Back To Course

Learn

Classroom

Theory

Quiz

LIVE BATCHES

Learn Quiz

Filter

We have combined Classroom and Theory tab and created a new Learn tab for easy access. You can access Classroom and Theory from the left panel.

Database Normalization

Objectives of Good Database Design:

1. No updation, insertion and deletion anomalies

2. Easily Extendible

3. Good Performance for all query sets

4. More Informative

Anomalies: There are different types of anomalies which can occur in referencing and referenced relation:

1. **Updation Anomaly:** If some particular attribute value has to be updated, then it has to be updated in every tuple consisting of that value, which would be too costly operation as all records have to be first transferred to the main memory, and then again transferred to the secondary memory after updation.

Student_Id	Student_Name	Subject_Id	Subject_Name
1001	ABC	CS101	Database Management System
1002	XYZ	CS101	Database Management System
1001	ABC	CS102	Operating System
1003	BCD	CS102	Operating System

In the above table, if someone tries to rename the Subject_Name Database Management System to DBMS and while doing so, the system crashes in between, then it would lead to inconsistency of the table. Similarly imposing a bad query can also lead to redundancy issue.

2. **Insertion Anomaly:** If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation. Suppose in the above table we need to add a student PQR with Student_ID 1004 and we don't know the Subject_name or the Subject_Id the student intends to take. If the fields are mandatory then, the addition of the data to the table will not be possible.

3. **Insertion Anomaly:** Deletion of some data may lead to loss of some other useful data. For example, Suppose some student details have to be deleted. Now when the required tuple is deleted, the subject name details corresponding to that student record also gets deleted, which might lead to loss of the details of the subject.

Database Normalization | Functional Dependencies

Database normalization is the process of organizing the attributes of the database to reduce or eliminate **data redundancy (ha[illegible] same data but at different places)** .

Problems because of data redundancy: Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete and update operations.

Functional Dependency

Functional Dependency is a constraint between two sets of attributes in a relation from a database. A functional dependency is denoted by an arrow (→). If an attribute A functionally determines B or B is functionally dependent on A, then it is written as A → B.
For example, employee_id → name means employee_id functionally determines the name of the employee. As another example in a time table database, {student_id, time} → {lecture_room}, student ID and time determine the lecture room where the student should be.
Let's look at the tables below:

Enroll_No	Name	Address	Phone_No

In this table we can say that:
Enroll_No → Name
Enroll_No → Address
Enroll_No → Phone No

Subject_Id	Student_Id	Marks	Grade

In this table we can say that:

(Subject_Id, Student_Id) → Marks
(Subject_Id, Student_Id) → Marks, Grades

Examples: For the following table, which of the following functional dependencies hold True.

1. A → B

2. B → A

A	B
x	1
y	1
z	2

Solution: The first condition holds True as for every value of A there is a unique value in B.
The second dependency results in False, as there is no unique identification for every value of B in A.

What does functionally dependent mean?

A function dependency A → B means for all instances of a particular value of A, there is the same value of B.
Let's do an exercise on Functional Dependency:
For the following table define which of the dependency is True or False

Enroll_No	Name	Address
101	Raj	ABC
102	Ravi	ABC
103	Raj	XYZ

1. Enroll_No → Name, Address

2. Name → Address

3. Address → Name

LIVE BATCHES

Solution:

1. This holds True as for every Enroll_No, there is a unique representation in Name and Address combined.

2. This is False as there is an anomaly in the Name Raj

3. This is False as there is an anomaly in the Address ABC

For example in the below table A → B is true, but B → A is not true as there are different values of A for B = 3.

```
A    B
------
1    3
2    3
4    0
1    3
4    0
```

Why do we study Functional Dependency?

We study functional dependency so that we can carve out a few attributes to another table so that the data redundancy can be reduced to a minimum. Let's look at the following table:

▲

Student_ID	Name	Dept_Id	Dept_Name
101	ABC	10	CS
102	BCD	11	ECE
103	ABC	10	CS
104	XYZ	11	ECE
105	CDE	10	CS

In this table let's find out the functional dependency between two attributes.
As we can see that Dept_Id has each unique identification in Dept_Name, so we can say that Dept_Id → Dept_Name
Therefore we can carve out this two table and create another one out of this.

Dept_Id	Dept_Name
10	CS
11	ECE

Trivial Functional Dependency:
X → Y is trivial only when Y is subset of X.
Examples:

```
ABC → AB
ABC → A
ABC → ABC
```

Non Trivial Functional Dependencies X → Y is a non trivial functional dependencies when Y is not a subset of X.

X → Y is called completely non-trivial when X intersect Y is NULL.
Examples:

```
Id → Name,
Name → DOB
```

Semi Non Trivial Functional Dependencies X → Y is called semi non-trivial when X intersect Y is not NULL.
Examples:

```
AB → BC,
AD → DC
```

LIVE BATCHES

DBMS | Normal Forms

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updating anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

1. First Normal Form

If a relation contains a composite or multi-valued attribute, it violates first normal form or 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**.

Let's understand the concept of 1NF using this exercise:

Customer_ID	Name	Mob No
101	ABC	8860033933, 9899868189
102	BCD	8960133933
103	XYZ	8681899900, 9681888800
104	PQR	8189399888

As we can see that the above table contains multi-valued attributes, which violates the principle of first normal form and hence reduced. There are various methods of doing this:

Method 1: This involves the creation of multiple columns, distributing the values alongside the new attributes. We can distribute the multivalued attributes to new columns by naming them as Mob No 1, Mob No 2, Mob No 3 etc, depending on the number of multivalued attributes till the table gets reduced to single-valued attribute.

Customer_ID	Name	Mob No 1	Mobile No 2	Mobile No 3
101	ABC	8860033933	9899868189	
102	BCD	8960133933	-	
103	XYZ	8681899900	9681888800	
104	PQR	8189399888	-	

But this method got few problems like various fields are needed to be left blank, resulting in wastage of space.

Method 2: This method involves the creation of multiple instead of columns, with copying up of the non-repeated attribute values for each repeated attribute values.

Customer_ID	Name	Mob No
101	ABC	8860033933
102	BCD	8960133933
103	XYZ	8681899900
104	PQR	8189399888
101	ABC	9899868189
103	XYZ	9681888800

In this table we have made a copy of the values of the repeated attributes, keeping other attributes the same. This saves our space but introduces another problem of repetition of Customer_ID, which must remain unique.

Method 3: This method involves the creation of another table and shifting the repeated attributes to that table and linking it with the previous table by using any type of ID.

Customer_ID	Name
101	ABC
102	BCD
103	XYZ
104	PQR

ID	Customer_ID	Mob No
1	101	8860033933
2	102	.
.	.	.
.	.	.

- **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

- **Example 2 -**

```
ID   Name   Courses
------------------
1    A      c1, c2
2    E      c3
3    M      C2, c3
```

In the above table, Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

```
ID   Name   Course
------------------
1    A       c1
1    A       c2
```

```
2     E        c3
3     M        c1
3     M        c2
```

LIVE BATCHES

2. Second Normal Form

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

User_ID	Course_ID	Course_Fee
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000
4	CS102	2000
2	CS103	7000

In the above table, the candidate key is the combination of (User_ID, Course_ID) . The non-prime attribute is Course_Fee. Since this non-prime attribute depends partially on the candidate key, this table is not in 2NF.
To convert this into 2NF, one needs to split the table into two and creating a common link between two. As we have done with the above table after splitting them into two and keeping Course_ID as the common key.

User_ID	Course_ID

Course_ID	Course_Fee

Example:

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

Table 3

Partial Dependency - If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1 -** In relation STUDENT_COURSE given in Table 3,

```
FD set: {COURSE_NO->COURSE_NAME}
Candidate Key: {STUD_NO, COURSE_NO}
```

  In FD COURSE_NO->COURSE_NAME, COURSE_NO (proper subset of candidate key) is determining COURSE_NAME (non-prime attribute). Hence, it is partial dependency and relation is not in second normal form.
  To convert it to second normal form, we will decompose the relation STUDENT_COURSE (STUD_NO, COURSE_NO, COURSE_NAME) as :

```
STUDENT_COURSE (STUD_NO, COURSE_NO)
COURSE (COURSE_NO, COURSE_NAME)
```

  Note - This decomposition will be lossless join decomposition as well as dependency preserving.

- **Example 2 -** Consider following functional dependencies in relation R (A, B , C, D)

```
AB -> C  [A and B together determine C]

BC -> D  [B and C together determine D]
```

  In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form

Def 1: It follows two rules:

1. The table must be in 2NF.
2. No non-prime attribute must defin▲er non-prime attribute.

Let's understand these rules using the following table:

Stud_No	Stud_Name	Stud_State	Stud_Country
101	Ram	Haryana	India
102	Ramesh	Punjab	India
103	Suresh	Punjab	India

LIVE BATCHES

Checking the first condition of 3NF: The candidate key for the table is Stud_No. It has the following dependency on other attributes:
Stud_No → [Stud_Name, Stud_State, State_Country]
Student_State → Stud_Country
In both the above cases, we see that there is only one candidate key and it has no other parts, and also there are no non-prime attributes. Hench the condition of 2NF holds.

Checking the second condition of 3NF: As we can see that Student_State → Stud_Country occurs which means that one non-prime attribute is defining another non-prime attribute, it violates the second rule of 3NF.

This problem can be fixed by splitting the table into two.
T1: Stud_No, Stud_Name, Stud_State
T2: Stud_State, Stud_Country
As in this case, no non-prime attribute is defining another non-prime attribute, hence the condition of 3NF holds.

Let's see another table and try to understand the concept of 3NF:

Exam_Name	Exam_Year	Topper_Name	Topper_DOB
ABC	2016	Ram	15-06-1992
BCD	2017	Ramesh	16-07-1994
EFG	2017	Suresh	10-06-1991
ABC	2017	Kamlesh	11-11-1993

Let's first find out the dependencies:
(Exam_Name, Exam_Year) → (Topper_Name, Topper_DOB)
Topper_Name → Topper_DOB

We can very well see that since there are no repetitions in the columns, 1NF holds. Also, 2NF holds, as no non-prime attribute is partially dependent on the candidate key. But the second dependency violates the 3NF.

To solve this we need to split the table into two as we did in an earlier case:
T1: Exam_Name, Exam_Year, Topper_Name
T2: Topper_Name, Topper_DOB

Let's look at a few other definitions of 3NF:
Def 2: This also defines two sets of rules which are mostly similar to the Def 1.

1. In 2NF and

2. Non-prime attributes are not transitively dependent on prime attributes.

Def 3: It says, for every X → A, one of the following should be true:

1. X is a Superkey

2. A-X is a prime attribute

3. X → is a trivial functional dependency.

Example:

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency - If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 -** In relation STUDENT given in Table 4,

  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE, STUD_STATE -> STUD_COUNTRY}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STU ▲ NTRY is transitively dependent on

STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 -** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF)

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF if in every non-trivial functional dependency X --> Y, X is a superkey.
Let's understand this in a more basic form. We can say that a table is in BCNF if it satisfies the BCNF condition along with all other normalization condition including 1NF, 2NF and 3NF.
For 1NF, there must not be any repeated values in any attributes.
For 2NF, no non-prime attribute must be defined by any prime attributes.
For 3NF, no non-prime attribute must be defined by any non-prime attributes.
For BCNF, no prime or non-prime attribute must be define any prime attributes
Let's look at the following table and try to understand how this works:

STUD_ID	Subject	Prof_id
1001	DBMS	103
1001	OS	110
1002	DBMS	111

Functional Dependencies:
(Stud_ID, Subject) → Prof_id
Prof_id → Subject

Candidate Keys:

(Stud_ID, Subject), Prof_id, (Prof_id, Stud_ID)

The second condition in Functional Dependencies does not follow the rule of BCNF.
Let's try to analyse whether the relation satisfies all type of Normal Forms.
1NF: Since there are no repeated attributes, this holds true.
2NF: Since there are no non-prime attributes, this also holds true.
3NF: Since there are no non-prime attributes, this also holds true.
BCNF: In the second dependency, Prof_id being a prime attribute defines another prime attribute Subject. Hence the condition fails to satisfy BCNF.

Now let's modify the table by adding another row to it:

STUD_ID	Subject	Prof_id
1001	DBMS	103
1002	DBMS	111
1003	DBMS	103

Then break the table into a BCNF compliance database.

STUD_ID	Prof_id
1001	103
1001	110
1002	111
1003	103

Prof_id	Subject
103	DBMS
110	OS
111	DBMS
103	DBMS

Although this decomposition satisfies with the conditions of BCNF, but the DBMS might not accept this split because while splitting the table the first Functional Dependency i.e., (Subject_ID, Subject) → Prof_ID does not exist anymore.

- **Example 1 -** Find the highest normal form of a relation R(A,B,C,D,E) with FD set as -

  ```
  BC->D,
  AC->BE,
  B->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. The prime attributes 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.
  Step 4. The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
  Step 5.The relation is not in 3rd normal form because in BC->D (neither BC is a superkey nor D is a prime attribute) and in B->E (neither B is a superkey nor E is a prime attribute) but to satisfy 3rd normal for, 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.

Key Points -

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also 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.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

```
ABC --> D
CD --> AE
```

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 --> 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 a given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, the CD is not a superkey so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider a CD -> AE. Since E is not a prime attribute, so the 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 a non prime attribute. So, the given relation is also not in 2 NF. So, the highest normal form is 1 NF.

Careers

Privacy Policy

Contact Us

Terms of Service

Data Structures

Languages

CS Subjects

Video Tutorials

LIVE BATCHES

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos

▲