UNIT-III

Schema Refinement (Normalization) : Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF, 2NF and 3 NF), concept of surrogate key, Boyce-codd normal form(BCNF), Lossless join and dependency preserving decomposition, Fourth normal form(4NF).

Schema Refinement (Normalization): Purpose of Normalization or schema refinement

Storing the same information redundantly, that is, in more than one place within a database, can lead to several problems:
Redundant storage: Some information is stored repeatedly.

Update anomalies: If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

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

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

Anomalies in DBMS are caused when there is too much redundancy in the database's information. Anomalies can often be caused when the tables that make up the database suffer from poor construction. The normalization process was created largely in order to reduce the negative effects of creating tables that will introduce anomalies into the database.

Example:

Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, e_name for storing employee's name, e_address for storing employee's address, and e_dept for storing the department details in which the employee works. At some point in time the table looks like this: e_id e_name e_address e_dept

e_id	e_name	e_address	e_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Update anomaly:
In the above table, we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two

rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly:
Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if the e_dept field doesn't allow nulls.

Delete anomaly:
Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having e_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

**To overcome these anomalies in DBMS, we need to normalize the data.*

Concept of functional dependency:

Introduction to Functional Dependency
Armstrong's axioms/properties of functional dependencies
Types of Functional Dependency
Attribute Closure

Introduction to Functional Dependency:

Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.
A functional dependency is denoted by an arrow "→". The functional dependency of X on Y is represented by X → Y.
Let's understand Functional Dependency in DBMS with example.

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.
Here, are some key terms for Functional Dependency in Database:

Key Terms	Description
Axiom	Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them. together

Armstrong's axioms/properties of functional dependencies:

Below are the Three most important rules for Functional Dependency in Database:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Secondary Rules Armstrong's Inference Rules

1. Projectivity or Decomposition Rule : If A → BC, Then A → B and A → C
2. Union or Additive Rule : If A→B, and A→C Then A→BC.
3. Pseudo Transitive Rule : If A → B, DB → C, then DA → C

Types of Functional Dependency

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency
5. Partial dependency

Trivial Functional Dependency:

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant. i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}** Similarly, **roll_no → roll_no** is also an example of trivial functional dependency.

Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If **X → Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no.**

Similarly, **{roll_no, name} → age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.**

i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

For example,

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

In this example, maf_year and color are independent of each other but dependent on car_model.
In this example, these two columns are said to be multivalue dependent on car_model.
This dependence can be represented like this:
car_model -> maf_year
car_model-> colour

Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency**

For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no → dept** and **dept → building_no**,
Hence, according to the axiom of transitivity, **enrol_no → building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

***Partial Dependency*:**

Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key.
The 2nd Normal Form (2NF) eliminates the Partial Dependency.
Let us see an example −

Example

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

In the above table, we have partial dependency; let us see how −
The prime key attributes are **StudentID** and **ProjectNo**, and

StudentID = Unique ID of the student
StudentName = Name of the student
ProjectNo = Unique ID of the project
ProjectName = Name of the project

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.
The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent.
The **ProjectName** can be determined by **ProjectNo**, which makes the relation Partial Dependent.

Attribute Closure:

Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
How to find attribute closure of an attribute set?
To find attribute closure of an attribute set:

- Add elements of attribute set to the result set.
- Recursively add elements to the result set which can be functionally determined from the elements of the result set.

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

- A → BC
- BC → DE
- D → F
- CF → G

- Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A-

A^+ = { A }
= { A , B , C } (Using A → BC)
= { A , B , C , D , E } (Using BC → DE)
= { A , B , C , D , E , F } (Using D → F)
= { A , B , C , D , E , F , G } (Using CF → G)
Thus,
A^+ = { A , B , C , D , E , F , G }

Closure of attribute D-

D^+ = { D }
= { D , F } (Using D → F)
We can not determine any other attribute using attributes D and F contained in the result set.
Thus,
D^+ = { D , F }

Closure of attribute set {B, C}-

{ B , C }$^+$= { B , C }
= { B , C , D , E } (Using BC → DE)
= { B , C , D , E , F } (Using D → F)

= { B , C , D , E , F , G } (Using CF → G)
Thus,
{ B , C }$^+$ = { B , C , D , E , F , G }

Finding the Keys Using Closure-
Super Key-

- If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.
- Thus, we can say-

"The closure of a super key is the entire relation schema."

Example-
In the above example,

- The closure of attribute A is the entire relation schema.
- Thus, attribute A is a super key for that relation.

Candidate Key-

- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Normal forms based on functional dependency: 1NF

First Normal Form (1NF):
If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in first normal form if it does not contain any **composite** or **multi-valued attribute**. A relation is in first normal form if every attribute in that relation is singled valued attribute.
A table is in 1 NF if:

1. There are only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.

Consider the examples given below.

Example-1:
Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Normal forms based on functional dependency : 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_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

Normal forms based on functional dependency : 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.

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:

{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

Concept of Surrogate Key:

Surrogate key also called a synthetic primary key, is generated when a new record is inserted into a table automatically by a database that can be declared as the primary key of that table . It is the sequential number outside of the database that is made available to the user and the application or it acts as an object that is present in the database but is not visible to the user or application.

We can say that , in case we do not have a natural primary key in a table, then we need to artificially create one in order to uniquely identify a row in the table , this key is called the surrogate key or synthetic primary key of the table.

Features of the surrogate key :

- It is automatically generated by the system.
- It holds anonymous integer.
- It contains unique value for all records of the table.
- The value can never be modified by the user or application.
- Surrogate key is called the factless key as it is added just for our ease of identification of unique values and contains no relevant fact(or information) that is useful for the table.

Consider an example :

Suppose we have two tables of two different schools having the same column registration_no , name and percentage , each table having its own natural primary key, that is registration_no.

Table of school A –

registration_no	name	percentage
210101	Harry	90
210102	Maxwell	65
210103	Lee	87
210104	Chris	76

Table of school B –

registration_no	name	percentage
CS107	Taylor	49
CS108	Simon	86
CS109	Sam	96
CS110	Andy	58

Now, suppose we want to merge the details of both the schools in a single table
Resulting table will be:

surr_no	registration_no	name	percentage
1	210101	Harry	90
2	210102	Maxwell	65
3	210103	Lee	87
4	210104	Chris	76
5	CS107	Taylor	49
6	CS108	Simon	86
7	CS109	Sam	96
8	CS110	Andy	58

*surr_no is the **Surrogate Key**

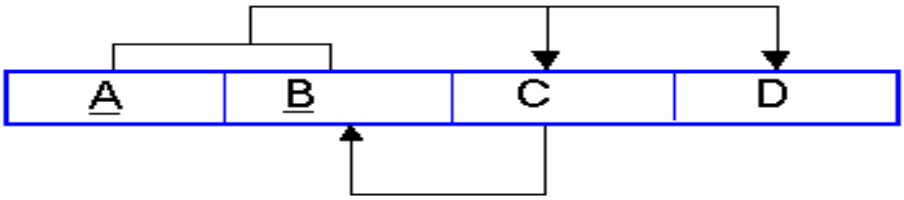
Boyce-Codd normal form(BCNF):

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF.

1. It should be in the **Third Normal Form**.
2. And, for any dependency A → B, A should be a **super key**.

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

- The dependency diagram may look like the one below



- The table is in 3NF. A and B are the keys and C and D depend on both A and B. There are no transitive dependencies existing between the non key attributes, C and D.
- The table is not in BCNF because a dependency exists between C and B. In other words if we know the value of C we can determine the value of B.

Example:

An example table from the University Database might be as follows:

S_Num	T_Code	Offering#	Review Date

123599	FIT104	01764	2nd March
123599	PIT305	01765	12th April
123599	PIT107	01789	2nd May
346700	FIT104	01764	3rd March
346700	PIT305	01765	7th May

The dependencies are *S_Num, T_Code → Offering#, Review Date* which means that the table is in third normal form.

The table is not in BCNF as if we know the offering number we know who the teacher is. Each offering can only have one teacher!
Offering# → T_Code (A non key attribute is a determinant.)

If we look at the table we can see a combination of T_Code and Offering# is repeated several times. eg FIT104 and 01764.

The original table is divided into two new tables. Each is in 3NF and in BCNF.

StudentReview

S_Num	**Offering#**	**Review Date**
123599	01764	2nd March
123599	01765	12th April
123599	01789	2nd May
346700	01764	3rd March
346700	01765	7th May

OfferingTeacher

Offering#	**T_Code**
01764	FIT104
01765	PIT305
01789	PIT107

Lossless Join Decomposition :

Lossless join decomposition is a decomposition of a relation R into relations R1, R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data…

In other words by lossless decomposition, it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition, we select the common attribute and the criteria for selecting a common attribute is that the common attribute must be a candidate key or super key in either relation R1, R2, or both.

Consider a relation R if we decomposed it into sub-parts relation R1 and relation R2.

Here,

R = (A, B, C)

R1 = (A, B)

R2 = (A, C)

The relation R has three attributes A, B, and C. The relation R is decomposed into two relation R1 and R2. . R1 and R2 both have 2-2 attributes. The common attribute is A.

Draw a table of Relation R with Raw Data −

R (A, B, C)

A	B	C
25	12	34
36	10	09
42	12	30

It decomposes into the two sub relations −

R1 (A, B)

A	B
25	12
36	10
42	12

It decomposes into the two sub relations −

R2 (A, C)

A	C
25	34
36	09
42	30

Dependency Preserving Decomposition:

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then
there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.

For example refer class notes. Please prepare this topic with the help 2 examples given in the class.

Fourth normal form(4NF).

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Rules for 4th Normal Form: For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

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

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for 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 conditions are true for any relation(table), it is said to have multi-valued dependency.

Example:

Below we have a college enrolment table with columns s_id, course and hobby.

S_id	Course	Hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

As you can see in the table above, student with s_id **1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

You must be thinking what problem this can lead to, right?

Well the two records for student with s_id **1**, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

S_id	Course	Hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Cricket
1	Maths	Hockey

And, in the table above, there is no relationship between the columns course and hobby. They are independent of each other. So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

How to satisfy 4th Normal Form?

To make the above relation satify the 4th normal form, we can decompose the table into 2 tables.

CourseOpted Table

S_id	Course
1	Science
1	Maths
2	C#
2	Php

And, **Hobbies Table**,

S_id	Hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form.

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).