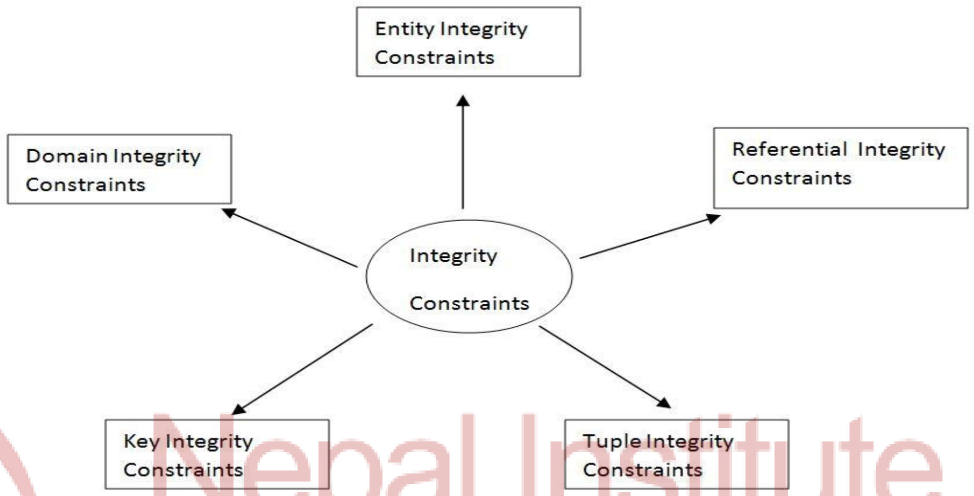
DATABASE CONSTRAINTS AND NORMALIZATION

4.1 Integrity Constraints

Integrity means something like 'be right' and consistent. The data in a database must be right and in good condition. It ensures that changes like update, delete, insertion etc made to database by authorized users do not result in loss of data consistency.


Entity Integrity Constraints
Domain Integrity Constraints
Referential Integrity Constraints
Integrity Constraints
Key Integrity Constraints
Tuple Integrity Constraints

Domain Integrity Constraints

- Domain integrity means the definition of a valid set of values for an attribute.
- It defines :- data type,
  length or size,
  Is NULL value allowed
  Is the value unique or not for an attribute

Example :

Eid	address	phoneNumber	Age	Name
1	P	111	21	A
2	Q	222	28	B
3	R	333	23	C

We have eid[int], address[varchar(20)], phoneNumber[varchar(10)], Age[int], Name[varchar(25)]

Here, we can't set character variable to Age.

Entity Integrity Constraints

It states that primary keys can't be NULL. There can be NULL value for other than primary key fields. It assures that specific row in a table can be identified.

Example :

Reg_id	Model_no	Rate
pq-11	12	100000
rs-23	21	200050
NULL	45	333

Null Value not allowed

Referential Integrity Constraints

It is specified between two tables and it is used to maintain the consistency among rows between the two tables.

The rule are :-

- We can't delete a record from a primary table if matching records exists in related table.
- We can't change a primary key value in the primary table if that records has related records in another table.
- You can't enter a value in the foreign key field of a related table that does not exists in the primary key of the primary table.

Example : Related Table

Reg_id	Model_id	Rate
pq-11	1	4500
rs-23	2	4600
ab-21	6	6000
cse-33	3	7000
as-44	4	80000

Primary Table

Model_id	Mark	Model
1	Ford	Focus
2	Ford	Modeo
3	Tx	307
4	Tx	407

Example : In baking system, the attribute (branch_name) in account relation is a foreign key referencing the primary key of branch relation.

Tuple Integrity Constraints

It states that each and every tuple in a relation should be unique.

S/No	Emp_id	Address
1	X1	A
2	X2	B
3	X2	C
3	X2	C

Same tuple does not exist.

Key Integrity Constraints

Value of primary key must be unique and can't be duplicate.

Duplicate value of primary key does not exist as shown in table .

S/No	Emp_id	Address
1	X1	A
2	X2	B
3	X3	C
3	X4	C

Foreign Key Integrity Constraints

There are two type of foreign key integrity constraints.

- Cascade update related fields
- Cascade delete related rows

These constraints affect the referential integrity constraints.

Cascade update related fields

Any time we change primary key value of a row in primary table, the foreign key value are updated in the related rows of related table.

Primary Table

Model_id	Mark	Model
1	Ford	Focus
2	Ford	Modeo
3	Tx	307
4	Tx	407

Related Table

Reg_id	Model_id	Rate
pq-11	1	4500
rs-23	2	4600
ab-21	6	6000
cse-33	3	7000
as-44	4	80000

If we change Model_id =3 to 6 in primary table, then Model_id=3 in related table will be changed to 6.

Cascade delete related rows

Any time you deleted a row in a primary table, the matching rows are automatically deleted in the related table.

Assertions And Triggering

An **assertion** is a predicate expressing a condition we wish the database to always satisfy.

Domain constraints, functional dependency and referential integrity are special forms of assertion.

Where a constraint cannot be expressed in these forms, we use an assertion, e.g.

- Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.
- Ensuring every loan customer keeps a minimum of $1000 in an account.

An assertion in DQL-92 takes the form,

create assertion assertion-name **check** predicate

Two assertions mentioned above can be written as follows.

Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.

```
Create assertion sum-constraint check
(not exists (select * from branch
            where (select sum)amount) from loan
            where (loan.bname =  branch.bname >= (select sum)amount) from account
            where (account.bname =  branch.bname)))
```

Ensuring every loan customer keeps a minimum of $1000 in an account.

create assertion *balance-constraint* **check**
(**not exists** (**select * from** *loan L* (**where not exists** (**select * from** *borrower B, depositor D, account A* where *L.loan#* = *B.loan#* **and** *B.cname* = *D.cname* **and** *D.account#* = *A.account#* **and** *A.balance* >= 1000)))

When an assertion is created, the system tests it for validity.
If the assertion is valid, any further modification to the database is allowed only if it does not cause that assertion to be violated.
This testing may result in significant overhead if the assertions are complex. Because of this, the **assert** should be used with great care.
Some system developer omits support for general assertions or provides specialized form of assertions that are easier to test.

Triggers

Another feature not present in the SQL standard is the **trigger**.
Several existing systems have their own non-standard trigger features.
A **trigger** is a statement that is automatically executed by the system as a side effect of a modification to the database.
We need to

- Specify the conditions under which the trigger is executed.
- Specify the actions to be taken by the trigger.

For example, suppose that an overdraft is intended to result in the account balance being set to zero, and a loan being created for the overdraft amount.
The trigger actions for tuple *t* with a negative balance are then

- Insert a new tuple *s* in the *borrow* relation with
  s[*bname*] = *t*[*bname*]
  s[*loan#*] = *t*[*account#*]
  s[*amount*] = - *t*[*balance*]
  s[*cname*] = *t*[*cname*]
- We need to negate balance to get amount, as balance is negative.
- Set *t*[*balance*] to 0.

Note that this is not a good example. What would happen if the customer already had a loan?
SQL-92 does not include triggers. To write this trigger in terms of the original System R trigger:

```
define trigger overdraft
    on update of account T
        (if new T.balance < 0 then (insert into loan values
        (T.bname, T.account#, - new T.balance)
        insert into borrower (select cname, account# from depositor
        where T.coount# = depositor.account#)
        update account S set S.balance = 0where S.account# = T.account# ))
```

Functional Algorithm (Chase Algorithm)

In any relation scheme R, where α and β are subset of R i.e. $\alpha \subseteq R$ and $\beta \subseteq R$. Then the functional dependency from α -> β exists on any legal relation r(R) if

$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

Let us take an example:

α	β	
a	1	$\rightarrow t_1$
b	2	$\rightarrow t_2$
c	3	
d	4	

f : α -> β

Where α is determinants and β is dependents.

We don't have to compute here.

If we have the value of α, we have to search the value of β.

There are two types of functional dependency

1. **Trivial Functional Dependency**

   The dependency of as attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

   Symbolically, AB -> B is a trivial functional dependency because B is subset of AB. Here if we consider a tale including two columns student_id and student_name. Then, {student_id , student_name} -> {student_id} is a trivial functional dependency as student_id is a subset of {student_id , student_name}.

2. **Non - Trivial Functional Dependency**

   If a functional dependency X -> Y holds true where Y is not subset of X then this dependency is known as non-trivial functional dependency.

   Example : An employee table with three attributes : emp_id, emp_name, emp_address. Then emp_id -> emp_name is non trivial. Here emp_name is not subset of emp_id.

   [Note : If a functional dependency X -> Y holds true where X∩X is NULL then this dependency is said to be completely non-trivial dependency.]

Multi-valued Dependency

Multi-valued dependency occurs when there are more than one independent multi-valued attributes in a table. A multi-valued dependency is a full constraints between relation. In contrast to the functional dependency, the multi-valued dependency requires that certain tuples be present in a relation. Consider a Bike Manufacturer Company which produces 2 colors (red , black) in each model.

Bike_Model	Mfd_Year	Color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2022	2009	Black
M2222	2009	Red

Here , columsn Mfd_Year and Color are independent of each other and dependent on Bike_Model. In this case these two columns are said to be multi-valued dependent on Bike_Model.

These dependencies can be represented like this.

Bike_Model ->> Mfd_Year

Bike_Model ->> Color

Transitive Dependency

A functional dependency is said to be transitive if it is indirectly formed by two funcitonal dependencies.

X -> Y is a transitive dependency if the following three functional dependencies holds true:

X -> Y

Y does not -> X

Y -> Z

A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF(3rd normal form).

Example :

Book	Author	Author_Age
ABC	E	60
PQR	F	61
XYZ	G	62

- {Book} -> {Author} (If we know the book, we know the author name)
- {Author} does not -> {Book}
- {Author} -> {Author_Age}

Therefore as per rule of transitive dependency, we get

- {Book} -> {Author_Age} should hold that makes sense because if we know the book name, we can know the author's age.

Closure Set of Attributes Dependencies

It is the set of all attributes that can be computed using given attributes. Closure of attribute set {X} is denoted as $\{X\}^+$.

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.

An algorithm to compute α^+, the closure of α under F is as follows :

```
result = $\alpha$
while (changes to result) do
    for each functional dependency $\beta \rightarrow \gamma$ in F do
    begin
        if $\beta \subseteq$ result then result = result $\cup\ \gamma$
    end
```

Let us consider R(A, B, C) and two functional dependencies :

A -> B

B -> C

Means, A -> BC

F - Total dependencies (Say)

We have

$F = F_1 + F_2$

Where,

F_1: A -> B and F_2:A -> C, which won't be visible initially. But whenever we find B and C is dependent on B means by transitivity rule A -> C.

Now $A^+ = \{A, B, C\}$ is the closure set of A.

$B^+ = \{B, C\}$ is the closure set of B.

$C^+ = \{C\}$ is the closure set of C.

Example : Consider an EMPLOYEE Table

EMPLOYEE:

<u>E-ID</u>	E-NAME	E-CITY	E-STATE
E001	John	Kathmandu	Kathmandu
E002	Mary	Kathmandu	Kathmandu
E003	John	Janakpur	Pokhara

Given R(<u>E-ID</u>, E-NAME, E-CITY, E-STATE)

FDs = { E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE }

The attribute closure of E-ID can be calculated as:

- Add E-ID to the set {E-ID}
- Add Attributes which can be derived from any attribute of set. In this case, E-NAME and E-CITY, E-STATE can be derived from E-ID. So these are also a part of closure.
- As there is one other attribute remaining in relation to be derived from E-ID. So result is:

$(\text{E-ID})^+$ = {E-ID, E-NAME, E-CITY, E-STATE }

Similarly,

$(\text{E-NAME})^+$ = {E-NAME}

$(\text{E-CITY})^+$ = {E-CITY, E_STATE}

Q. Find the attribute closures of given FDs R(ABCDE) = {AB->C, B->D, C->E, D->A}To find $(B)^+$,we will add attribute in set using various FD which has been shown in table below.

Attributes Added in Closure	FD used
{B}	Triviality
{B,D}	B->D
{B,D,A}	D->A
{B,D,A,C}	AB->C
{B,D,A,C,E}	C->E

- We can find $(C, D)^+$ by adding C and D into the set (triviality) and then E using(C->E) and then A using (D->A) and set becomes.

  $(C,D)^+$ = {C,D,E,A}
- Similarly we can find $(B,C)^+$ by adding B and C into the set (triviality) and then D using (B->D) and then E using (C->E) and then A using (D->A) and set becomes

  $(B,C)^+$ ={B,C,D,E,A}

Candidate Key

Candidate Key is **minimal set of attributes** of a relation which can be used to identify a tuple uniquely. For Example, each tuple of EMPLOYEE relation given in Table 1 can be uniquely identified by **E-ID** and it is minimal as well. So it will be Candidate key of the relation.
A candidate key may or may not be a primary key.

Super Key

Super Key is **set of attributes** of a relation which can be used to identify a tuple uniquely.For Example, each tuple of EMPLOYEE relation given in Table 1 can be uniquely identified by **E-ID or (E-ID, E-NAME) or (E-ID, E-CITY) or (E-ID, E-STATE) or (E_ID, E-NAME, E-STATE)** etc. So all of these are super keys of EMPLOYEE relation.
Note: A candidate key is always a super key but vice versa is not true.

Q. Finding Candidate Keys and Super Keys of a Relation using FD set The **set of attributes** whose attribute closure is set of all attributes of relation is called super key of relation. For Example, the EMPLOYEE relation shown in Table 1 has following FD set. **{E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE}**Let us calculate attribute closure of different set of attributes:

$(E\text{-}ID)^+$ = {E-ID, E-NAME,E-CITY,E-STATE}
$(E\text{-}ID,E\text{-}NAME)^+$ = {E-ID, E-NAME,E-CITY,E-STATE}
$(E\text{-}ID,E\text{-}CITY)^+$ = {E-ID, E-NAME,E-CITY,E-STATE}
$(E\text{-}ID,E\text{-}STATE)^+$ = {E-ID, E-NAME,E-CITY,E-STATE}
$(E\text{-}ID,E\text{-}CITY,E\text{-}STATE)^+$ = {E-ID, E-NAME,E-CITY,E-STATE}
$(E\text{-}NAME)^+$ = {E-NAME}
$(E\text{-}CITY)^+$ = {E-CITY,E-STATE}

As $(E\text{-}ID)^+$, $(E\text{-}ID, E\text{-}NAME)^+$, $(E\text{-}ID, E\text{-}CITY)^+$, $(E\text{-}ID, E\text{-}STATE)^+$, $(E\text{-}ID, E\text{-}CITY, E\text{-}STATE)^+$give set of all attributes of relation EMPLOYEE. So all of these are super keys of relation.

The **minimal set of attributes** whose attribute closure is set of all attributes of relation is called candidate key of relation. As shown above, $(E\text{-}ID)^+$ is set of all attributes of relation and it is minimal. So E-ID will be candidate key. On the other hand $(E\text{-}ID, E\text{-}NAME)^+$ also is set of all attributes but it is not minimal because its subset $(E\text{-}ID)^+$ is equal to set of all attributes. So (E-ID, E-NAME) is not a candidate key.

Example : Let us consider a relation R(A, B, C, G, H, I) and the set of functional dependencies are:

A -> B
A -> C
CG -> F
CG -> I
B -> H

We shall compute $(AG)^+$ using the algorithm on closure calculation.

{A , G}	
{A , B , G}	As A -> B
{A , B , C , G}	As A -> C
{A , B , C , G , H}	As CG -> H
{A , B , C , G , H , I}	As CG -> I

Here, AG can be super key. Super key can have some redundant attributes. So, it is not necessary that AG can be candidate key. So we have to prove AG are essential and sufficient.

If we take G -> all value not found
A -> all value not found

So, $(AG)^+$ is candidate key.

<u>**Closure Set of Functional Dependencies**</u>

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

Denoted by F^+

Armstrong axioms are set of rules used to find F^+.

Rule 1 - Reflexivity rule

If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$

Rule 2 - Augmentation rule

It states that addition of attributes in dependencies does not change the basic dependencies. That is, if a -> b holds and y is attribute set, then ay -> by also holds true.

Rule 3 - Transitivity

If a -> b holds and b -> c holds then a -> c also holds.

<u>**Additional Rules**</u>

Union Rule : If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\gamma$.

Decomposition Rule : If $\alpha \rightarrow \beta\gamma$ holds then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.

Pseudo-transitivity Rule : If $\alpha \rightarrow \beta$ and $\beta\gamma - \delta$ holds then $\alpha\gamma \rightarrow \delta$ holds.

Example : Consider a STUDENT table with functional dependencies :
STUD_NO->STUD_NAME, STUD_NO->STUD_ADDR hold.

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

How to find functional dependencies for a relation?

Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in Table 1.

- We know that STUD_NO is unique for each student. So STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY and STUD_NO -> STUD_AGE all will be true.
- Similarly, STUD_STATE->STUD_COUNTRY will be true as if two records have same STUD_STATE, they will have same STUD_COUNTRY as well.
- For relation STUDENT_COURSE, COURSE_NO->COURSE_NAME will be true as two records with same COURSE_NO will have same COURSE_NAME.

Functional Dependency Set: Functional Dependency set or FD set of a relation is the set of all FDs present in the relation. For Example, FD set for relation STUDENT shown in table 1 is:
{ STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY, STUD_NO -> STUD_AGE, STUD_STATE->STUD_COUNTRY }

Suppose we are given a relation scheme $R=(A,B,C,G,H,I)$, and the set of functional dependencies:

$A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H$

Then the functional dependency A -> H is logically implied.

Applying these rules to the scheme and set F mentioned above, we can derive the following:

- $A \rightarrow H$, as we saw by the transitivity rule.
- $CG \rightarrow HI$ by the union rule.
- $AG \rightarrow I$ by several steps:
  - Note that $A \rightarrow C$ holds.
  - Then $AG \rightarrow CG$, by the augmentation rule.
  - Now by transitivity, $AG \rightarrow I$.

(You might notice that this is actually pseudo-transitivity if done in one step.)

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes. For Example, STUD_NO in STUDENT relation is prime attribute, others are non-prime attribute.

Canonical Cover of F (Irreducible set of FD/Canonical Set/ Canonical Form)

Minimal set of functional dependencies equivalent to F which does not have redundant dependencies or redundant parts of dependencies.

In an irreducible set of functional dependency, we try to reduce all the transactions to less waste of the set of attributes. We have to follow some steps to decompose the set of the attribute in functional dependency:

- Decompose all possible right side attribute not left side attribute.
- Find closure of all the transaction after decomposition of attribute including and excluding the same transaction.
- If any changes are done in closure set of the attribute after including and excluding the same transaction; then we can't ignore the transaction otherwise, we ignore the transaction if the closure of that transaction is same in both cases.

- Follow this process in all decompose transactions then after check closure of the transactions we have after follow the aforesaid steps; if their closure is different then we can say that the transaction is in a reducible form otherwise, follow the steps again.

Example : R(w , x , y , z)
FD : x -> w
wz -> xy
y -> wxz

If we have α -> β, 3 possible cases to show redundancy, where α side may have redundancy or β side may have redundancy or full functional dependency is extra.

Decompose all the transactions:

x --- > w
wz --- > x
wz --- > y
y --- > w
y --- > x
y --- > z

Here, Now, find closure of all the decompose transactions:

1) x --- > w

- **Including the transaction**
  $\{x\}^+=\{x,w\}$
- **Excluding the transaction**
  $\{x\}^+=\{x\}$

Closure are different in both cases so we can't ignore this transaction

2) wz --- > x

- **Including the transaction**
  $\{wz\}^+=\{x,y,w,z\}$
- **Excluding the transaction**
  $\{wz\}^+=\{w,x,y,z\}$

Closure is same in both cases so, we ignore this transaction.

3) wz --- > y

- **Including the transaction**
  $\{wz\}^+=\{x,y,w,z\}$
- **Excluding the transaction**
  $\{wz\}^+=\{w,z\}$

Closure is different in both cases so, we can't ignore this transaction.

4) y --- > w

- **Including the transaction**
  $\{y\}^+=\{x,y,w,z\}$
- **Excluding the transaction**
  $\{y\}^+=\{w,x,y,z\}$

Closure are same in both cases so we ignore this transaction.

5) y --- > x

- **Including the transaction**
  $\{y\}^+=\{x,y,w,z\}$
- **Excluding the transaction**
  $\{y\}^+=\{y,z\}$

Closure are different in both cases so we can't ignore this transaction.

6) y --- > z

- **Including the transaction**
  $\{y\}^+=\{x,y,w,z\}$
- **Excluding the transaction**
  $\{y\}^+=\{w,x,y\}$

Closure are different in both cases so we can't ignore this transaction.

Now, write all the transactions which we can't ignore:

x --- > w
wz --- > y
y --- > x
y --- > z

Find closure of each attribute of **wz --- > y** for cross check its value are same or different.

$\{wz\}^+=\{w,x,y,z\}$
$\{w\}^+=\{w\}$
$\{z\}^+=\{z\}$

Closure are different so now we can say that it is in the reducible form. The, the final transactions are:

x --- > w
wz --- > y
y --- > xz

x -> w	x -> w	x -> w required
wz -> xy	wz -> x *not required	wz -> y required
y -> wxy	wz -> y	y -> xz required
	y -> w	
	y -> x	
	y -> z	

<u>**Different Normal Forms (1st , 2nd , 3rd , BCNF , DKNF)**</u>

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

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, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

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 emp_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 emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

<u>**ADVANTAGES OF NORMALIZATION**</u>

Here we can see why normalization is an attractive prospect in RDBMS concepts.

- A smaller database can be maintained as normalization eliminates the duplicate data. Overall size of the database is reduced as a result.

- Better performance is ensured which can be linked to the above point. As databases become lesser in size, the passes through the data becomes faster and shorter thereby improving response time and speed.
- Narrower tables are possible as normalized tables will be fine-tuned and will have lesser columns which allows for more data records per page.
- Fewer indexes per table ensures faster maintenance tasks (index rebuilds).
- Also realizes the option of joining only the tables that are needed.

DISADVANTAGES OF NORMALIZATION

- More tables to join as by spreading out data into more tables, the need to join table's increases and the task becomes more tedious. The database becomes harder to realize as well.
- Tables will contain codes rather than real data as the repeated data will be stored as lines of codes rather than the true data. Therefore, there is always a need to go to the lookup table.
- Data model becomes extremely difficult to query against as the data model is optimized for applications, not for ad hoc querying. (Ad hoc query is a query that cannot be determined before the issuance of the query. It consists of an SQL that is constructed dynamically and is usually constructed by desktop friendly query tools.). Hence it is hard to model the database without knowing what the customer desires.
- As the normal form type progresses, the performance becomes slower and slower.
- Proper knowledge is required on the various normal forms to execute the normalization process efficiently. Careless use may lead to terrible design filled with major anomalies and data inconsistency.

De-normalization

De-normalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that de-normalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization.

In a traditional normalized database, we store data in separate logical tables and attempt to minimize redundant data. We may strive to have only one copy of each piece of data in database. For example, in a normalized database, we might have a Courses table and a Teachers table. Each entry in Courses would store the teacherID for a Course but not the teacherName. When we need to retrieve a list of all Courses with the Teacher name, we would do a join between these two tables.

In some ways, this is great; if a teacher changes is or her name, we only have to update the name in one place.

The drawback is that if tables are large, we may spend an unnecessarily long time doing joins on tables.

De-normalization, then, strikes a different compromise. Under de-normalization, we decide that we’re okay with some redundancy and some extra effort to update the database in order to get the efficiency advantages of fewer joins.

Pros of De-normalization:-

- Retrieving data is faster since we do fewer joins
- Queries to retrieve can be simpler (and therefore less likely to have bugs), since we need to look at fewer tables.

Cons of De-normalization:-

- Updates and inserts are more expensive.
- De-normalization can make update and insert code harder to write.
- Data may be inconsistent . Which is the “correct” value for a piece of data?
- Data redundancy necessities more storage.

In a system that demands scalability, like that of any major tech companies, we almost always use elements of both normalized and de-normalized databases.

Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First Normal Form

For a table to be in the First Normal Form, it should follow the following 4 rules:

- It states that it should only have single(atomic) valued attributes/columns.
- Every values stored in a column should be of the same domain
- Every columns in a table should have unique names.
- And the order in which data is stored, does not matter.
- It ensures that each relation has a primary key.

Here is our table, with some sample data added to it.

roll_no	**Name**	**subject**
101	A	OS, CN
103	C	Java
102	B	C, C++

Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

How to solve this Problem?

It's very simple, because all we have to do is break the values into atomic values.

Here is our updated table and it now satisfies the First Normal Form.

roll_no	**Name**	**subject**
101	A	OS
101	A	CN
103	C	Java
102	B	C
102	B	C++

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row.

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form

A relation R is in 2NF iff

- R is in 1NF initially
- All non-prime attributes are fully dependent on the candidate key i.e. there should not be partial dependency.

Example : We have R (A , B , C , D) with functional dependencies : AB ---> D and B ---> C.

So closure set of $(AB)^+$ ---> ABCD

$(B)^+$ ---> BC

Here $(AB)^+$ can find all values of A, B, C and D and not any axon is coming towards AB. So AB is essential attributes set to find all attributes. Hence AB is candidate key.

Now attribute A and B belongs to candidate key attributes set, so A and B will be prime attributes. And remaining attribute C and D will be non prime attributes.

Here primary key/candidate key can't be NULL. So AB can't be NULL. But one of them can be NULL.

To translate relation R to 2NF, decompose R into R_1 and R_2. We get :

R_1 (A , B , D) and

R_2 (B , C)

Here if we check dependency , no partial dependency present.

Example : Let's take a **Student** table with columns **student_id, name, reg_no (registration number), branchand address(student's home address).**

In this table, student_id is the primary key and will be unique for every row, hence we can use student_id to fetch any row of data from this table

Even for a case, where student names are same, if we know the student_id we can easily fetch the correct record.

student_id	**name**	**reg_no**	**branch**	**address**
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

I can ask from branch name of student with student_id **10**, and I can get it. Similarly, if I ask for name of student with student_id **10** or **11**, I will get it. So all I need is student_id and every other column **depends** on it, or can be fetched using it.

This is **Dependency** and we also call it **Functional Dependency**.

What is Partial Dependency?

Now that we know what dependency is, we are in a better state to understand what partial dependency is.

For a simple table like Student, a single column like student_id can uniquely identfy all the records in a table.

But this is not true all the time. So now let's extend our example to see if more than 1 column together can act as a primary key.

Let's create another table for **Subject**, which will have subject_id and subject_name fields and subject_id will be the primary key.

subject_id	**subject_name**
1	Java
2	C++
3	Php

Now we have a **Student** table with student information and another table **Subject** for storing subject information.

Let's create another table **Score**, to store the **marks** obtained by students in the respective subjects. We will also be saving **name of the teacher** who teaches that subject along with marks.

score_id	**student_id**	**subject_id**	**marks**	**teacher**
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

In the score table we are saving the **student_id** to know which student's marks are these and **subject_id** to know for which subject the marks are for.

Together, student_id + subject_id forms a **Candidate Key**(learn about Database Keys) for this table, which can be the **Primary key**.

Confused, How this combination can be a primary key?
See, if I ask you to get me marks of student with student_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject_id, you would not know for which student. Hence we need student_id + subject_id to uniquely identify any row.

But where is Partial Dependency?

Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.
Now as we just discussed that the primary key for this table is a composition of two columns which is stuent_id & subject_id but the teacher's name only depends on subject, hence the subject_id, and has nothing to do with student_id.
This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?

There can be many different solutions for this, but out objective is to remove teacher's name from Score table.
The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

subject_id	**subject_name**	**teacher**
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

And our Score table is now in the second normal form, with no partial dependency.

score_id	**student_id**	**subject_id**	**marks**
1	10	1	70
2	10	2	75
3	11	1	80

Quick Recap

- For a table to be in the Second Normal form, it should be in the First Normal form and it should not have Partial Dependency.
- Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.
- To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

Third Normal Form

A relation R can be in 3NF iff

- R is already in 2NF.
- It should not have any transitive dependency.

To translate the relation to 3NF, relation must already be in 2NF. Then transitive dependency can be eliminated between non-key attributes and prime attributes.

Let us take an example

R(A , B , C , D) with functional dependency :AB --> C and C --> D.

Here again AB are essential attributes .

$(AB)^+ = ABCD$

Here AB is a candidate key. So, A and B are prime attributes and C & D are non-prime attributes.

Here C ---> D shows transitive dependency.

[Note : A relation R is in 3NF iff it does not have partial and transitive dependency.]

Here , C & D are both non-prime attributes. It means when a non-prime attribute finds another non-prime attribute that is the case of transitive dependency. Because C is non-prime attribute that means at some instant C can be NULL too. It means there can be problem in finding the value of D because D is totally dependent on non-prime attribute C and C is dependent on prime attribute (Candidate Key) AB.

To translate relation to 3NF.We have to divide relation in 2 different relations.

R_1 (A , B , C) and R_2 (C , D)

Where AB determines C and C determines D.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	P	Kathmandu	A
1002	Ajeet	222008	Q	Kalimati	B
1006	Lora	282007	Q	Kalimati	C
1101	Lilly	292008	R	Teku	D
1201	Steve	222999	S	Kalanki	E

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys : {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.
Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	P	Kathmandu	A
222008	Q	Kalimati	B
282007	Q	Kalimati	C
292008	R	Teku	D
222999	S	Kalanki	E

Boyce - Codd Normal Form

It is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- And, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Let us take an example

R(A , B , C) with functional dependencies : AB ---> C and C ---> B

Now take $A^+ =$....
$(AB)^+ = ABC$
$(AC)^+ = ABC$

Here AB ---> C
C ---> B

First check partial dependency and transitive dependency.
There is no partial and transitive dependency present in R.
In BCNF if there is any dependency from α ---> β then α must be super-key.
So here in R(A , B , C), both AB and AC can be candidate key.

To translate R in BCNF, decompose R into R_1(C , B) and R_2(A , C). Here decomposition will be lossless because AC is taken as candidate key. Here we can see, AB ---> C is not seen in child relation that means this dependency is lost.
This shows that the dependency is not dependency preserving.
Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id --> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}
Candidate key: {emp_id, emp_dept}
The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
Stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

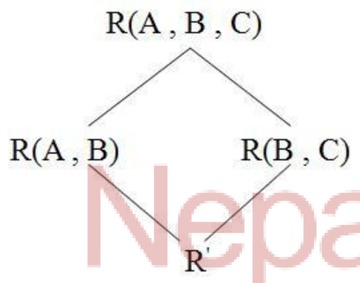
Functional dependencies :
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}
This is now in BCNF as in both the functional dependencies left side part is a key.

Decomposition

A relation(R) can be decomposed into a collection of schemes to eliminate some of the anomalies in original relation(R). It should always be lossless because it confirms that the information in the original relation can be accurately reconstructed based n the decomposed relation. If there is not proper decomposition of the relation then it may lead to problems like loss of information.

Example : Given relation R(A , B , C) then



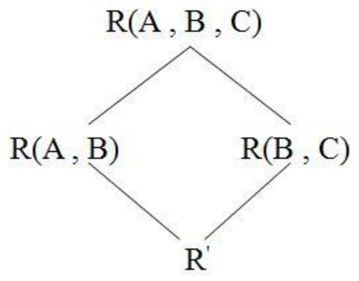
If R=R' then decomposition is lossless decomposition.

Why decomposition is required in Normalization?

- Redundancy can be removed.
- Data independence can be maintained.
- Anomalies can be reduced.

Properties of Decomposition

- Attributes preservation



From above decomposition, we can see relation R_1(A , B) and R_2(B , C) have the attributes A , B and C (all attributes preserved from original relation(R)).

- Lossless join decomposition

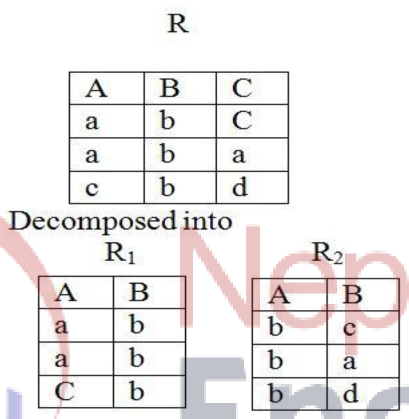
  Conditions for lossy and lossless decomposition

Lossless	Lossy
R_1 |X| R_2 =R	R_1 |X| $R_2 \subset$ R (Subset)
	R_1 |X| R_2 $\supset$ R (Superset)
	R_1 |X| R_2 = Φ (Empty)

Let R is a relation and has set of functional dependency F over R. The decomposition of R into R_1 and R_2 is lossless with respect to F if R_1 |X| R_2 = R.

[Note : If ($R_1 \cap R_2$) forms a super key in either R_1 or R_2 then it is lossless decomposition.]

Example 1: R (A , B , C) is decomposed into R_1(A , B) and R_2(B , C)



R

A	B	C
a	b	C
a	b	a
c	b	d

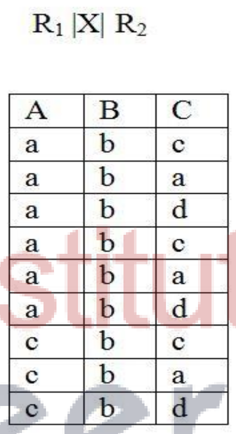
Decomposed into

R_1

A	B
a	b
a	b
C	b

R_2

A	B
b	c
b	a
b	d



R_1 |X| R_2

A	B	C
a	b	c
a	b	a
a	b	d
a	b	c
a	b	a
a	b	d
c	b	c
c	b	a
c	b	d

And here we get 9 numbers of rows in R_1 |X| R_2. So, it is the superset of R (Original relation) i.e. (R_1 |X| R_2) $\supset$ R. So decomposed relation is Lossy.
Lossy decomposition contains extra or less tuples.

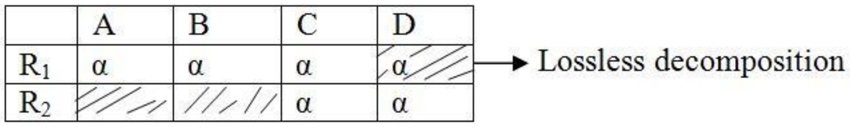
Example 2: R(A , B , C) decomposed into R_1(A , B) and R_2(B , C). Check whether lossless or lossy.
Given functional dependency of R (A ---> B)

Here, ($R_1 \cap R_2$) =B which is not a super key either in R_1 or R_2
Hence decomposition is lossy.

Example 3 : R(A , B , C) decomposed into R_1(A , B) and R_2(A , C). Check whether lossless or lossy.
Given functional dependency of R (A ---> B)

Here, ($R_1 \cap R_2$) =A which is a super key either in R_1 or R_2
Hence decomposition is lossless.

Example 4 : R(A , B , C , D) decomposed into R_1(A , B , C) and R_2(C , D). Check whether lossless or lossy.
Given functional dependency of R: {A ---> B , A ---> C , C ---> D)
Draw matrix table :



	A	B	C	D
R_1	α	α	α	α
R_2	▨	▨	α	α

Lossless decomposition

Here matrix table should have at least one row fulfilled and here this table has one row fulfilled. So decomposition is lossless.

Example 5 : R(A , B , C , D , E) into R_1(A , B , C) , R_2(B , C , D) and R_3(C , D , E) with functional dependency F : {AB ---> CD , A ---> E , C ---> D}
Matrix table :

	A	B	C	D	E
R_1	A	α	α	α	▨
R_2	▨	α	α	α	▨
R_3	▨	▨	α	α	α

No row will have all α values . So, lossy decomposition.

Dependency Preservation

The decomposition of relation R having functional dependency set F into two tables R_1 and R_2 having functional dependency set F_1 and F_2 respectively is said to be dependency preserving if,

$(F_1 \cup F_2)^+ = F^+$

Example 1: R(A , B , C) , F ={A ---> B , B ---> C}, decomposed into R_1(A , C) and R_2(B , C).
Here A^+ = ABC and B^+ = BC
For R_1
F_1 : A ---> C (Due to A ---> B and B ---> C, Transitivity)
For R_2
F_2 : B ---> C
But A ---> B does not hold.
Now $(F_1 \cup F_2)^+ \; != F^+$
Hence decomposition is not dependency preserving.

Example 2: R(A , B , C) , F ={A ---> B , B ---> C}, decomposed into R_1(A , B) and R_2(B , C).
Here A^+ = ABC and B^+ = BC
For R_1
F_1 : A ---> B (Due to A ---> B and B ---> C, Transitivity)
For R_2
F_2 : B ---> C
Now $(F_1 \cup F_2)^+$ = {A ---> B , B ---> C} = F^+
Hence decomposition is dependency preserving

Example 2: R(city , street , zip-code) , F ={(city , street) ---> zip-code , zip-code ---> city}, decomposed into R_1(street , zip-code) and R_2(city , zip-code).
Here A^+ = ABC and B^+ = BC
For R_1
F_1 : {street ---> street , zip-code ---> zip-code } (Due to A ---> B and B ---> C, Transitivity)
For R_2
F_2 : {city ---> city , zip-code ---> zip-code , zip-code ---> city}
Now $(F_1 \cup F_2)^+$ = {zip-code ---> city} != F^+
Hence decomposition is not dependency preserving.

Example 3 : Let R(A , B , C , D , E) with functional dependency {A ---> BC , CD ---> E , B ---> D , E ---> A} is decomposed into R_1(A , B , C) and R_2(A , D , E)
Here A ---> BC gives A ---> B and A ---> C. A decomposition {R_1 , R_2} is said to be lossless join decomposition when R_1 R_2 --> R_1 or R_1 R_2 --> R_1.
For this case , we use given decomposed schema R_1 and R_2.
This means that R_1 = {A , B , C} and R_2 ={A , D , E}.
Now $R_1 \cap R_2$ = A
When A is the candidate key, it can be implied that R1 $\cap$ R_2 = R_1 but not otherwise.
Hence, the given decomposition is lossless decomposition when A is the candidate key.
Alternative,
We have ,
A --> BC gives A--> B and A --> C
Since A -->B and B --> D , A ---> D (Decomposition , Transitivity)
Since A --> CD and CD --> E, A --> E (Union , Decomposition , Transitivity)
Since A--> A, we have (Reflexivity)
A --> ABCDE from above steps (Union)
Since E --> A , E --> ABCDE (Transitivity)
Since CD --> E , CD --> ABCDE (Transitivity)
Since B --> D and BC --> CD , BC --> ABCDE (Augmentation , Transitivity)
Also,
C --> C , D --> D , BD --> D etc.
Therefore any functional dependency with A , E , BC or CD on the left hand side give the arrow is in F^+ , no matter which other attributes appear in the functional dependency. Here the candidate keys are A , BC , CD and E.
Now for the decomposition {R_1 , R_2} to be lossless decomposition, the condition is
$R_1 \cap R_2 = R_1$ or $R_1 \cap R_2 = R_2$
So here let R_1 =(A , B , C) and R_2 (A , D , E) then $R_1 \cap R_2$ = A
As A is the candidate key, we get $R_1 \cap R_2 = R_1$. Hence the decomposition is lossless.

Dependency Preservation

We have R(A , B , C , D , E) decomposed into R_1(A , B , C) and R_2(A , D , E) with functional dependencies : F : {A ---> BC , CD ---> E , B ---> D , E ---> A}

Then for R_1(A , B , C)

F_1 : {A --> BC}

For R_2(A , D , E)

F_2 : {A --> DE , E --> A}

Now , $(F_1 \cup F_2)^+$ ={A --> BC , A --> DE , E --> A} != F^+

Hence **dependency is not preserved**.

Multi-valued and Joined Dependency

Fourth Normal Form

A relation R is in 4NF iff the following conditions are satisfied :

- R is already in 3NF or BCNF
- If it contains no MVDs.

Multi-valued Dependency

It is the dependency where one attribute value is potentially a multi-valued fact about another.

Conditions for MVD to exists.

- There must be 3 or more than 3 attributes.
- Attributes must be independent of each other.

[Functional dependency (A ---> B) says we can't have two tuples with same A value but different B value]

But MVD says for same value of A there can be two or more different value of B.

MVD denoted by :

A -->--> B

Example : We have

Person(P)	Mobile(M)	Food_Like(F)
P_1	M_1 , M_2	F_1 , F_2
P_2	M_3	F_2

Now

Person(P)	Mobile(M)	Food_Like(F)
P_1	M_1	F_1
P_1	M_1	F_2
P_1	M_1	F_2
P_1	M_2	F_1
P_2	M_3	F_2

Here P -->--> M and P -->--> F are MVDs.

And food and mobile are independent attributes here.

Definition : Any relation R where A and B are the subsets of R. MVD : A --->--> B holds if all pairs of t_1 and t_2 in R such that $t_1[A] = t_2[A]$, then there exists tuple t_3 and t_4 in R such that

$t_3[A] = t_4[A] = t_1[A] = t_2[A]$

$t_3[B] = t_1[B]$

$t_4[B] = t_2[B]$

MVD occurs if two or more independent relations are kept in a single relation. Consider two relations R_1(sid , sname) and R_2(cid , cname)

If we put $R_1$and R_2 in single relation then there can be the possibility of MVD.

R_1 :

sid	Sname
s_1	A
s_2	B

R_2 :

cid	Cname
c_1	C
c_2	B

Merging using Cross Product

sid	Sname	cid	cname
s_1	A	c_1	C
s_2	A	c_2	B
s_2	B	c_1	C
s_2	B	c_2	B

Now we can see that

MVD : {sid -->---> cid , sid -->--> cname}

Example 2 : Consider the following table student (name , computer , language) with the following records.

name	computer	Language
Aman	Windows , Apple	English , Nepali
Mohan	Linux	English , Hindi

Normalize the table to 4NF

First we have to make attribute single valued then we do

name	computer	Language
Aman	Windows	English
Aman	Windows	Nepali
Aman	Apple	English
Aman	Apple	Nepali
Mohan	Linux	English
Mohan	Linux	Hindi

MVD : name -->--> computer

name -->--> language

To remove MVD we have to decompose R into R_1 and R_2. Where
R_1 (name , computer)

name	computer
Aman	Windows
Aman	Apple
Aman	Linux

Here, Key is composite key (name , computer)
R_2 (name , language)

name	computer
Aman	English
Aman	Nepali
Mohan	English
Mohan	Hindi

Key is composite key (name , language)
Next
For given relation R and set of MVDs for R, R is in 4NF with respect to its MVDs if for every non-trivial MVD,

$A_1 , A_2 , A_3 \ldots A_n$ -->--> $B_1 , B_2 , B_3 \ldots B_n$

Where $A_1 , A_2 , A_3 \ldots. A_n$ is a super key.

Fifth Normal Form (5NF)
It is based on joined dependency.
A relation R is in 5NF iff the following conditions are satisfied simultaneously
- R is already in 4NF (No MVD)
- It can't be further non-loss decomposed i.e. if we can further decompose the relation losslessly then that relation can't be in 5NF.

Join dependency
Let R be a relation schema and $R_1 , R_2 , R_3 , \ldots , R_n$ be the decomposition of R, then Ris said to satisfy the join dependency.

$(R_1 , R_2 , R_3 , \ldots , R_n)$ iff

$\pi_{R1}(R)$ |X| $\pi_{R2}(R)$ |X| .... |X| $\pi_{Rn} = R$

[Order of join does not matter]
We can decompose the relation to lossless decomposition up to n number of relation.
or
Iff every legal instance r(R) is equal to join of its projections on $R_1 , R_2 , \ldots , R_n$.
Example 1: We have

Agent	Company	Product
Aman	C_1	Pendrive
Aman	C_1	Mic
Aman	C_2	Speaker
Mohan	C_1	Speaker

Decomposes in to R_1(Agent , Company) and R_2(Agent , Product)
If we have to get information that which company sell which product. Then
We have to perform natural join between R_1 and R_2 i.e. R_1 |X| R_2 = R if we get then we can say that there is join dependency.
If R_3(Company , Product)
Again we find
R_1 |X| R_2 |X| R_3 = R if we get this. There is join dependency.
And it is violating the concept of 5NF.

R_1

Agent	Company
Aman	C_1
Aman	C_2
Mohan	C_1

R_2

Agent	Product
Aman	Pendrive
Aman	Mic
Aman	Speaker
Mohan	Speaker

R_3

Company	Product
C_1	Pendrive
C_1	Mic
C_1	Speaker
C_2	Speaker

Now R_1 |X| R_2 is

Agent	Company	Product
Aman	C_1	Pendrive
Aman	C_1	Mic
Aman	C_1	Speaker
Aman	C_2	Pendrive
Aman	C_2	Mic
Aman	C_2	Speaker
Mohan	C_1	Speaker

After comparing with R, decomposition is lossy.
Again we computer R_1 |X| R_2 |X| R_3

Agent	Company	Product
Aman	C_1	Pendrive
Aman	C_1	Mic
Aman	C_1	Speaker
Aman	C_2	Speaker
Mohan	C_1	Speaker

Extra

We get ,
R_1 |X| R_2 |X| R_3 != R. Relation R is in 5NF. There is not join dependency.

Example 2: A relation R is in 5NF with respect to a set of F of functional , multi-valued and join dependencies if for every non-trivial join dependency (R_1 , R_2 , ... , R_n) in F^+, every R_1 is a super key of R.

R :

Pname	Skill	Job
Aman	DBA	J_1
Mohan	Tester	J_2
Rohan	Programmer	J_3
Sohan	Analyst	J_1

R_1 :

Pname	Skill
Aman	DBA
Mohan	Tester
Rohan	Programmer
Sohan	Analyst

R_2 :

Pname	Job
Aman	J_1
Mohan	J_2
Rohan	J_3
Sohan	J_1

R_3 :

Skill	Job
DBA	J_1
Tester	J_2
Programmer	J_3
Analyst	J_1

R_1 |X| R_2 :

Pname	Skill	Job
Aman	DBA	J_1
Mohan	Tester	J_2
Rohan	Programmer	J_3
Sohan	Analyst	J_1

We get R_1 |X| R_2 = R

There is join dependency and all R_1 , R_2 and R_3 are not key of R.

Hence R is not in 5NF.

<u>Domain Key Normal Form (DKNF / 6NF)</u>

A relation will be in DKNF iff every content of the relation is a logical consequence or domain constraints or key constraints i.e. each and every data in database must be logical consequences of key of domain constraints.

Key Constraints

Using key we can identify tuples uniquely. Each attribute participating in construction of key must be NOT NULL. Each and every attribute in key must have some value in each tuple.

Domain Constraints

It means from which set of values, the values should be picked up to instantiate one attribute. So in case of foreign key, we got the domain constraint. So while defining one database, all the values must be a logical consequence of domain constraints and key constraints. It has been insured in DKNF. DKNF and 6NF are synonymous.

In 6NF, it is intended to decompose relation variables to irreducible components.

Example : In 5NF the following relation

Student_table : (roll_no , dept , grade) is implied by the candidate key.

In more specific, the only possible join dependencies are {roll_no , dept} and {roll_no , grade}.

So in 6NF, the respective version will look like:

Student_details : (roll_no , dept)

Result_details : (roll_no , grade)

<u>Normalization Practice</u>

<u>1NF</u>

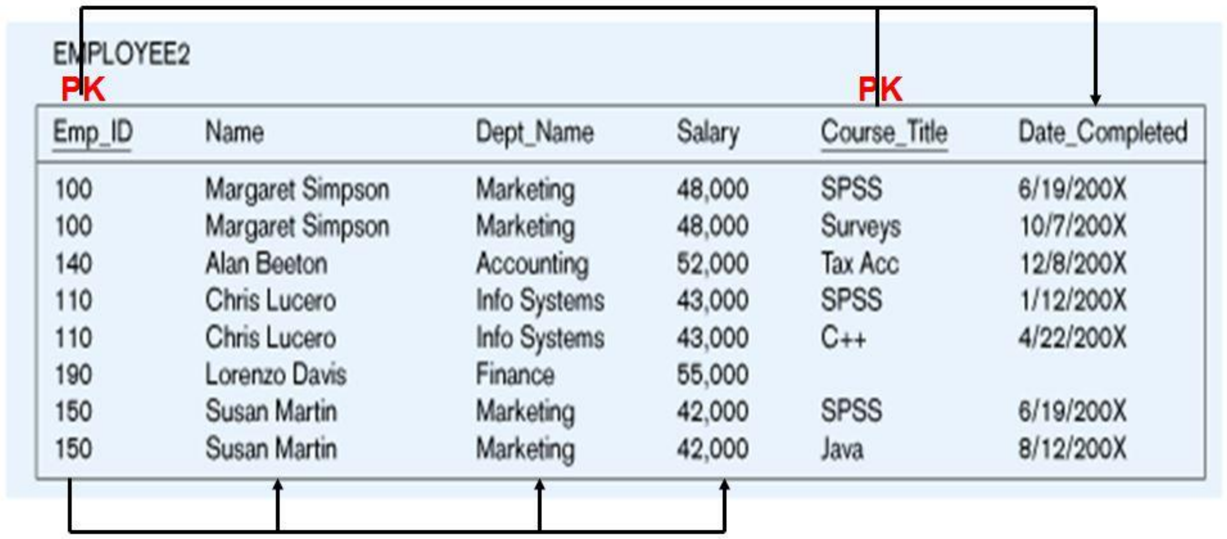
PK

Customer_ID	Name	Address	Company Name
104	Mr. Ray Suchecki	123 Pond Hill Road, Detroit, MI, 48161	CSUB
624	Mr. Toby Stein	431 North Phillips Road, South Bend, IN, 46611	Bakersfield
627	Mr.Gilbert Scholten	3915 Hawthorne Avenue, Toledo, OH, 43603	BPA

PK

Student_ID	Advisor	Advisor_Office	Class_1	Class_2	Class_3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01

2NF


EMPLOYEE2
PK
PK

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Functional Dependencies in EMPLOYEE2

EMPLOYEE2(<u>EmpID</u> , <u>CourseTitle</u> , Name , DeptName , Salary , DateCompleted)

Dependencies are :

{(EmpID , CourseTitle) ---> DateCompleted}

{EmpID ---> Name , DeptName , Salary}

Q. Define functional dependency of above table using a functional dependency notation, then normalize to 2NF.

Q. Define functional dependencies of table using functional dependency notation then normalize to 2NF

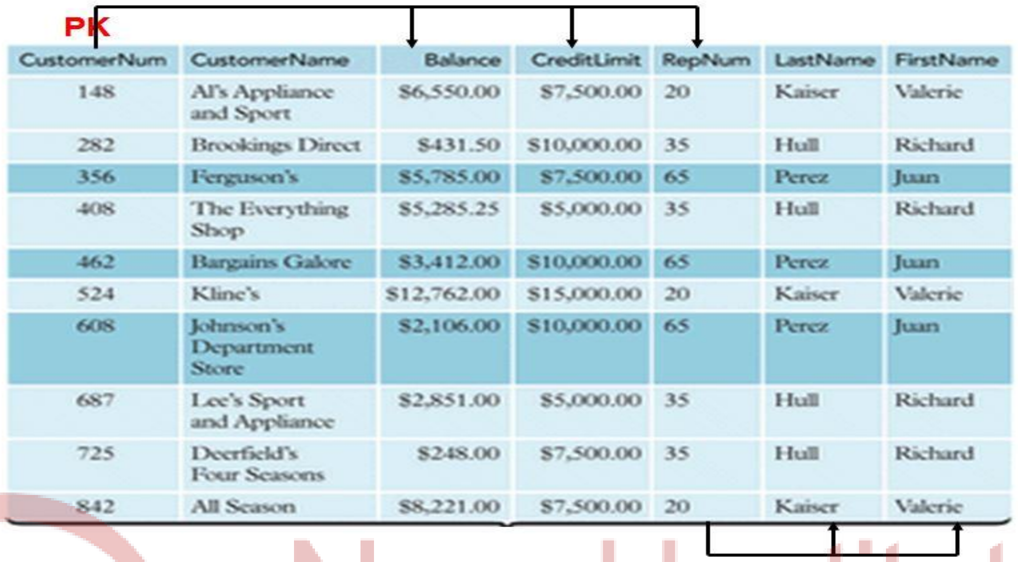
RentalInfo(TransID , VideoID , Copy , Title , Rent) with dependencies :

{(transID , VideoID) ---> Copy}

{VideoID ---> Title , Rent}

3NF

Q. Define functional dependencies of below table using a functional dependency notation, then normalize to 3NF



CustomerNum	CustomerName	Balance	CreditLimit	RepNum	LastName	FirstName
148	Al's Appliance and Sport	$6,550.00	$7,500.00	20	Kaiser	Valerie
282	Brookings Direct	$431.50	$10,000.00	35	Hull	Richard
356	Ferguson's	$5,785.00	$7,500.00	65	Perez	Juan
408	The Everything Shop	$5,285.25	$5,000.00	35	Hull	Richard
462	Bargains Galore	$3,412.00	$10,000.00	65	Perez	Juan
524	Kline's	$12,762.00	$15,000.00	20	Kaiser	Valerie
608	Johnson's Department Store	$2,106.00	$10,000.00	65	Perez	Juan
687	Lee's Sport and Appliance	$2,851.00	$5,000.00	35	Hull	Richard
725	Deerfield's Four Seasons	$248.00	$7,500.00	35	Hull	Richard
842	All Season	$8,221.00	$7,500.00	20	Kaiser	Valerie

CustomerNum ---> Balance , CreditLimit , RepNum
RepNum ---> LastName , FirstName

BCNF

Q. Define functional dependencies of below table using a functional dependency notation, then normalize to BCNF

Project (<u>ProjectID</u> , <u>ManagerID</u> , ManagerDept , ManagerLname , ProjectBudget , ProjectLocation)

Dependencies :

{(ProjectID , ManagerID) ---> (ManagerLname , ProjectBudget , ProjectLocation)}
{ManagerDept ---> ManagerID}