Chapter-5

Database Constraints and Relational Database Design

Relational Database Design

The aim of database design is to reduce redundancy. The storing information several time leads to the waste age of memory space and increases the total size of database. Modification of such database becomes complicated and it generates three types of problems / issues related to redundancy know as anomalies. Normalization is needed to prevent these anomalies.

1. **Insertion Anomaly:**

   An anomaly that occurs during the insertion of record is called insertion anomaly. It is the inability to add information about a new entry into all the places in the database where information about that new entry needs to be stored. In normalized database, information about a new entry needs to be inserted into only one place in the database.

2. **Deletion Anomaly:**

   An anomaly that occurs during the deletion of record is called deletion anomaly. It is the inability to remove all the information about an existing database entry completely from the database. It is also the unintended loss of data due to the deletion of other data. In normalized database, deletion of any entry is done from only one place in the database.

3. **Modification Anomaly:**

   An anomaly that occurs during the modification of records is called modification anomaly. This is the problem in which if a single data is updated in one table then it requires several records in same or another table to be updated. If not done then, the database becomes inconsistent. In normalized database, the updation can be done from only one place in the database.

Integrity Constraints

Integrity refers to accuracy of data in a database. Integrity constraint is a mechanism to prevent invalid data entries in the database. So constraints are used for enforcing the rules in a database.

There are three types of integrity constraints.

1. **Domain Integrity Constraints:**

   It ensures that only a valid range of values is stored in a column. It prevent unnecessary manipulation of data at column level of a table. There are three types of domain integrity constraints.

   a. **NOT NULL:**

      It specifies that a value for a column can't be NULL. If a column is declared as not null, a value has to be necessary to be inserted for such column. Normally, primary key column are declared as NOT NULL however any other fields also can be declared as NOT NULL.

   b. **CHECK:**

It specifies a particular range of values for a column. A check constraint enforces domain integrity by restricting the value to be inserted in a column.

c. **DEFAULT:**

A default constraint can be used to assign a constant value to a column and the user need not insert values for such column. However if the user provides value then the particular value will be stored.

2. **Entity Integrity Constraints:**

It can be applied to single or multiple columns in a table which ensures that each row can be uniquely identified by an attribute called primary key. The primary key column contains unique values. In addition, the primary key column can't be NULL. There are two types of Entity integrity constraints.

a. **Primary key:**

It ensures a column value can't be NULL or must be unique.

b. **Unique:**

It ensures a column value must be unique i.e. no duplication of records. However the Unique constraint allows one NULL value also.

3. **Referential Integrity Constraints:**

It ensures that the values of the foreign key match the value of the corresponding primary key i.e. it ensure that a value that appear in one relation / table for a given set of attributes also appears for certain set of attributes in another table. It is a set of rules which we can set to establish and preserve the relationship between tables when we add, change or delete the records. The referential integrity constraint are enforces by the use of primary key and foreign key. The rules that can be enforces Referential integrity are

a. **Cascade Update:**

The cascade update rule between two tables is an option that causes a change to primary key field in primary table to automatically change all the related filed in foreign table that references the primary key.

b. **Cascade Delete:**

The cascade delete rule between two tables is an option that causes deletion of a record from primary table to automatically delete all the related record in foreign table.

Example:

Let us create two table tbl_std and tbl_marks which are coupled together using foreign key.

```
CREATE TABLE tbl_std
(
id INT PRIMARY KEY,
name VARCHAR(50),
roll INT,
sem VARCHAR(5)
)

CREATE TABLE tbl_mark
```

```
(
id INT FOREIGN KEY REFERENCES tbl_std(id) ON DELETE CASCADE ON UPDATE
CASCADE,
maths INT,
science INT,
english INT
)
```

What is the output of execution of following queries.

1. INSERT INTO tbl_mark VALUES (1,55,66,77)

Result: Not allowed because, the foreign key field has null references to Primary key field i.e. the student with id 1 is not available in primary table tbl_std.

2.
```
INSERT INTO tbl_std VALUES (1,'Ram',5,'I')
INSERT INTO tbl_mark VALUES(1,55,66,77)
INSERT INTO tbl_std VALUES (2,'Hari',6,'I')
INSERT INTO tbl_mark VALUES(2,45,56,57)
INSERT INTO tbl_std VALUES (3,'Rabina',7,'I')
INSERT INTO tbl_mark VALUES(3,95,59,89)
```

Result:

tbl_std

id	Name	Roll	Sem
1	Ram	5	I
2	Hari	6	I
3	Rabina	7	I

tbl_mark

Id	Maths	Science	English
1	55	66	77
2	45	56	77
3	95	59	89

3. Change the id of Ram to 10

```
UPDATE tbl_std SET id=10 WHERE name='RAM'
```

Result:

tbl_std

id	Name	Roll	Sem
10	Ram	5	I
2	Hari	6	I
3	Rabina	7	I

tbl_mark

Id	Maths	Science	English
10	55	66	77
2	45	56	77
3	95	59	89

4. Delete the record of student whose id is 3

```
DELETE FROM tbl_std WHERE id=3
```

Result:

tbl_std

Id	Name	Roll	Sem

tbl_mark

Id	Maths	Science	English
10	55	66	77
2	45	56	77

10	Ram	5	I
2	Hari	6	I

Functional Dependencies

The normalization theory is based on the fundamental notation of functional dependency. Give a relation R, attribute B is said to be functionally dependent on attribute A if each value of B in R is associated **with precisely one value** of A. In other word, attribute B is functionally dependent on A if and only if, for each value of B there is exactly one value of A. So it is also called, A determine B or B is function of A.

A functional dependency is denoted by A⟶ B and read as 'A determine B'. A is called the determinant and B is the object of determinant. If more than one attribute is necessary to determine another attribute, then such determinant is called as **compound Determinant.**

Example: Let us consider the following realation.

Roll number	Course code	Course name	Teachers name	Room number	Marks	grade
1	Cmp104	DSA	Bhesh Thapa	202	90	A
2	Cmp103	C	Ranjan Adhikari	204	86	A-
3	Cmp103	C	Bhesh Thapa	204	88	A-
1	Cmp103	C	Bhesh Thapa	204	80	B+

The preceding table has composite key (Roll number + Course code).In the preceding table, for a particular value of Roll number, Course code there is precisely one corresponding value for Marks. Hence Marks is functionally dependent on these two attributes. This can be symbolically represented as

(Roll number, Course code) ⟶ Marks

The other functional dependencies in the preceding table are:

1. Course code ⟶ Course name
2. Course code ⟶ Teacher name
3. Marks ⟶ Grade

Trivial and Non-Trivial Functional Dependency

A functional dependency is **trivial**, if the consequent is a subset of the determinant i.e. a functional dependency of the form

$\alpha \longrightarrow \beta$ is trivial if β is subset of α

Example:

$A \longrightarrow A$

AB ⟶ A

Course code, Course name ⟶ Course code

Course code, Course name ⟶ Course name

Course code ⟶ Course code

So from above examples we see that, it is impossible for trivial dependency not to be satisfied.

A functional dependency is **non-trivial**, if dependency is not trivial i.e. the consequent is not a subset of the determinant.

Example:

A ⟶ B

Teacher name ⟶ Room number

Roll number, Course code ⟶ Marks

Partial and Full Functional Dependency

A partial dependency is dependency in which a non-key attribute is dependent on only a part of composite key. This is the situation in which only a subset of the attributes of the composite key is used to uniquely identify its object.

A full functional dependency is dependency in which a non-key attributes is dependent on all the attributes of composite key. This is the situation in which all the attributes of the composite key is used to uniquely identify its object.

Example:

The attribute Course name is not fully functionally dependent on composite key (Roll number + Course code). It is only dependent on subset of composite key i.e. Course code. Similarly, Teacher name and Room number are only dependent on Course code. So Course name, Teacher name, Room number attributes are said to have partial dependencies on the whole key.

Again, the attribute Mark and Grade are dependent on all the attributes of composite key (Roll number +Course code). So the attributes Mark and Grade are said to have full functional dependencies on the whole key.

Transitive (Indirect) Dependency

A functional dependency of the form X ⟶ Z is said to be Transitive if both X ⟶ Y and Y ⟶ Z holds.

Example:

In the preceding table, Room number is dependent on Teacher name and Teacher name is dependent on Course code. Therefore Room number is dependent on Course code. This type of dependency is called transitive dependency.

It is shown as below

Course code ⟶ Teacher name and Teacher name ⟶ Room number means Course code ⟶ Room number

Closures of a set of functional dependency

If F be a set of functional dependencies, then closure of F, denoted by F^+, is the set of all functional dependencies logically implied by F. F^+ is superset of F.

Finding all F^+ by applying rules of inference

There are 6 different rules of inference.

1. If β is subset of α, then $\alpha \longrightarrow \beta$. (Reflexivity Rule)
2. If $\alpha \longrightarrow \beta$, then $\gamma\alpha \longrightarrow \gamma\beta$ (Augmentation Rule)
3. If $\alpha \longrightarrow \beta$ and $\beta \longrightarrow \gamma$, then $\alpha \longrightarrow \gamma$ (Transitive Rule)
4. If $\alpha \longrightarrow \beta$ and $\alpha \longrightarrow \gamma$, then, $\alpha \longrightarrow \beta\gamma$ (Union /Additive Rule)
5. If $\alpha \longrightarrow \beta\gamma$, then $\alpha \longrightarrow \beta$ and $\alpha \longrightarrow \gamma$ (Projection / Decomposition Rule)
6. If $\alpha \longrightarrow \beta$ and $\beta\gamma \longrightarrow \delta$, then $\alpha\gamma \longrightarrow \delta$ (Pseudotransitivity Rule)

Example

Suppose we are given Schema R={A,B,C,G,H,I} and set of functional dependencies F={A→ B,A→ C, CG→ H, B→H, CG→I}. Find the closures of functional dependency F.

Sol$^{n:}$ F= {A→ B,A→ C, CG→ H, B→H, CG→I}

Some of members of F^+= {A → H, AG → I, AG → H, CG → HI }

A → B and B → H, therefore A → H (Transitivity Rule)

AG → CG (Augmentation Rule)

Now, AG → CG and CG → I, CG → H , therefore AG →I, AG → H

CG → HI (Union Rule)

Multi-valued Dependency (MVD):

MVD occurs if two or more independent multi-valued facts about the same attribute occur within the same relation. MVD X→→ Y read as " X multi-determines Y" defines a relationship in which a set of attributes Y are determined by a single value of X.

An MVD X →→ Y is said to hold over Relation R if, for each instance r of R, X value is associated with a set of Y values and this set is independent of the values in the other attribute.

An MVD X→→Y over Relation R is said to be trivial MVD if y is subset of x or X and Y together form the relation R i.e. X U Y=R otherwise it is said to be non-trivial MVD.

Consider the following relation with MVD.

Course	Teacher	Book
DBMS	Ram Thapa	DB concept
DBMS	Hari Sharma	Katson Books
DBMS	Hari Sharma	DB concept
DBMS	Ram Thapa	Katson Books

Here Course→→Teacher and Course →→Book

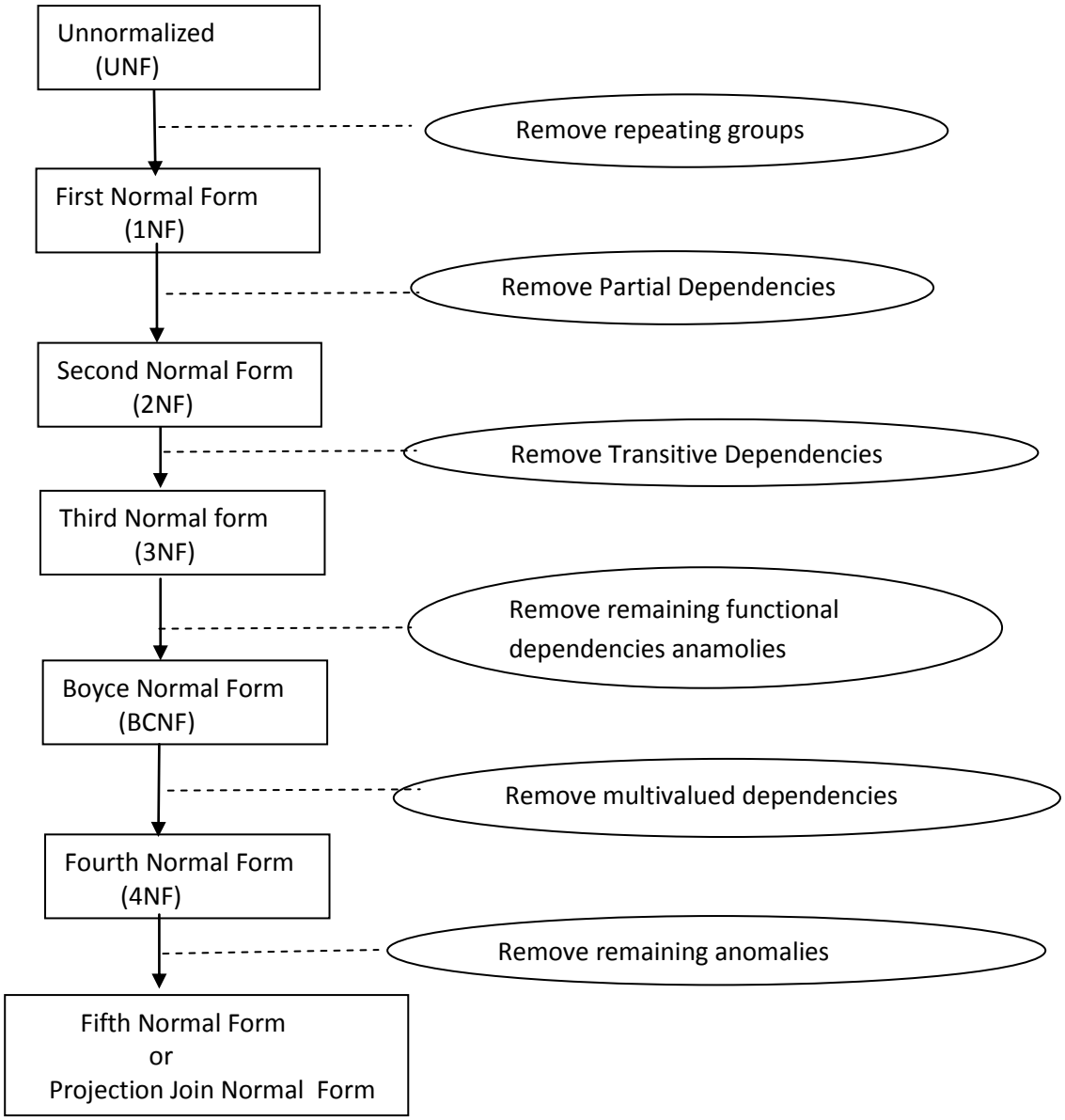
Normalization

Normalization is the process of breaking down complex table structures into simple table structure by using certain rules. Using this method, we can reduce redundancy in a table, and eliminate the problems of inconsistency and disk space usage. Also we can ensure that there is no loss of information.

Advantages of Normalization

1. It helps in maintaining data integrity.
2. It helps in simplifying the structure of table.
3. Easier to add data.
4. It is flexible model for structuring of data.
5. It eliminates redundant data.
6. Less storage space.
7. Better Understanding of data because each table stores data for a single type of entity.

Stages of Normalization



1. Un-Normalized Normal Form (UNF)

A relation is Un-normalized if no any normalization r rules has been applied to it. An un-normalized relation suffers from various anomalies. Any un-normalized relation has repeating groups, i.e. it has more than one value or list of values for a cell. Hence Cell is not single-valued. Data redundancy is another main problem for such relation.

Consider the following PROJECT relation.

EId	Dept	Depthead	Projectcode1	Projectcode2	Hours1	Hours2
1	System	Ram Thap	P27	P51	90	101
2	Finance	Hari Sharma	P27	P22	109	98
3	Admin	Rita Shrestha	NULL	P27	NULL	72

The above PROJECT relation is un-normalized because there are repeating groups .ie two Projectcode fields and two Hours field. In this table, to add a third Projectcode, we have to add new projectcode3 field. So this is not an efficient way of designing a database.

2. First Normal Form (1NF):

A relation is said to be in 1NF if and only if

- There are no duplicate rows in the table .
- Each cell is single valued i.e. there are no repeating groups or arrays.
- Data for a particular column are of similar kind.

The above PROJECT table already satisfies two conditions to be in 1NF i.e. Data for each attributes are of same kind and there are no duplicate rows in the table. However it is not in 1NF because it has two repeating groups i.e. Projectcode and Hours. So by applying the 1NF rules, we arrive at the following table.

Eid	Dept	Depthead	Projectcode	Hours
1	System	Ram Thapa	P27	90
1	System	Ram Thapa	P51	101
2	Finance	Hari Sharma	P27	109
2	Finance	Hari Sharma	P22	98
3	Admin	Rita Shrestha	P27	92

Example 1:

1. Convert the following un-normalized relation to 1NF

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

Fig: Un-normalized relation (because Color columns is not atomic i.e. contain multiple values)

Product ID	Color	Price
1	Red	15.99
1	Green	15.99
2	Yellow	23.99
3	Green	17.50
4	Yellow	9.99
4	Blue	9.99
5	Red	29.99

Fig: 1NF

Or

TABLE_PRODUCT_PRICE

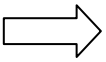
Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Example 2:

ID	Lastname	Phone
1	Smith	555-1212
2	Jones	352-555-2323
3	Smith	233-555-9877

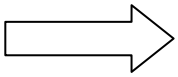


ID	Lastname	areacode	exchange	extension
1	Smith		555	1212
2	Jones	352	555	2323
3	Smith	233	555	9877

Our phone number is really not as atomic as it could be. The following would place this entity in 1NF.

Example 3:

patient	Allergies
1	pollen, cats, dust
2	dander, wheat
3	None



patient	Allergies
1	pollen
1	Cats
1	Dust
2	dander
2	wheat
3	none

3. Second Normal Form (2NF):

A relation is in 3NF if and only if

- It is in 1NF.
- No partial dependencies exists between non key attributes and key attributes.

The main aim of second normal form is to ensure that all information in one relation is only about one thing. We should also maintain relationship between these new table and their predecessors through the use of foreign key.

In the preceding table, the primary key is composite (Eid + Projectcode) and there are no repeating groups so it satisfies the definition of 1NF. Also the non key attribute Hours is fully functionally dependent on whole key. However the above table in not in 2NF because the non key attributes Dept and Depthead are functionally dependent on part of Composite key i.e. EiD but not Project code.

So by applying the rules of 2NF, i.e. removing the partial dependencies by placing the removed attributes in a different table along with the attribute (determinant) they are functionally dependent on, we arrive at the following two tables.

Eid	Dept	Depthead
1	System	Ram Thapa
2	Finance	Hari Sharma
3	Admin	Rita Shrestha
4	System	Ram Thapa

Eid	Projectcode	Hours
1	P27	90
1	P51	101
2	P27	109
2	P22	98
3	P27	72
4	P80	88
5	P99	90

Example:

Consider the following example:

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

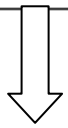
TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

Example:

Product ID	Color	Price
1	Red	15.99
1	Green	15.99
2	Yellow	23.99
3	Green	17.50
4	Yellow	9.99
4	Blue	9.99
5	Red	29.99



TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Example:

Order(order_no, order_date, item_code, quantity, price)

Given functional dependecnies fd1= order_no→order_date

Fd2= order_no, item_code→quantity

Fd3= item_code→price

Here composite key is (order_no + item_code) and we see partial dependencies so converting to 2NF we get

Tbl1(order_no,order_ate)

Tbl2(item_code, price)

Tbl3(order_no,item_code,quantity)

4. Third Normal Form (3NF):

A relation is said to be in 3NF if and only if

- It is in 2NF.
- No transitive dependencies exists between non-key attributes and the key attributes i.e. no non key attribute functionally dependent on other non-key attributes.

In the preceding tables, both of them are in 2NF because hours is fully functionally dependent on whole key (id + project code), Dept and Depthead are also functional y dependent on Eid. However it is not in 3NF because, the non key attribute Depthead is dependent on the non key attribute Dept, resulting transitive dependencies.

So by applying the rules of 3NF, i.e. removing the non key attribute Depthead which is dependent on non-key attribute Dept, by placing the removed attribute in different table with the attribute it is functionally dependent on , we arrive at the following three tables.

Eid	Dept
1	System
2	Finance
3	Admin
4	System
5	Sales

Dept	Depthead
System	Ram Thapa
Finance	Hari Sharma
Admin	Rita Shrestha

Eid	Projectcode	Hours
1	P27	90
1	P51	101
2	P27	109
2	P22	98
3	P27	72
4	P80	88
5	P99	90

5. Boyce-Codd Normal Form (BCNF)

BCNF is an extended form of 3NF. The original definition of 3NF is not sufficient in -for the table:

- That has multiple candidate keys.
- Where the multiple candidate keys were composite.
- Where the multiple candidate keys overlapped i.e. some attribute in the keys are common.

Therefore a new normal form named BCNF was introduced. In tables, where the preceding two conditions do not apply, we can stop at third normal form. In such case 3NF is same as the BCNF.

A relation is in BCNF is and only if

- It is in 3NF.
- if every determinant is a candidate key.

Consider the following PROJECT table.

ECODE	NAME	PROJECTCODE	HOURS
E1	Vimal	P001	45
E2	Arbin	P002	100
E3	Nabin	P001	20
E1	Vimal	P002	70
E3	Nabin	P003	80

In the preceding table, ECODE + PROJECTCODE is the primary key, however NAME + PROJECTCODE could be choose as the primary key hence is a candidate key. It is in 3NF because, HOURS is Functional dependent on ECODE + PROJECTCODE and NAME + PROJECTCODE i.e. no partial and transitive dependencies exist here. Also we see that, Name is functional dependent on ECODE and ECODE is functional dependent on NAME, Multiple candidate keys, Composite candidate key, and overlapped candidate key i.e. PROJECTCODE is common between the two candidate key. So this situation requires conversion to BCNF. As per the rule of BCNF, every determinant must be candidate key but here ECODE and NAME are determinant but are not candidate key by themselves. Hence using BCNF , make the determinant to be candidate keys, i.e. removing NAME and ECODE and place them in different table, we arrive at following two tables.

ECODE	NAME
E1	Vimal
E2	Arbin
E3	Nabin

ECODE	PROJECTCODE	HOURS
E1	P001	45
E2	P002	100
E3	P001	20
E1	P002	70
E3	P003	80

Example:

Consider the following relation

R={Student_id, student_name, Subject_code, Grade}, convert it to BCNF.

It results in two table as below

R1={Student_id, Student_name} and R2={Student_id, subject_code, Grade }

6. Fourth Normal Form (4NF):

A table is in 4NF is and only if

- if it is in BCNF.
- it contains no multivalued dependencies.

The redundancy caused by MVDs can't be removed by transforming the database schema to BCNF. So there is a stronger normal form called 4NF that treats MVDs as FDs when it comes to decomposition. If a relation is in 4NF then it is also in BCNF.

Let us consider the following relation.

Course	Teacher	Book
DBMS	Ram Thapa	DB concept
DBMS	Hari Sharma	Katson Books
DBMS	Hari Sharma	DB concept
DBMS	Ram Thapa	Katson Books

Decomposing the above tabele into 4NF, we arrive at the following two tables.

Course	Teacher
DBMS	Ram Thapa
DBMS	Hari Sharma

Course	Book
DBMS	DB concept
DBMS	Katson Books

So in 4NF, if we have more than one multi-valued attribute then, we should decompose it to have relations that has information about only one entity so as to remove the difficulties with multi-valued fact.

Decomposition of Relation Schema

If a relation R is not in any normal form and we wish the relation to be normalized to remove several anomalies then it is necessary to decompose the relation into two or more relations set R_1, R_2,R_3…….R_N. The decomposed relations R_1, R_2, R_3…….R_N. are projections of R and are not disjointing otherwise the glue holding the information together would be lost.

Hence, Careless decomposition leads to another problem of bad design.

Properties of decomposition

- Attribute preservation
- Lack of redundancy
- Lossless join decomposition.
- Dependency Preservation

a. Attribute Preservation:

All the attributes that were in the original relation which is being decomposed, must be preserved in the resulting decomposed relations set.

b. Lack of Redundancy:

Redundancy is the problem of repetition of data in the database. Such problem should be avoided as much as possible.

c. Lossless Join Decomposition:

A decomposition of a relation R into relations R_1, R_2, R_3……R_N is called lossless / on-loss decomposition if the relation R is always the natural join of relations R_1, R_2, R_3……R_N. It should be noted that natural join is the way to recover the relation form the decomposed relations.

i.e. if. $R=\prod_{R1}(R) \bowtie \prod_{R2}(R) \bowtie \prod_{R3}(R)\ldots\ldots\ldots\ldots \prod_{RN}(R)$ is lossless decomposition otherwise it is lossy decomposition.

Example: Decomposition of Relation R = {A, B, C}

R

A	B	C
1	P	U
2	Q	V
3	R	w

Decomposed Relations R1 and R2

R 1

A	B
1	P
2	Q
3	R

$\prod_{A, B}(R)$

R2

B	C
P	U
Q	V
R	W

$\prod_{B, C}(R)$

The decomposition of R into R1 and R2 is lossless because $R=\prod_{R1}(R) \bowtie \prod_{R2}(R)$.

Also we can check if the decomposition is lossless or not using concept of functional dependency. Using it we can say the two relation R1 and R2 decomposed from Relation R is lossless if at least one of the following condition is satisfied.

$$R_1 \cap R_2 \rightarrow R_1 - R_2 \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2 - R_1$$

d. Dependency Preservation: A decomposition $D=\{R_1, R_2, R_3 \ldots\ldots\ldots R_N\}$ of R is dependency preserving with respect to F if the union of the projections of F on each R_i in D is equivalent to F i.e. $(F_1 \cup F_2 \cup \ldots\ldots F_N)^+ = F^+$

Example: Given R(A, B, C) and F={A → B, B → C}
The above relation R can be decomposed in two different ways.

1. $R_1 = \{A,B\}$ and $R_2 = \{B, C\}$
   Here $F_1 = \{A \rightarrow B\}$ and $F_2 = \{B \rightarrow C\}$
   Here $(F_1 \cup F_2)^+ = F^+$, Therefore the decomposition is dependency preserving.

2. $R_1 = \{A,B\}$ and $R_2 = \{A, C\}$
   Here $F_1 = \{A \rightarrow B\}$ and $F_2 = \{A \rightarrow C\}$
   Here $(F_1 \cup F_2)^+ \neq F^+$, Therefore the decomposition is not dependency preserving.

Join Dependency (JD) and Fifth Normal Form (5NF)

A Relation R is subject to a join dependency denoted by JD if R can always be recreated by joining multiple tables each having a subset of the attributes of R. If one of the Relations in the join has all the attributes of the Relation Rs, the join dependency is called trivial join dependency.

Let R be a relation schema and $R_1, R_2 \ldots\ldots\ldots R_N$ are projection of R. a legal relation r(R) satisfies the join dependency (JD) if and only if join of projection of relation on R_i [i=1, 2, 3……..N] is equal to

$$R = \prod_{R1}(r) \bowtie \prod_{R2}(r) \bowtie \prod_{R3}(r)\ldots\ldots\ldots\ldots \prod_{RN}(r)$$

Fifth normal form is based on the join dependency. A relation R is in fifth normal form or project join normal form (PJNF) if and only if every non trivial join dependency is implied by the super keys of R. Also, if the relation is in 4NF and every join dependency in it is implied by the candidate key then, we can say that the relation is in 5NF.
Let us consider the following relation PRODUCT.

Company	Product	Dealers
Chaudari Groups	Television	Gautam Brothers
Chaudari Groups	Noodles	Kunwar Brothers
Jyoti Groups	Television	Kunwar Brothers
Khetan Groups	Noodles	Janaki Traders
Jyoti Groups	Bike	Gautam Brothers
Jyoti Groups	Television	Gautam Brothers
Chaudari Groups	Television	Kunwar Brothers

Applying the rules of 5Nf, we arrive at the following three relations.

Company	Product
Chaudari Groups	Television
Chaudari Groups	Noodles
Jyoti	Television
Khetan	Noodles
Jyoti Groups	Bike

Company	Dealer
Chaudari Groups	Gautam Brothers
Chaudari Group	Kunwar Brothers
Jyoti groups	Kunwar Brothers
Khetan	Janaki Traders
Jyoti Groups	Gautam Brothers

Product	Dealer
Television	Gautam Brothers
Noodles	Kunwar Brothers
Television	Kunwar Brothers
Noodles	Janaki Traders
Bike	Gautam Brothers

Triggers

A Trigger is a special kind of stored procedure that the system executes automatically as a side effect of a modification to the database. Most of the time while performing data manipulation on a database object, we might also need to perform manipulation on another object. To perform such operation, DBMS allow us to implement triggers. A trigger is a set of T-SQL statement activated in response to certain action, such as insert or delete. Triggers are used to ensure data integrity before and after performing data manipulation.

When to use Trigger?

If you own an organization and your employees use online leave approval system to apply for leaves. Now when any employees apply for a leave, the leave details are stored in LEAVE_DETAIL table. In addition a new record is automatically added to the LEAVES_APPROVAL table so that the supervisor sees it sometime later. So for performing such automated task in database, triggers are used.

Steps to design trigger

1. Specify when trigger to be executed.

2. Specify the action to be taken when the trigger executes.

Three parts of trigger

1. **Event** like insert, update or delete which activates a trigger.
2. **Condition**, which test if trigger should run or not.
3. **Action,** task that run if trigger is activated.

Types of Trigger

1. **DML Trigger:**

   A DML trigger is fired when data in the underlying table is affected by DML statements such as insert, update, or delete. These trigger help in maintaining consistent, reliable and correct data in tales.

   Types:

   a) **Insert trigger:** is fired whenever an attempt is made to insert a row in the trigger.

   b) **Delete trigger:** is fired whenever an attempt is made to delete a row in the trigger.

   c) **Update trigger:** is fired whenever an update statement is executed in the trigger table.

2. **DDL Trigger:**

   A DDL trigger is fired in response to DDL statement, such as create, alter table etc. DDL trigger can be used to perform administrative tasks. E.g. A database administrator can be notified whenever a table is created in master database by creating a DDL trigger.

EXAMPLE:

a) Create an insert trigger to display all the record of PRODUCT relation.

```
create trigger trg_display on PRODUCT
for INSERT AS
BEGIN
SELECT * FROM PRODUCT
END
```

Under execution of above statement, a trigger named trg-display is created for table PRODUCT.

```
insert into tbl_ram values (6,'ram',3,4,5,5)
```

Now if the above INSERT query is executed on trigger table PRODUCT, the insert trigger automatically fired to show the entire records on table PRODUCT.

b) Delete the above trigger.

```
Drop trigger trg_display
```

Assertions

An assertion is a statement in SQL that ensures a certain condition will always exist in the database. Assertions are like columns and table constraints, except that they are specified separately from table definition. An assertion is activated automatically on any modification of any table mentioned in assertion. The components in assertions include

a) A constraint name

b) Followed by check

c) Followed by condition

The assertion is said to be violated if the query result is not empty and hence the user is denied from modifying the database.

EXAMPLE: Create assertion to verify the number of boat plus number of sailor is less than 100 .

```
Create ASSERTION validate
CHECK
((select COUNT (S.sid) FROM Sailors S) + (SELECT COUNT(B.bid) FROM Boats B)<100)
```