UNIT-IV

Schema Refinement:
Introduction to Schema Refinement, Properties of Decomposition, Functional Dependencies, Reasoning about FD,
Normalization: Normal Forms – 1NF, 2NF, 3NF, BCNF, 4NF and 5NF.

INTRODUCTION TO SCHEMA REFINEMENT

Introduction

Schema Refinement is the process of improving a database schema to reduce redundancy, eliminate anomalies, and ensure data integrity. It is an important step in database design and is achieved mainly through Normalization.

A poorly designed schema may lead to data duplication, inconsistency, and difficulties in maintaining the database. Schema refinement helps organize data efficiently and improves the overall performance and reliability of the database.

Need for Schema Refinement

A database schema may suffer from the following problems:

1. Data Redundancy

The same information is stored repeatedly in multiple records.

Example

StudentID	StudentName	Course	Instructor
101	Arun	DBMS	Kumar
102	Ravi	DBMS	Kumar

Instructor name "Kumar" is repeated.

2. Update Anomaly

A change in one place must be made in several records.

Example

If Instructor "Kumar" changes to "Ramesh", all rows must be updated.

3. Insertion Anomaly

Certain data cannot be inserted unless other data is available.

Example

A new course cannot be entered until a student enrolls in it.

4. Deletion Anomaly

Deleting a record may result in loss of useful information.

Example

If the last student enrolled in DBMS is deleted, course information may also be lost.

Objectives of Schema Refinement

1. Reduce data redundancy.
2. Eliminate insertion, deletion, and update anomalies.
3. Improve data consistency.
4. Maintain data integrity.
5. Organize data efficiently.
6. Improve database performance.

PROPERTIES OF DECOMPOSITION

Decomposition is the process of dividing a relation (table) into two or more smaller relations to eliminate redundancy and anomalies while preserving the original information.

During normalization, large relations are decomposed into smaller relations. A good decomposition should satisfy certain properties to ensure that no information is lost and queries can still be answered correctly.

A decomposition is considered good if it satisfies the following properties:

1. Lossless Join Decomposition
2. Dependency Preservation

LOSS LESS JOIN DECOMPOSITION

Decomposition of a relation R into R1 and R2 is lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

$$R1 \cap R2 \rightarrow R1$$

OR

$$R1 \cap R2 \rightarrow R2$$

- ConsiderarelationR whichis decomposedinto subrelationsR_1 and R_2.
- Thisdecompositioniscalledlosslessjoindecompositionwhenwejoin$R_1$and$R_2$and if we get the same relation R that was decomposed.

Forlossless joindecomposition, wealways have:**$R_1 \bowtie R_2$**

Example 1: Consider the following relation R(A , B , C). Let this relationis decomposed into two sub relations R_1(A , B) and R_2(B , C)

R

A	B	C
1	2	1
2	5	3
3	3	3

decompose→

R_1

A	B
1	2
2	5
3	3

and

R_2

B	C
2	1
5	3
3	3

Now, let us check whether this decomposition is lossless or not. For lossless decomposition, we must have:**$R_1 \bowtie R_2 = R$.**Now, if we perform the natural join (⋈) of the sub relations R_1 and R_2 , we get

A	B	C
1	2	1
2	5	3
3	3	3

Thisrelationis sameastheoriginalrelation R.

Thus, we conclude that the above decomposition is lossless join decomposition. This is because the resultant relation after joining the sub relations is same as the decomposed relation. No extraneous tuples (rows) appear after joining of the sub-relations.

Loss of Functional Dependency

- Once tables are decomposed, certain functional dependencies might not be preserved, which can lead to the inability to enforce specific integrity constraints.
- Example: If you have the functional dependency **`A → B`** in the original table, but in the decomposed tables, there is no table with both **`A`** and **`B`**, this functional dependency can't be preserved.

Example: Let's consider a relation R with attributes A,B, and C and the following functional dependencies:

A → B

B → C

Now, suppose we decompose R into two relations:

R1(A,B) with FD A → B

R2(B,C) with FD B → C

In this case, the decomposition is dependency-preserving because all the functional dependencies of the original relation R can be found in the decomposed relations R1 and R2. We do not need to join R1 and R2 to enforce or check any of the functional dependencies.

Functional Dependencies (FD)

A functional dependency **`\( X \rightarrow Y \)`** between two sets of attributes X and Y in a relation R is defined as: if two tuples (rows) of R have the same value for attributes X, then they must also have the same values for attributes Y. In other words, the values of X determine the values of Y.

Consider a relation (table) Students:

sid	sname	zipcode	cityname	state
S001	Aarav	110001	New Delhi	Delhi
S002	Priyanka	400001	Mumbai	Maharashtra
S003	Rohit	700001	Kolkata	West Bengal
S004	Ananya	560001	Bengaluru	Karnataka
S005	Kartik	600001	Chennai	Tamil Nadu
S006	Lakshmi	500001	Hyderabad	Telangana
S007	Aditya	160017	Chandigarh	Chandigarh

1. sid functionally determines sname because for a given student ID, there's only one possible student name
2. zipcode functionally determines cityname, a specific zip code should determine a unique cityname
3. cityname functionally determines state, A city name could determine a state.

Mathematically, these functional dependencies can be represented as:

$sid \rightarrow sname$
$zipcode \rightarrow cityname$
$cityname \rightarrow state$

Reasoning About Functional Dependencies

1. Trivial Dependency

- If Y is a subset of X, then the dependencyX ->Yis trivial.
- For example, in {A, B} -> {A}, the dependency is trivial because A is part of {A, B}.

Example: For attributes A,B,C:

- **$A \rightarrow A$** is a trivial dependency because an attribute always determines itself.
- **$AB \rightarrow A$** is a trivial dependency because the combined attributes A and B always determine A as it's a subset.
- **$ABC \rightarrow AC$** is a trivial dependency for the same reason; the combined attributes A,B, and C always determine A and C.

Full Functional Dependency

An attribute functionally depends on a set of attributes, X, and does not functionally depend on any proper subset of X.

Example: Consider a relation StudentCourses that has the following attributes:

- **StudentID** (unique identifier for each student)
- **CourseID** (unique identifier for each course)
- **Instructor** (name of the instructor teaching the course)

The relation is used to keep track of which student is enrolled in which course and who the instructor for that course is.

Now, assume we have the following functional dependency: **(StudentID,CourseID) → Instructor**

This means that a combination of a specific student and a specific course will determine who the instructor is.

Thus, Instructor is fully functionally dependent on the combined attributes StudentID and CourseID

Transitive Dependency

If A -> B and B -> C, then A has a transitive dependency on C through B.
Example: Consider a relation Employees with the following attributes:

- EmployeeID (unique identifier for each employee)
- EmployeeName
- Department (department in which the employee works)
- DepartmentLocation (location of the department)

Now, let's consider the following functional dependencies:
EmployeeID → Department
Department → DepartmentLocation
From the above functional dependencies:
An EmployeeID determines the Department an employee works in.
A Department determines its DepartmentLocation.

However, the DepartmentLocation is also dependent on the EmployeeID through Department. This means the DepartmentLocation has a transitive dependency on EmployeeID via Department.

This kind of transitive dependency can lead to redundancy.

4. Closure

1. **- The closure of a set of attributes X with respect to a set of functional dependencies FD, denoted as X+, is the set of attributes that are functionally determined by X.**
   - For example, given FDs: {A -> B, B -> C}, the closure of {A}, denoted as A+, would be {A, B,

C}.1NF(FIRST NORMAL FORM)

A relation (table)is said tobein first normal form if and only if:

- Each table cell contains only atomic values(single value).
- Each record needs to be uniquely identified by the primary key.

1NF Example:

HTNO	FIRSTNAME	LASTNAME	MOBILE
501	Jhansi	Rani	9999988888 7777799999
502	Ajay	Kumar	8888888881 7897897897
503	Priya	Verma	9898989898

Theabovetableis not in 1NFbecause **501** and **502** is havingtwo valuesin mobilecolumn. If we add a new column as *alternative mobile number* to the above table, then for 503 *alternative mobile number* is NULL.Moreover, if a student has 'n' mobile numbers, then adding 'n' extra column is meaningless. It is better to add extra rows. If we add extra row for each 501 and 502 then the table looks like

HTNO	FIRSTNAME	LASTNAME	MOBILE
501	Jhansi	Rani	9999988888
501	Jhansi	Rani	7777799999
502	Ajay	Kumar	8888888881
502	Ajay	Kumar	7897897897
503	Priya	Verma	9898989898

But the above table violates primary key constraint. Therefore instead of adding either columns or rows, the best solution is to split the table into two tables as shown below. If we do as shown below, if a student having 'n' number of mobile numbers also can be added.

HTNO	FIRST NAME	LAST NAME
501	Jhansi	Rani
502	Ajay	Kumar
503	Priya	Verma

HTNO	MOBILE
501	9999988888
501	7777799999
502	8888888881
502	7897897897
503	9898989898

2. 2NF (SECOND NORMAL FORM)

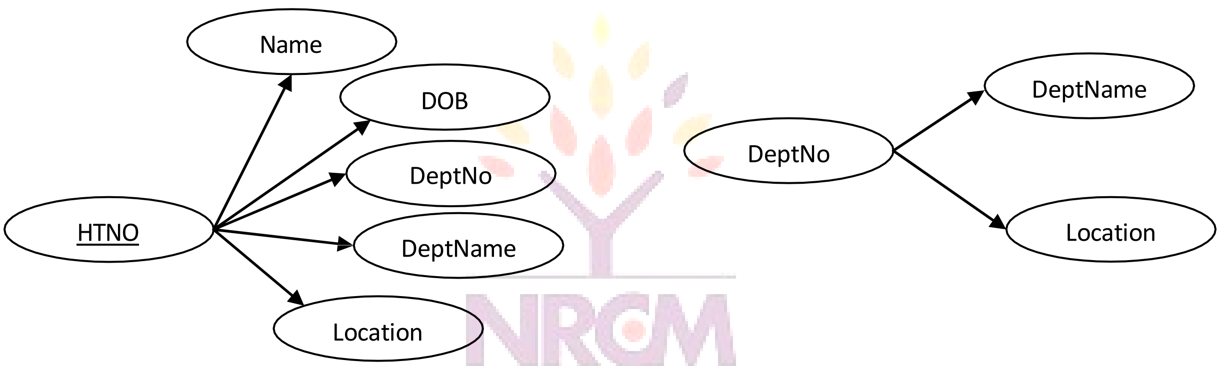
A relation is said to bein 2-NFifand only if

- It should be in 1-NF(First Normal Form)
- There should not be any partial functional dependencies

2NF Example:

HTNO	***Name***	***DOB***	***DeptNo***	***DeptName***	***Location***
501	Jhansi	30-10-1998	05	CSE	A-Block
502	Ajay	24-12-1999	05	CSE	A-Block
410	Priya	12-03-2000	04	ECE	B-Block
120	Rahul	30-10-1998	01	CIVIL	C-Block
415	Smitha	18-06-1999	04	ECE	B-Block

The above table is not in 2NF because there exist partial function dependencies. ***HTNO*** is a key attribute in the above table. If every non-key attribute fully dependent on key attribute, then we say it is fully functional dependent. Consider the below diagram. ***{Name, DOB, DeptNo, DeptName, Location}*** depends on ***HTNO***. But ***{DeptName, Location}*** also depends on ***DeptNo***.



It is clear that ***DeptName*** and ***Location*** not only depends upon ***HTNO*** but also on ***DeptNo***. So, there exists partial function dependency. This partial functional dependency can be removed by splitting the above table into two tables as follows.

HTNO	***Name***	***DOB***	***DeptNo***
501	Jhansi	30-10-1998	05
502	Ajay	24-12-1999	05
410	Priya	12-03-2000	04
120	Rahul	30-10-1998	01
415	Smitha	18-06-1999	04

DeptNo	**DeptName**	**Location**
05	CSE	A-Block
04	ECE	B-Block
01	CIVIL	C-Block

3. 3NF(THIRD NORMAL FORM)

A relation (table)is in third normal form if and only if it satisfies the following conditions:

- It is in second normal form

- There is no transitive functional dependency

Transitive functional dependency means, we have the following relationships in the table: A is functionallydependent on B (A→B), and Bis functionallydependent on C (B→C). In this case, C is transitively dependent on A via B (A→B and B→C mean A→B→C implies A→C).

3NFExample:

Consider the following book details table example:

BOOK_DETAILS

BookID	GenreID	GenreType	Price
1	1	Gardening	250.00
2	2	Sports	149.00
3	1	Gardening	100.00
4	3	Travel	160.00
5	2	Sports	320.00

The above table is not in 3NF because there exist transitive dependency.In the table able,

BookIDdetermines***GenreID*** ***{BookID→GenreID}***

GenreIDdetermines***GenreType***. ***{GenreID→GenreType}***

BookIDdetermines***GenreType***via***GenreID.*** {***BookID→GenreType***}

It implies that transitive functional dependency is existing and the structure does not satisfy third normal form.To bring this table into third normal form,we split the table into two as follows:

BOOK_DETAILS

BookID	GenreID	Price
1	1	250.00
2	2	149.00
3	1	100.00
4	3	160.00
5	2	320.00

GENRE_DETAILS

GenreID	GenreType
1	Gardening
2	Sports
3	Travel

4. *B*OYCE *C*ODD *N*ORMAL *F*ORM(BCNF)

A relation (table) is said to be in the BCNF if and only if it satisfy the following conditions:

- It should be in the**Third Normal Form**.
- For any functional dependencyA→B, A should be a**s uperkey**.

- In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

Example: Below we have a Patient table of a hospital.A patient can go to hospital many times to take treatment. On a single day many patients can take treatment.

PatientID	Name	EmailID	AdmittedDate	Drug	Quntity
101	Ram	ram@gmail.com	30/10/1998	A-10	10
102	Jhon	jho@gmail.com	30/10/1998	X-90	10
101	Ram	ram@gmail.com	10/06/2001	X-90	20
103	Sowmya	sam@gmail.com	05/03/2002	Y-30	15
102	Jhon	jho@gmail.com	05/03/2002	A-10	15

In the above table,{***PateintID,AdmittedDate***}acts as Primarykey.But if we know the ***EmailID***value, wecan find ***PatientID*** value.

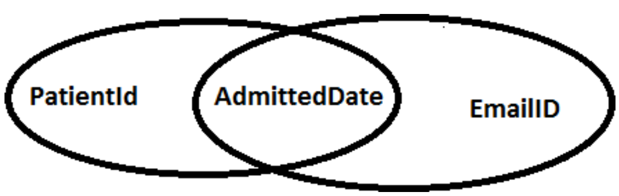
Thatis***EmailID → PatientID***.

In the above dependency, ***EmailId*** is non-prime attribute and ***PatientID*** is a primeattribute. Therefore the above table is not in BCNF. In order to bring the table into BCNF, we split it into two tables as shown below.

PatientID	***Name***	***AdmittedDate***	***Drug***	***Quntity***
101	Ram	30/10/1998	A-10	10
102	Jhon	30/10/1998	X-90	10
101	Ram	10/06/2001	X-90	20
103	Sowmya	05/03/2002	Y-30	15
102	Jhon	05/03/2002	A-10	15

PatientID	***EmailID***
101	ram@gmail.com
102	jho@gmail.com
103	sam@gmail.com

In other words we can also define BCNF as there should not be any overlapping between candidate keys. If you consider the original table (before splitting), we can get two candidate keys {***PateintID, AdmittedDate***} and{***EmailID, AdmittedDate***}.



As there exist overlapping in the candidate keys,the table is not in BCNF. To bring it into BCNF, we split into two tables as shown above.

5. 4-NF (FOURTH NORMAL FORM)

A relation is said to be in 4-NF if and only if it satisfies the following conditions

- It should be in the **ThirdNormalForm**.
- The table should not have any**Multi-valuedDependency**.

WhatisMulti-valuedDependency?

A table is said to have multi-valued dependency, if the following three conditions are true.

i. A table should have at-least 3 columns for it to have a multi-valued dependency.

ii. For any dependency***A* →*B***,if there exists multiple value of ***B*** for a single value of ***A***, then the table may have multi-valued dependency. It is represented as ***A* →→ *B***.

iii. In a relation R(A,B,C),if there is a multi-valued dependency between ***A*** and ***B***,then***B*** And ***C*** should be independent of each other.

If all these three conditions are true for any relation (table), then it contains multi-valued dependency. The multi-valued dependency can be explained with an example. Let the RelationR containing three columns *A*, *B*, *C* and four rows *s*, *t*, *u*, *v*.

	A	B	C
s	a1	b1	c1
t	a1	b1	c2
u	a1	b2	c1
v	a1	b2	c2

If s(A) =t(A)=u(A)=v(A)

s(B)=t(B) ands(B) =v(B)

s(C)=u(C)andt(C)=v(C), then thereexist multi-valueddependency.

Example: Consider the below college enrolment table with columns HTNO,Subject and Hobby.

501

Subject

Java C#

Hobby

Cricket Dancing

Hobby As shown in the above figure, if501 opted forsubjects likeJavaand C# and hobbies of 501 are Cricket and Dancing. Similarly, If 502 opted for subjects like Python andAndroid and hobbies of 501 are Chess and Singing, then it can be written into a table with three columns as follows:

HTNO	***Subject***	***Hobby***
501	Java	Cricket
501	Java	Dancing
501	C#	Cricket
501	C#	Dancing
502	Python	Chess
502	Python	Singing
502	Android	Chess
502	Android	Singing

As there exis tmulti valued dependency, the above table is decomposed into two tables such that

HTNO	***Subject***
501	Java
501	C#
502	Python
502	Android

HTNO	***Hobby***
501	Cricket
501	Dancing
502	Chess
502	Singing

Now these tables(relations)satisfy the fourth normalform.

6. 5NF

A relation is said to be in 5-NFif and onlyif it satisfies the followingconditions

- Itshouldbeinthe**Fourth NormalForm**.
- Thetableshould not haveany**join Dependency**and joiningshould belossless.

5NF is also known as Project-join normal form (PJ/NF).

A table is decomposed into multiple small tables to eliminate redundancy, and when we re-join the decomposed tables, there should not be any loss in the original data or shold not create any new data. In simple words, joining two or more decomposed table should not lose records nor create new records.

Example: Consider a table which contains a record of **Subject**, **Professor** and **Semester**in three columns. The primarykeyis the combination of all three columns. No column itself is not a candidate key or a super key.

In the table, DBMS is taught by Ravindar and Uma Rani in semester4, DS by Sindhusha and Venu in sem 3. In this case, the combination of all these fields required to identify valid data.

So to make the table into 5NF, we can decompose it intot here relations,

Subject	***Professor***	***Semester***
C	Srilatha	2
DBMS	Ravindar	4
DS	Sindhusha	3
DBMS	UmaRani	4
CN	Srikanth	5
DS	Venu	3
WT	Srinivas	5

The above table is decomposed into three tables as follows to bringit into 5-NF.

Subject	***Professor***
C	Srilatha
DBMS	Ravindar
DS	Sindhusha
DBMS	UmaRani
CN	Srikanth
DS	Venu
WT	Srinivas

Semester	***Professor***
2	Srilatha
3	Ravindar
5	Sindhusha
3	UmaRani
5	Srikanth
2	Venu
5	Srinivas

Semester	***Subject***
2	C
4	DBMS
3	DS
5	CN
5	WT