

UNIT-IV

SCHEMA REFINEMENT AND NORMALISATION

Unit 4 contents at a glance:

1. Introduction to schema refinement,
2. functional dependencies,
3. reasoning about FDs.
4. Normal forms: 1NF, 2NF, 3NF, BCNF,
5. properties of decompositions,
6. normalization,

Schema Refinement:

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is **decomposition**.

Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

Anomalies or problems facing without normalization(problems due to redundancy) :

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

1.insertion anomalies : It may not be possible to store some information unless some other information is stored as well.

2.redundant storage: some information is stored repeatedly

3.update anomalies: If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.

4.deletion anomalies: It may not be possible to delete some information without losing some other information as well.

Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

7k > 7k
Costly Operation

More IO Cost

Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

Insertion Anomalies: New course is introduced C4, But no student is there who is having C4 subject.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

NULL	NULL	CA	DB	12k
------	------	----	----	-----

To Insert that Row, It is Required to Put Dummy Data..

Therefore,

xx	xx	CA	DB	12k
----	----	----	----	-----

Because of insertion of some data, It is forced to insert some other dummy data.

Deletion Anomaly :

Deletion of S3 student cause the deletion of course.

Because of deletion of some data forced to delete some other useful data.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

Solutions To Anomalies : Decomposition of Tables – Schema Refinement

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
S3	B	C2
S3	B	C2

PK(SID,CID)

Deletion Anamoly
Removed

CID	CNAME	FEE
C1	C	5k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

7k (Updation Anamoly
Removed

Insertion Anamoly
Removed

There are some Anomalies in this again –

SID	Sname	CID
S1	A (AA)	C1
S2	A (AA)	C1
S1	A (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Update Anomaly

Deletion Anomaly as C2 course is allotted to some students

CID	CNAME	FEE
C1	C	5k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

A student having no course is enrolled. We have to put dummy data again.

What is the Solution ??

Solution : decomposing into relations as shown below

R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

→ **TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement technique called **DECOMPOSITION**.

Decomposition:- Process of decomposing a larger relation into smaller relations.

→ Each of smaller relations contain subset of attributes of original relation.

Functional dependencies:

→ Functional dependency is a relationship that exist when one attribute uniquely determines another attribute.

→ Functional dependency is a form of integrity constraint that can identify schema with redundant storage problems and to suggest refinement.

→ A functional dependency $A \rightarrow B$ in a relation holds true if two tuples having the same value of attribute A also have the same value of attribute B

IF $t1.X=t2.X$ then $t1.Y=t2.Y$ where $t1, t2$ are tuples and X, Y are attributes.

Reasoning about functional dependencies:**Armstrong Axioms :**

Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:

Primary axioms:

Rule 1	Reflexivity If A is a set of attributes and B is a subset of A, then A holds B. $\{A \rightarrow B\}$
Rule 2	Augmentation If A hold B and C is a set of attributes, then AC holds BC. $\{AC \rightarrow BC\}$ It means that attribute in dependencies does not change the basic dependencies.
Rule 3	Transitivity If A holds B and B holds C, then A holds C. If $\{A \rightarrow B\}$ and $\{B \rightarrow C\}$, then $\{A \rightarrow C\}$ A holds B $\{A \rightarrow B\}$ means that A functionally determines B.

secondary or derived axioms:

Rule 1	Union If A holds B and A holds C, then A holds BC. If $\{A \rightarrow B\}$ and $\{A \rightarrow C\}$, then $\{A \rightarrow BC\}$
Rule 2	Decomposition If A holds BC and A holds B, then A holds C. If $\{A \rightarrow BC\}$ and $\{A \rightarrow B\}$, then $\{A \rightarrow C\}$
Rule 3	Pseudo Transitivity If A holds B and BC holds D, then AC holds D. If $\{A \rightarrow B\}$ and $\{BC \rightarrow D\}$, then $\{AC \rightarrow D\}$

Attribute closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

NOTE:

To find attribute closure of an attribute set-

1)add elements of attribute set to the result set.

2)recursively add elements to the result set which can be functionally determined from the elements of result set.

Algorithm : Determining X^+ , the closure of X under F.

Input : Let F be a set of FDs for relation R.

Steps:

```

1.  $X^+ = X$  //initialize  $X^+$  to X
2. For each FD :  $Y \rightarrow Z$  in F Do
    If  $Y \subseteq X^+$  Then //If Y is contained in  $X^+$ 
         $X^+ = X^+ \cup Z$  //add Z to  $X^+$ 
    End If
End For
3. Return  $X^+$  //Return closure of X

```

Output : Closure X^+ of X under F

Types of functional dependencies:

1) Fully Functional dependency: If $X \rightarrow Y$ is a function dependency if Y is not determined by any subset of X, this type of FD's called as Fully Functional Dependency.

2) Partially Functional Dependency: If $X \rightarrow Y$ is functional dependency if Y is determined by any subset of X, these type of FD's called as Partially functional dependency.

3) **Trivial functional dependency**:-If $X \rightarrow Y$ is a functional dependency where $Y \subseteq X$, these type of FD's called as trivial functional dependency.

4) **Non-trivial functional dependency**:-If $X \rightarrow Y$ and Y is not subset of X then it is called non-trivial functional dependency.

5) **Completely non-trivial functional dependency**:-If $X \rightarrow Y$ and $X \cap Y = \Phi$ (null) then it is called completely non-trivial functional dependency.

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

Candidate Key:

Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.

Consider student table: student(sno, sname, sphone, age)

we can take **sno** as candidate key. we can have more than 1 candidate key in a table. types of candidate keys:

1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

Super Key:

Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(sno, sname, sphone, age) we can take sno, (sno, sname)

as super key

Finding candidate keys problems:**Example 1:** Find candidate keys for the relation R(ABCD) having following FD's $AB \rightarrow CD, C \rightarrow A, D \rightarrow A$ **Solution:** $AB^+ = \{ABCD\}$ A and B are prime attributes $C \rightarrow A$ replace A by c $BC^+ = \{ABCD\}$ A and C are prime attributes ($A^+ = A^+ = \{AC\}$) $D \rightarrow A$ replace A by D $AD^+ = \{ABCD\}$ A and D are prime attributes ($D^+ = \{BD\}$) $CD^+ = \{ABCD\}$ (replacing A by C in AD)

AB, BC, CD, AD are candidate keys.

Example 2: Find candidate keys for R(ABCDE) having following FD's $A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A$ **Solution:** $A^+ = \{ABCDE\}$ A is candidate key and prime attribute $E \rightarrow A$ so replace A by E $E^+ = \{ABCDE\}$ E is candidate key and prime attribute $CD \rightarrow E$ replace E by CD $CD^+ = \{ABCDE\}$ ($C^+ = C$ and $D^+ = D$) no proper subset of CD is superkey. so CD is candidate key $B \rightarrow D$ $BC^+ = \{ABCDE\}$ ($B^+ = BD$) BC is candidate key

A, E, CD, BC are candidate keys

Question 1 : Given a relation R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F} Identify the prime attributes and non prime attributes .

Solution :

$(AB)^+ : \{ABCDEF\} \Rightarrow$ Super Key
 $(A)^+ : \{A\} \Rightarrow$ Not Super Key
 $(B)^+ : \{B\} \Rightarrow$ Not Super Key
 Prime Attributes : {A,B}
 $(AB) \rightarrow$ Candidate Key
 ↓ (as $F \rightarrow B$)
 $(AF)^+ : \{AFBCDE\}$
 $(A)^+ : \{A\} \Rightarrow$ Not Super key
 $(F)^+ : \{FB\} \Rightarrow$ Not Super Key
 $(AF) \rightarrow$ Candidate Key
 ↓
 $(AE)^+ : \{AEFBCD\}$
 $(A)^+ : \{A\} \Rightarrow$ Not Super key
 $(E)^+ : \{EFB\} \Rightarrow$ Not Super key
 $(AE) \rightarrow$ Candidate Key
 ↓
 $(AD)^+ : \{ADEFBBC\}$
 $(A)^+ : \{A\} \Rightarrow$ Not Super key
 $(D)^+ : \{DEFB\} \Rightarrow$ Not Super key
 $(AD) \rightarrow$ Candidate Key
 ↓
 $(AC)^+ : \{ACDEFB\}$
 $(A)^+ : \{A\} \Rightarrow$ Not Super Key
 $(C)^+ : \{DCEFB\} \Rightarrow$ Not Super Key
 \Rightarrow Candidate Keys {AB, AF, AE, AD, AC}
 \Rightarrow Prime Attributes {A,B,C,D,E,F}
 \Rightarrow Non Prime Attributes {}

Question 2: Given a relation R(ABCDEF) having FDs {AB → C, C → DE, E → F, C → B} Identify the prime attributes and non prime attributes.

Solution :

$(AB)^+ : \{A B C D E F\}$
 $(A)^+ : \{A\}$
 $(B)^+ : \{B\}$
 $(AB) \Rightarrow (AC), (AC)^+ : \{ABCDEF\}$
 $(C)^+ : \{DECBF\}$
 \Rightarrow Candidate Keys {AB, AC}
 \Rightarrow Prime Attributes {A,B,C}
 \Rightarrow Non Prime Attributes {D,E,F}

Normalization:

Normalization is a process of designing a consistent database with minimum redundancy which support data integrity by grating or decomposing given relation into smaller relations preserving constraints on the relation.

→ Normalisation removes data redundancy and it will helps in designing a good data base which involves a set of normal forms as follows -

1) First normal form (1NF)

2) Second normal

form (2NF) 3) Third normal

form (3NF)

4) Boyce coded normal

form (BCNF) 5) Forth normal

form (4NF)

6) Fifth normal form (5NF)

Normal Forms

1 st Normal Form	No repeating data groups
2 nd Normal Form	No partial key dependency
3 rd Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4 th Normal Form	No multi-valued dependency
5 th Normal Form	No join dependency

$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$

Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	Most	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	Maybe	Maybe
Preserves MVD's	Maybe	Maybe	Maybe

Properties of normal forms and their decompositions

1 First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

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

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

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

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

2 Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they

teach. In a school, a teacher can teach more than one subject.

TEACHER table

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

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

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

3 Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

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

Super key in the table above:

1.

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007

555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

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

4 Boyce Codd normal form (BCNF)

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

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

EMPLOYEE table:

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

In the above table Functional dependencies are as follows:

1. $EMP_ID \rightarrow EMP_COUNTRY$

2. EMP_DEPT \rightarrow {DEPT_TYPE, EMP_DEPT_NO}

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

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

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

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID \rightarrow EMP_COUNTRY

2. EMP_DEPT \rightarrow {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

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

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

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

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry

74	Biology
59	Physics

STUDENT_HOBBY

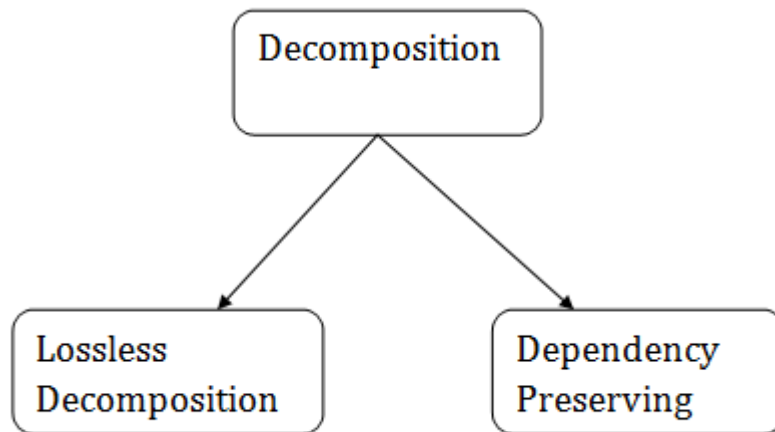
STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Note: In some cases multi value dependencies may exist not more than one time in a given relation.

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper 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.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees 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	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai

33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

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 look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

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

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

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

P2

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

P3

SEMSTER	LECTURER
Semester 1	Anshika

Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Algorithm fifth normal form: fifth normal form is related to join dependencies.

→A relation R is said to be in fifth normal form if for every join dependency JD join $\{R_1, R_2, \dots, R_N\}$ that holds over relation R one of the following statements must be

true- 1) $R_i = R$ for some i

2) the join dependency is implied by the set of those functional dependency over relation R in which the left side is key attribute for R.

NOTE: if the relation schema is a third normal form and each of its keys consist of single attribute, we can say that it can also be in fifth normal form.

→A join dependency JD join $\{R_1, R_2, \dots, R_N\}$ is said to hold for a relation R if R_1, R_2, \dots, R_N this decomposition is a loss less join decomposition of R.

→When a relation is in forth normal form and decompose further to eliminate redundancy and anomalies due to insert or update or delete operation, there should not be any loss of data or should not create a new record when the decompose tables are rejoin.

Key Points related to normal forms –

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

problems on normal forms:**Problem 1:**

Find the highest normal form in R (A, B, C, D, E) under following functional dependencies. $ABC \rightarrow D$
 $CD \rightarrow AE$

Solution:

Important Points for solving above type of question.

- 1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
 - 2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, $ABC \rightarrow D$ is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.
- Candidate keys in given relation are {ABC, BCD}

BCNF: $ABC \rightarrow D$ is in BCNF. Let us check $CD \rightarrow AE$, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: $ABC \rightarrow D$ we don't need to check for this dependency as it already satisfied BCNF. Let us consider $CD \rightarrow AE$. Since E is not a prime attribute, so relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2 NF. **So, the highest normal form is 1 NF.**

problem 2:

Find the highest normal form of a relation

R(A,B,C,D,E) with FD set as

{BC-
 $\rightarrow D$,
 AC-
 $\rightarrow BE$,
 $B \rightarrow E$ }

Step 1: As we can see, $(AC)^+ = \{A, C, B, E, D\}$ but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2: Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because $BC \rightarrow D$ is in 2nd normal form (BC is not proper subset of candidate key AC) and $AC \rightarrow BE$ is in 2nd normal form (AC is candidate key) and $B \rightarrow E$ is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in $BC \rightarrow D$ (neither BC is a super key nor D is a prime attribute) and in $B \rightarrow E$ (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal form, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

Decomposition: It is the process of splitting original table into smaller relations such that attribute sets of two relations will be the subset of attribute set of original table.

Rules of decomposition:

If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1) Union of two smaller subsets of attributes gives all attributes of 'R'.

$$R1(\text{attributes}) \cup R2(\text{attributes}) = R(\text{attributes})$$

2) Both relations interaction should not give null value.

$$R1(\text{attributes}) \cap R2(\text{attributes}) \neq \text{null}$$

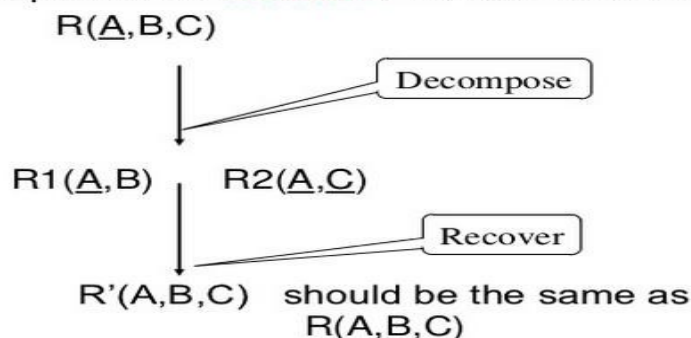
3) Both relations interaction should give key

$$\text{attribute. } R1(\text{attribute}) \cap R2(\text{attribute}) = R(\text{key attribute})$$

Properties of decomposition:

Lossless decomposition: while joining two smaller tables no data should be lost and should satisfy all the rules of decomposition. No additional data should be generated on natural join of decomposed tables.

A decomposition is *lossless* if we can recover:



----- Must ensure $R' = R$ -----

Lossless Decomposition example

- Sometimes the same set of data is reproduced:

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100


Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)

example 2 for loseless decomposition:

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8




A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	C
1	3
4	6
7	8



B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8


But, now we can't check $A \rightarrow B$ without doing a join!

Lossy join decomposition: if information is lost after joining and if do not satisfy any one of the above rules of decomposition.

example 1:

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

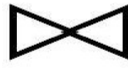


A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	B
1	2
4	5
7	2

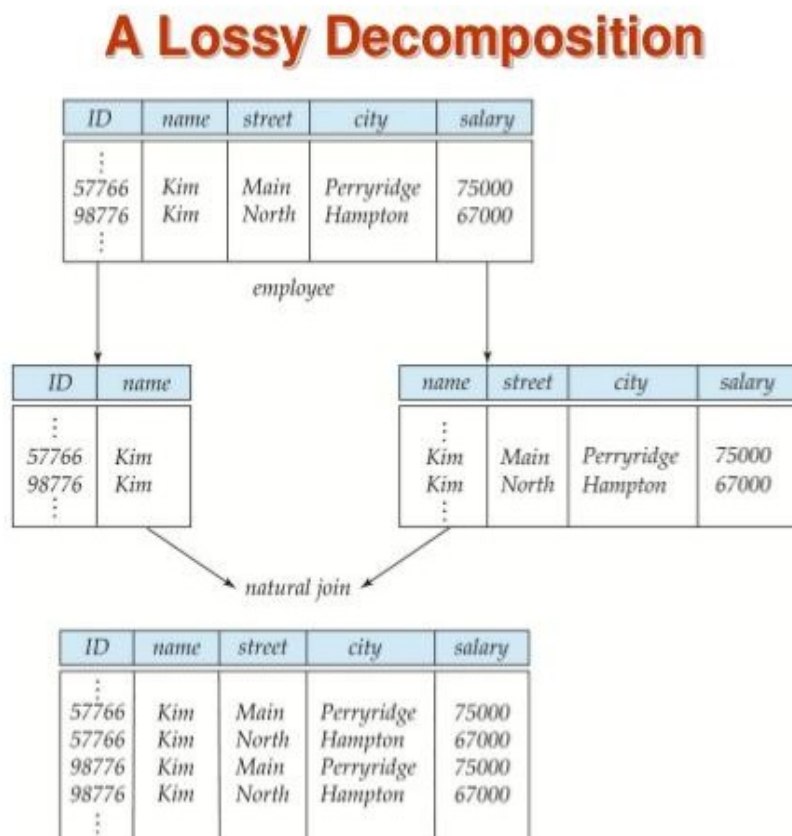


B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

example 2:



8

In above examples, on joining decomposed tables, extra tuples are generated. so it is lossy join decomposition.

Dependency preservation: functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables.

Dependency Preservation

A Decomposition $D = \{ R_1, R_2, R_3, \dots, R_n \}$ of R is dependency preserving wrt a set F of Functional dependency if

$$(F_1 \cup F_2 \cup \dots \cup F_m)^+ = F^+.$$

Consider a relation R

$R \rightarrow F \{ \dots \text{with some functional dependency (FD)} \}.$

R is decomposed or divided into R_1 with FD $\{ f_1 \}$ and R_2 with $\{ f_2 \}$, then there can be three cases:

$f_1 \cup f_2 = F \rightarrow$ Decomposition is dependency preserving.

$f_1 \cup f_2$ is a subset of $F \rightarrow$ Not Dependency preserving.

$f_1 \cup f_2$ is a super set of $F \rightarrow$ This case is not possible.

example for dependency preservation:**Dependency preservation****Example:**

$R=(A, B, C), F=\{A \rightarrow B, B \rightarrow C\}$

Decomposition of R: $R_1=(A, B)$ $R_2=(B, C)$

Does this decomposition preserve the given dependencies?

Solution:

In R_1 the following dependencies hold: $F_1=\{A \rightarrow B, A \rightarrow A, B \rightarrow B, AB \rightarrow AB\}$

In R_2 the following dependencies hold: $F_2=\{B \rightarrow B, C \rightarrow C, B \rightarrow C, BC \rightarrow BC\}$

$F' = F_1' \cup F_2' = \{A \rightarrow B, B \rightarrow C, \text{trivial dependencies}\}$

In F' all the original dependencies occur, so this decomposition preserves dependencies.

lack of redundancy: It is also known as repetition of information. The proper decomposition should not suffer from any data redundancy.