDJ—>JP

JP—>CSJDPQV

Using transitivity

Infer: SDJ—>CSJDPQV

Several additional FD's that can infer in the closure of using augmentation and decomposition

Example: C—>CSJDPQV

Using decomposition:

Infer C->C, C->S, C->J, C->D, C->P, C->Q, C->V.

Finally using reflexivity rule we have number of trivial FD's.

Attribute closure:

If we want to check whether a given functional dependency $X \rightarrow Y$ is in the closure of the set of FD's, we can do so efficiently without computing F^+. Instead of it first compute the attribute closure X^+ with respect to F.

Closure =X;

Repeat until there is no change :

{

If there is an FD U→V in F such that U ⊆ *Closure*,

then set *Closure* = *Closure* U V

}

Example 1:

Given a Relational schema R(A, B, C, G, H, I) and the set of FDs are:

{A->B, A->C, CG->H, CG->I, B->H} Deduce the F+ for the given relation

A → H. Since A → B and B → H hold, we apply the transitivity rule.

CG → HI . Since CG → H and CG → I , the union rule implies that CG → HI.

AG → I .Since A → C and CG → I , the pseudo-transitivity rule implies that AG → I holds.

F+ = { A->B, A->C, CG->H, CG->I, B->H, A->H, CG->HI, AG->I}

Example 2:

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

FD: A->D
D->B
B->C
E->B
If any attribute is not present on the right side can be considered as closure.
(A)+ =ADBC
(E)+ = EBC
(AE)+ = EDBC

Example 3:

R(A,B,C,D,E,F,G,H)

FD ={ CH->G A->BC B->CFH E->A F->EG}
(D) + = D

(DA)+=DABCFHEG (CK)

(DB)+=DBCFHEGA (CK)

(DC)+= DCG

(DE)+=DEABCFHEG (CK)

(DF)+=DFEGABCH (CK)

Minimal Cover for a set of FDs/ Canonical Form:

1. Every dependency is as small as possible; that is, each attribute on the left side of FD is necessary and on the right side is a single attribute.
2. Every dependency in it is required for the closure to be equal to F^+ (is closure set)

To identify Canonical cover for the given set, follow the steps:

a. Splitting Rule: For every FD given, on the right side of FD should have a single attribute.

b. Remove extraneous attributes
c. Remove redundant FD(using closure set)

Example 1: Consider R(P,Q,R) with following FDs

F={ P->QR, Q->R,P->Q,PQ->R}
Step 1: P->QR(split into P->Q,P->R)
F={P->Q,P->R,Q->R,P->Q,PQ->R}
Step 2: F={P->Q,P->R,Q->R,P->Q}
Step 3: F={P->Q,P->R,Q->R}
P->Q
find the closure for P+={PR} no Q is there in closure set so don't discard P->Q
P->R P+={PQR} discard P->R
Q->R Q+={Q} don't discard Q->R
F'={P->Q, Q->R}

Example 2: Consider R(A,B,C,D) with following FDs

F={ A->B, C->D, AB->C}

Step 1: check for single attribute on the right side

Step 2: Remove extraneous attributes

F={ A->B, it is clear that from A we can determine B, from AB->C, discard B and write A->C}

Step 3: Remove redundant FDs
A->B closure of A+={AC} don't discard A->B

C->D closure of C+={C} don't discard C->D

A->C closure of A+={AB} don't discard A->C

Irreducible set F'={A->B,C->D,A->C}

Example 3: Semester End Question(2023-

24, Even Sem)

Given the relational schema- R (ABCDEFGH) and the set of functional dependencies, F , F = {A→B, ABCD→E, EF→G, EF→H, ACDF→EG}. Find the minimal cover of F.

Solution to Example 3

We can find the minimal cover of F by following the 3 steps.

Step 1 : Splitting Rule: Split the FDs, such that RHS contains a single attribute. **(Example, if** A→BC, then A→B and A→C)

A→B,
ABCD→E
EF→G
EF→H
ACDF→E
ACDF→G

Step 2 : R**emove all redundant FDs and delete them from the set of FDs.**

First exclude A→B and take A+ using the other FDs, A+={A}, since A+ does not contain B, so A→B is not redundant.

Next exclude ABCD→E
ABCD+ = ABCD
Since E is not a part of closure of ABCD, so ABCD→E is not redundant

Next exclude EF→G
EF+ = EFH
Since G is not a part of closure of EF, so EF→G is not redundant

Next exclude EF→H
EF+ = EFG
Since H is not a part of closure of EF, so EF→H is not redundant

Next exclude ACDF→E
ACDF+ = ACDFBEGH
Since E is a part of closure of ACDF, so ACDF→E is redundant, so remove ACDF→E from the set F.

Next exclude ACDF→G
ACDF+ = ACDFEHGB
Since G is a part of closure of ACDF, so ACDF→G is redundant, so remove ACDF→G from the set F.

After Step 2, the final FDs are

A→B,
ABCD→E
EF→G
EF→H

Step: 3 Find the Extraneous/Redundant attributes on LHS and delete them .

(Example: if AB→C, A can be deleted if B+ contains A)

Final FDs after step3 are:

Minimal cover of F :

A→B
ACD→E
EF→G
EF→H

NORMALIZATION:

It is a process of analyzing given relation schema based on their FD and Primary Keys to achieve desired properties of minimizing redundancy and minimizing the insertion, deletion and Update anomalies.

Need for Normalization:

For minimizing data redundancy ie., no unnecessarily duplication of data.

To Make DB Structure flexible ie. It should be possible to add new data values and rows without reorganizing the DB Structure.

Data should be consistent throughout the DB ie, it should not suffer from following anomalies.

a) Insertion Anomaly
b) Update Anomaly
c) Delete Anomaly
d) Redundant Storage

Advantages or Merits of Normalization:

- More Efficient data structure.
- Avoid redundant fields
- Minimizes Data Duplication

Disadvantages or Demerits of Normalization:

- We cannot develop the DB unless knowing other user needs.
- On Normalizing the relations to higher normal forms, ie.,4NF, 5NF the performance degrades.
- It is very time consuming and difficult process in normalizing relations of high degree.
- Careless decomposition may leads to bad design of database which may leads to serious problems.

NORMAL FORMS: It is a state of relation that result by decomposition of that relation for a good design to avoid redundancy and inconsistency. The Relation is said to be in a particular Normal Form if it satisfies certain set of conditions.

There are different Forms. They are:

a) 1st NORMAL FORM (1NF)
b) 2nd NORMAL FORM (2NF)
c) 3rd NORMAL FORM (3NF)
d) BCNF – BOYCEE CODD NORMAL FORM
e) 4th NORMAL FORM (4NF)
f) 5th NORMAL FORM (5NF)

1st NORMAL FORM (1NF) :

In I NF, the values of each attribute should be atomic and no composite values and all entries in any column must be of the same kind. Each column must have a unique name, No two rows are identical.

Example:

Students

Roll Number	Name	Courses
101	X	DBMS, CN, SE
102	Y	CO, OS
103	Z	CD, CN

As per the I NF, the values should maintain atomic and unique. For satisfying the 1NF Criteria have to represent in following way.

Students:

Roll Number	Name	Courses
101	X	DBMS
101	X	CN
101	X	SE
102	Y	CO
102	Y	OS
103	Z	CD
103	Z	CN

Now the Relation is satisfied as 1NF Criteria.

2nd NORMAL FORM:

IN 2NF , first it should satisfy the 1NF Criteria and All Non- Prime Attributes are should be Fully Functional Dependency on any key of Relation R. Ie., it should not Partial Functional Dependency .

Example:

R(A,B,C,D,E,F)

F ={ A-> BCDEF

BC->ADEF

B->F

D->E}

Step 1: First find the Prime Attribute and Non Prime Attributes.

For Finding the Prime and Non Prime Attribute have to perform the Attribute Closure operation on LHS or which is not present in RHS.

(A)+={ABCDEF}
(BC)+ ={BCADEF}
(B)+={BF}
(D) +={DE}

Candidate Keys (A, BC)
So, Now check Prime Attributes and Non-Prime Attributes.

PRIME ATTRIBUTE ={A, B, C}
NON-PRIME ATTRIBUTE = { D, E, F}

Step 2: Check whether all Non Prime Attributes are Partial FD or Fully FD on candidate Key.

Case(i) :- If Non-Prime Attributes are partial FD. The Relation R is considered as not in 2NF form.

Case(ii): - If Non-Prime Attribute are Fully Functional Dependency, then Relation R is considered as in 2 NF Form.

Now check the NON-Prime Attributes {D, E, F} are Full Functional Dependency or not:

D Fully FD on A and BC
E Fully FD on A and BC as well as on D, But D is not a candidate Key.
F Fully FD on A and BC, but again F is determined by B.

Ie., BC->F
B->F

Partial FD on Candidate Key on BC .This F is not considered, because it is not suitable for 2NF Criteria.
For making this relation to be 2NF, this B->F. Now this B->F is decomposed into two Relations

R1(A B C D E) R2(B F)

As per 2NF Criteria now, R1 is satisfied.

3rd NORMAL FORM:

In 3NF, the Relation R be in 2NF and No Non Prime Attribute should be Transitively dependent on candidate key. There should not be the case that a non prime attribute is determined by another non prime attribute.

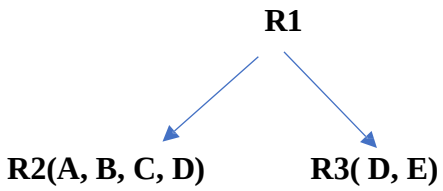
Example: Consider R1(A B C D E)

FD = { A->BCDE
BC->ADE
D->E}

Candidate Key –(A, BC)
Non Prime Attribute – (D, E)

Here, you having FD of D->E ie., now you have to remove D->E FD because both are NPA.

According to 3NF, No Prime Attribute should not be determined. For removing , decomposed the Table into Two.



For D determining E, As per 3NF Criteria R2 is in 3NF.

BCNF – BOYCEE CODD NORNAL FORM:

In BCNF and it should be in 3NF. For any dependency A->B, A should be a super Key, which means for A->B, If A is Non-Prime Attribute and B is a Prime Attribute.

1) Prime Attribute -> Non-Prime Attribute (Functional Dependency)
2) Part of Primary Key -> Non-Prime Attribute (Partial Dependency)
3) Non – Prime Attribute -> Non-Prime Attribute (Transitive Dependency)
4) Non-Prime attribute-> Prime Attribute – This is the 4th Case is there it simply means not in BCNF.

Example:
R(A, B, C, D)

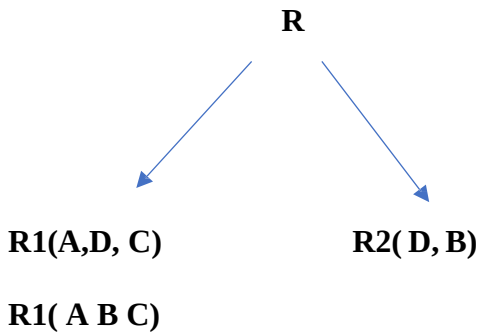
A->BCD
BC-> AD
D->B

Consider this FD Key is { A, BC}
i) A->BCD

ii) BC->AD
iii) D->B

Here, A B C are super key and D is not super key ie., it is not present in BCNF.

To make this relation to BCNF. We must decompose Table.



Here, R is considered as two Relations i) R1 (A, D, C)
ii) R2(A, B, C)

Therefore, B is a Prime Attribute and D is Non-Prime Attribute so from the above Relation R1 , B cannot be dependent on D. i.e. D -> B cannot be hold.
So, we consider as R1(A, B, C). Now this Relation is in BCNF.

4th NORMAL FORM:

A Relation should satisfy 3NF and BCNF. It should not contain any multivalued Dependency. 4 NF is a level of database normalization where there are non-trivial multivalued dependencies other than a candidate key.

Multivalued Dependency:

When we declared that the relation in multivalued dependency according to criteria below 3 conditions should satisfy.

1) A Relation R Should contain 3 columns/ attributes A, B, C or more.
2) A -> -> B, For a single value of A , more than one value of B exist.
3) For the Relation with A, B, C Columns B and C should be independent.

Example:

R(A,B,C)

A->->B – Multivalued Dependency between A and B

A->->C – Multivalued Dependency between A and C

Example:

Course	Instructor	Text book
DBMS	**X**	**Raghu Ramakrishnan**
	Y	**Korth**
	Z	
OS	**W**	**Galvin**
	V	**Abraham Tanenbaum**

Course -> -> Instructor

Course -> -> Text Book

There is no relation between Instructor and Text Book. So, this Relation is in Multivalued Dependency.

To Satisfy 4NF , Decomposed into 2 Relations

Course -> -> Instructor - Relation 1

Course -> -> Text Book -Relation 2

Course -> -> Instructor - Relation 1

Course	Instructor
DBMS	X
DBMS	Y
DBMS	Z
OS	W
OS	V

Course -> -> Text Book -Relation 2

Course	Text Book
DBMS	Raghu Ramakrishnan
DBMS	Korth
OS	Galvin
OS	Abraham Tanenbaum

5th NORMAL FORM:

It is denoted by JD(R1, R2, R3 -------- Rn) specified on Relation Schema R specifies a constraint on the states r of R. Every Legal state r of R should have a non-addictive Join Decomposition in R1, R2------Rn.

Example:

Check Given Relation is Join Dependency or not.

Consider a Relation R

Agent	Company	Product
A	C1	Pen drive
A	C1	CD
A	C2	Speaker
M	C1	Speaker

R1

Agent	Company
A	C1
A	C2
M	C1

R2

Agent	Product
A	PENDRIVE
A	CD
A	SPEAKER
M	SPEAKER

R3

Company	Product
C1	PENDRIVE
C1	CD
C2	SPEAKER
C1	SPEAKER

Therefore, (R1 JOIN R2) JOIN R3

Agent	Company	Product
A	C1	PENDRIVE
A	C1	CD
A	C1	SPEAKER
A	C2	PENDRIVE
A	C2	CD
A	C2	SPEAKER
M	C1	SPEAKER

R1 JOIN R2 JOIN R3 =! R , Hence, It is not in JOIN DEPENDENCY

5th NF Criteria :

1) R must be in 4NF
2) If Join Dependency(JD) not exist , then it will be in 5NF, else If JD exists, then check whether JD is trivial JD or not.
3) It cannot do decomposition is in 5NF or not.

Different Types of Dependencies: -

FUNCTIONAL DEPENDENCY:
Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS).A functional dependency (FD) has the form X -> Y (read X functionally determines Y) where X and Y are sets of attributes in a relation R

X -> Y if and only if: for any instance r of R

For any tuples t1 and t2 of r t1(X) = t2(X) implies t1(Y)=t2(Y)

The Functional Dependencies will identify relation between attributes within a relation. This facilitate to decompose for removing redundancies.

TRIVIAL FUNCTIONAL DEPENDENCY : In Trivial FD a dependent is always subset of the determinant ie., **if X->Y and Y is the subset of X then it is called Trivial Functional Dependency.**

Example:
R (A, B, C)
AB-> B then it is a Trivial Functional Dependency
A->A called as Trivial Functional Dependency.
{StudentID,Name}→StudentID is a trivial functional dependency.

NON -TRIVIAL FUNCTIONAL DEPENDENCY : In Non Trivial Functional Dependency the dependent is strictly not a subset of determinant. i.e, if X->Y and Y is not a subset of X then it is call Trivial Functional Dependency.

Example:
R (A, B, C)
AB-> C then it is a NON-Trivial Functional Dependency, i.e, C is not a subset of AB.
StudentID→Name is non-trivial if Name is not part of StudentID

MULTIVALUED FUNCTIONAL DEPEDENCY
Definition: A multivalued dependency X→→Y exists if, for a single value of X, there is a set of values for Y that are independent of other attributes.

Example: Consider a table with StudentID, Language, and Hobby. If a student can have multiple languages and multiple hobbies, then StudentID →→ Language and StudentID →→ Hobby represent multivalued dependencies.

Impact: Multivalued dependencies lead to anomalies that 4NF addresses.

TRANSISTIVITY FUNCTIONAL DEPENDENCY

Definition: A transitive dependency occurs when a non-prime attribute depends on another non-prime attribute, which in turn depends on a candidate key.

Example: If A → B and B → C, then A → C is a transitive dependency.

Impact: Transitive dependencies are problematic in 3NF normalization.

FULLY FUNCTIONAL DEPENDENCY

Definition: A functional dependency X→Y is fully functional if Y is functionally dependent on the entire set of attributes in X and not on any proper subset of X.

Example: In a table with attributes StudentID, CourseID, and Grade, the dependency StudentID, CourseID → Grade is fully functional if Grade depends on both StudentID and CourseID together, not just one of them.

PARTIAL DEPENDENCY

Definition: A partial dependency occurs when a non-prime attribute is functionally dependent on a part of a candidate key (rather than the entire candidate key).

Example: In a table with a composite key StudentID, CourseID, if StudentName depends only on StudentID (not on both StudentID and CourseID), this is a partial dependency.

Impact: Partial dependencies are problematic in 2NF normalization.

JOIN DEPENDENCY

Definition: A join dependency occurs when a relation can be decomposed into two or more relations that can be joined back together without loss of information.

Example: If a relation R can be decomposed into R1, R2 and R3 such that R=R1⋈R2⋈R3 and this holds without losing information, then there is a join dependency.

Impact: Join dependencies are addressed by 5NF.

2023-24: Even Sem End Question.

Consider an IT services Agency, which tests products of various companies. Three tables exist: Q) Agency, Product, Company. Company can offer one or more products, Agencies can test one or more products from one or more companies. A product can be tested in one or more agencies and can be offered by one or more companies. If an IT services Agency is equipped to analyze a product from a given company, this information can be recorded using a relation schema: Agency_Product Company (Agency_Id, Product_Id, Company Id).

What type of dependencies the relation schema, Agency_Product Company has?

Is this relation in BCNF? What normal form the relation is in? Give reasons. What better structure of relation schemas do you suggest? What advantages/disadvantages do you see with the decomposition you suggest?

Solution:
Let us consider

Agency_id to be as represented as aid
Product_ id to be represented as pid
Company_ id to be represented as cid

As per the given question ,a sample instance of the relation Agency_Product_Company can be as follows:

aid	pid	cid
A1	P1	C1
A1	P2	C1
A1	P1	C2
A1	P2	C2
A2	P1	C1
A2	P2	C2

Agency_Product_Company

As per the above instance, it is seen that there are multivalued dependencies existing.
aid ->->pid , aid ->->cid , pid ->->aid, pid ->->cid, cid ->->aid, cid ->->pid.

- Identifying the Functional Dependencies(FDs) of Agency_Product_Company relation:

{aid} →{pid,cid}

{pid} →{aid,cid}

{cid} →{aid,pid}

- 3 Candidate Keys(CK) for Agency_Product_Company relation has been identified:

CK1: aid
CK2: pid
CK3: cid

- **To determine if the relation is in BCNF (Boyce-Codd Normal Form), we need to check if it satisfies the condition that for every non-trivial functional dependency X → Y, X must be a superkey. In this case, the relation is in BCNF since all the functional dependencies listed above have keys on their left-hand side.**
  The relation is in BCNF because every determinant (aid, pid, and cid) is a candidate key.

- **A better structure of relation schemas could involve decomposing the current relation into 3 relations to eliminate redundancy**

Relation 1

a	p

Relation 2

a	c

Relation 3

p	c

- **Advantages of this decomposition:**

- Reduced redundancy: Each relation focuses on a specific aspect of the data, reducing the chance of redundant information.
- Simplified maintenance: With smaller, more focused relations, it becomes easier to update and maintain the database.
- Improved clarity: The decomposition separates concerns, making it clearer to understand the relationships between agencies, products, and companies.