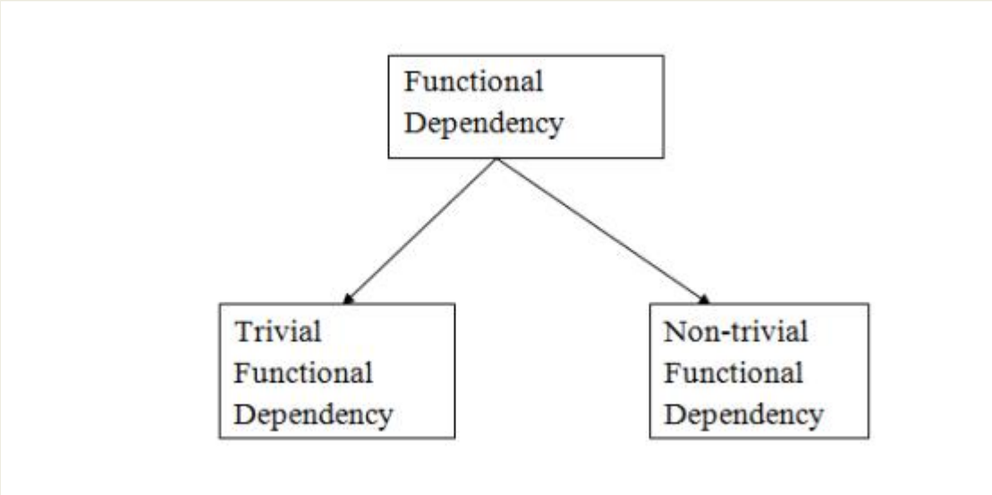
Different types of Functional Dependency

1. Functional Dependency
2. Multivalued Dependency
3. Join Dependency



Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

1. Functional Dependency

A functional dependency(FD) is a kind of IC that generalizes the concept of a key.

Definition: Let R be a relation schema and let X and Y be **nonempty** sets of attributes in R. We say that an instance **r** of R satisfies the FD X->Y(X determines Y) if the following holds for every pair of tuples t1 and t2 in R.

If t1.X=t2.X, then t1.Y=t2.Y

The notation t1.X refers to the projection/column of tuple t1 onto the attribute in X.

An FD X->Y says that if two tuples agree on the values in attributes X, they must agree on the values in attributes Y.

Example of the FD :

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

- The FD AB->C by showing an instance that satisfies this dependency.
- The 1st two tuples show that a FD is not the same as a key constraint: although the FD is not violated, AB is clearly not a key for the relation.
- The 3rd and 4th tuples illustrates that if two tuples differ in either the A field or the B field, they can differ in C field without violating the FD.
- If we add a tuple<a1, b1, c2,d1> to the instance of relation R, the resulting instance would violate the FD; to see the violation, compare the 1st tuple with new tuple

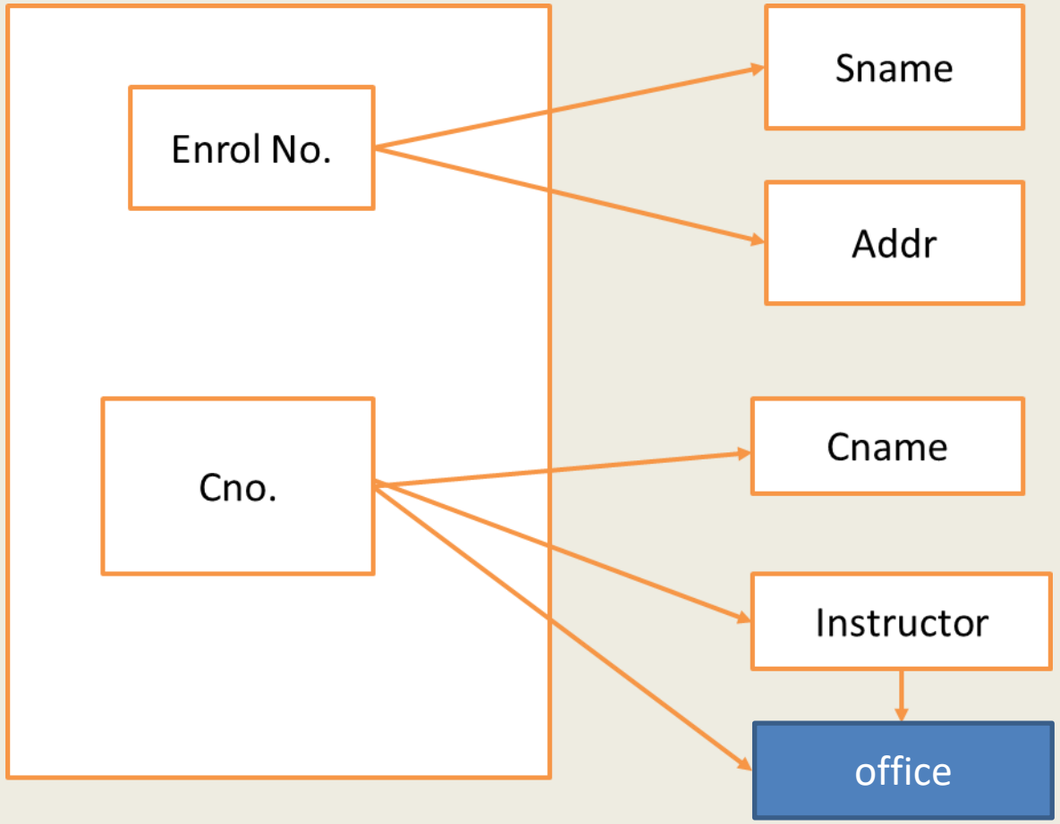
Note: Recall that a legal instance of a relation must satisfy all specified ICs, including all specified FDs. However , we can never deduce that an FD does hold by looking at one or more instances of the relation, b'coz like other ICs , FD is a statement about all possible legal instances of the relation.

- A primary key constraint is a special case of an FD.

- The attributes in the key play the role of X(X is a Super key), and the set of all attributes in the relation plays the role of Y
- Example: let s is an instance of Relation Student.

Enrol No	Sname	Addr	Cno	Cname	Instructor	Office
123	Rahul	Ranchi	EID301	PS	Komal	102
123	Rahul	Ranchi	EID302	CO	Anurag	105
123	Rahul	Ranchi	EID304	DBMS	Preethi	103
341	Aparna	Bombay	EID 304	DBMS	Preethi	103
Null	Null	Null	EID 305	DAA	Sunil	105

The dependencies for the Relation Student are shown as:



- The dependencies are:

Enrol no.->Sname

Enrol no.->Addr

Cno->Cname

Cno->Instructor

Instructor->office

1.These FD imply that there can be only one student name for each Enrol no, only one address for each student and only one subject name for each Cno.

2. It is possible that several students may have the same name and several students may live in the same address.

- If we consider Cno->Instructor, the dependency implies that no subject can have more than one instructor. Functional Dependencies therefore places constraints on, what information the database may store.
- Consider the FDs:

Sname->Enrol no ----------(1)

Cname->Cno----------(2)

1.To see that whether the above FDs hold or not depends on the university or college whose database we are considering allows duplicate student names and course names.

2. If duplicate names are possible, then there is a possibility of two students having exactly the same name, the (1) does not hold.

3.If it was an enterprise policy to have unique course names the (2) holds.

For the above example, draw the ER diagram, the FDs exists are:

1.**FDs in Entities:**

Student entity:

Enrol no->Sname,Addr

Course Entity:

Cno->Cname

Instructor Entity:

Instructor->office

2.**FDs in Relationship:**

Enrols Relationship: many-many

Teachers Relationship:

Cno->Instructor

Instructor->Cno

Multi Valued Dependency

Multivalued Dependency (MVD) in DBMS

Let R be a Relation Schema and X and Y be subsets of the attributes of R, the MVD X->-> Y is said to hold over R if, in every legal instance r of R, each X value is associated with a set of Y values and this set is independent of the values in the other attributes.

Formally,

Given a relation Schema R with attributes (X, Y, Z), if the MVD X->->Y holds over R and Z=R-XY, the following must be true for every legal instance r of R:

If t1∈r, t2∈r and t1.X=t2.X, then there must be some t3∈r such that t1.XY=t2.XY and t2.Z=t3.Z

If it satisfies the above condition then we can say MVD, X->->Y holds on R and Z and Y are independent to each other

Note: To remove this redundancy, we use Fourth Normal Form.

X	Y	Z	
A	B1	C1	Tuple t1
A	B2	C2	Tuple t2
A	B1	C2	Tuple t3
A	B2	C1	Tuple t4

Example
Below we have a college enrolment table

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

In the table above, student with s_id1 has opted for two courses, Science and Maths, and has two hobbies, Cricket and Hockey and the two records for student with s_id1,wil l give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

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

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

- To satisfy 4th Normal Form – To make the above relation satisfy the4th normal form, we can decompose the table into 2 tables.

CourseOpted Table

s_id	course
1	Science
1	Maths
2	C#
2	Php

Hobbies Table,

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form. A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multivalued dependent columns are moved to separate tables.

Join Dependency

Join Dependencies (It is a generalization of MVDs)

Definition:

A join dependency (JD) ⋈ {R1, R2,……,Rn} is said to hold over a relation R if R1, ….., Rn is a lossless-join decomposition of R.

An MVD X->->Y over a relation R can be expressed as join dependency ⋈{XY, X(R-Y)}.

Example: **Assume an Employee table in which there is a relationship between the project and skill**.

create table EMP(eno number(5), pno varchar2(20), skill varchar2(10));

- insert into EMP values(1111,'Railway reservation','Oracle');
- insert into EMP values(1111,'Railway reservation','JSP');
- insert into EMP values(2222,'Railway reservation','Oracle');
- insert into EMP values(1111,'library management','PHP');
- **select * from EMP;**

ENO	PNO	SKILL
1111	library management	PHP
1111	Railway reservation	Oracle
2222	Railway reservation	Oracle
1111	Railway reservation	JSP

- An employee may have skills appropriate for a project and he/she may have skills which are not required in a project. So we split the Employee table into two tables as

1. EMP_project (eno, project)
2. EMP_skill(eno, skill)

create table EMP_project(eno number(5),project varchar2(20))

- insert into EMP_project values(1111,'Railway reservation')
- insert into EMP values(1111,'Railway reservation','Oracle');
- insert into EMP_project values(1111,'library management')
- insert into EMP_project values(2222,'Railway reservation')
- select * from EMP_project;

ENO	PROJECT
1111	library management
1111	Railway reservation
2222	Railway reservation

```
create table EMP_Skill(eno number(5),skill varchar2(20));

insert into EMP_Skill values(1111,'Oracle’);

insert into EMP_Skill values(1111,'JSP’);

insert into EMP_Skill values(1111,'PHP’);

insert into EMP_Skill values(2222,'Oracle’);

select * from EMP_Skill;
```

ENO	SKILL
1111	JSP
2222	Oracle
1111	PHP
1111	Oracle

- select * from EMP_project p natural join EMP_Skill s;
- **natural join table** **original table**

ENO	PROJECT	SKILL
1111	library management	JSP
1111	Railway reservation	JSP
2222	Railway reservation	Oracle
1111	library management	PHP
1111	Railway reserve	PHP
1111	library management	Oracle
1111	Railway reserve	Oracle

ENO	PNO	SKILL
1111	library management	PHP
1111	Railway reservation	Oracle
2222	Railway reservation	Oracle
1111	Railway reservation	JSP

In **Natural Join** table we got 6 records with **eno:1111** which does not exist in the original table. This problem occurred because we lost the relationship between the project and skills by splitting the table into two projections.

Note: the original table is in 4NF but it cannot be splitted into two projections and then recombined to produce the original table, this table can be normalized to fifth normal form(5NF).

Fifth Normal Form(based on Join Dependency):

Let R be a relation Schema and R1, R2,….,Rn be the decomposition of R, R is said to satisfy the join dependency iff

$$\pi_{R1}\ (R) \bowtie \pi_{R2}\ (R) \bowtie \pi_{R3}\ (R) \ldots\ldots \bowtie \pi_{Rn}\ (R) = R$$

OR

A relation schema R is said to be in fifth normal form (5NF) if, for every JD ⋈ {R1,R2, …., Rn} that holds over R, one of the following statement is true:

R_i= R for some i, or

The JD is implied by the set of those FDs over R in which the left side is a key for R.

Explanation of 2nd statement:

The decomposition of R into {R1, R2, …, Rn} is lossless-join whenever the key dependencies hold. JD ⋈ {R1,R2, …., Rn} is a trivial JD if R_i= R for some i; such a JD always holds.

Example:

So to get back to the original table , we need to join the below three tables .

First , we will join EMP_Project and EMP_Skill based on the common key eno and the result of this is joined with project_skill based on project and skill.EMP_Project(eno, project)

EMP_Skill(eno, skill)

project_skill(project, skill)

OUTPUT after joining

select * from project_skill ps natural join EMP_Skill s natural join EMP_project p;

Result is loss-less join

ENO	PROJECT	SKILL
1111	library management	PHP
1111	Railway reservation	JSP
1111	Railway reservation	Oracle
2222	Railway reservation	Oracle

Properties of Decomposition

1) Lossless / lossy join decomposition
2) Dependency preserving decomposition

Lossless / lossy join decomposition: This property says that extra tuple or less tuple generation problem does not occur after decomposition. Let's explain with an example.

We have a relation R

R

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

The relation R is divided randomly into R1(AB) and R2(BC).

R1

A	B
1	2
2	2
3	3

R2

B	C
2	1
2	2
3	2

We have divided the relation R, we should have some common attribute between two tables to query it. So here B is a common attribute.

Query: find the value of C, if the value of A=1.

Sol: select R2.C from R2 natural join R1 where R1.A=1;

Before it perform natural join, it performs cross product of two tables.

To that cross-product result natural join will be applied. (natural join means only common values between two tables will be retrieved)

Cross product (R1 X R2)

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

After applying natural join to the above cross product , we get the following result R11

R11

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

==→

Compare this with original table R

R

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

Observe in R11, for A=1, it has two values of C (1 and 2). But in original table A =1 means C has only one value. This indicates after joining extra tuples are added.

So, this is called lossy decomposition. Lossy does not mean that we are losing data, lossy is in terms of tuples, after joining tables there should be no extra tuples. Data inconsistency occurs. All the values in new table R11 are not valid.

Always we should get lossless decomposition when we join tables. So for that there is a rule to choose attribute which should be common between tables

Rule: common attribute should be a candidate key or super key of either R1 or R2 or both.

So, in above relation R, A attribute is a candidate key or primary key because it doesn't have duplicate values. So, A attribute should be common between tables.

R1

A	B
1	2
2	2
3	3

R2

A	C
1	1
2	2
3	2

If we apply natural join for the above tables. We get lossless decomposition, so no redundancy.

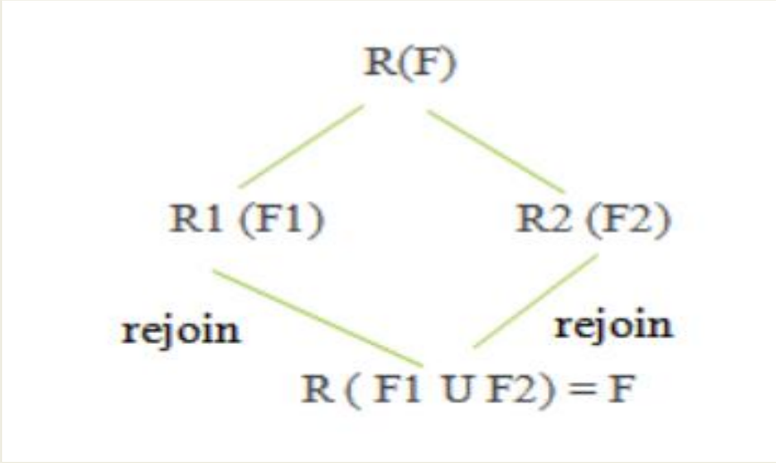
- Conditions are:

1. R1 U R2 = R (ex AB U AC = ABC)
2. R1 ∩ R2 = ϕ (ex AB ∩ AC = ϕ)
3. Common attribute should be candidate key or super key .

Dependency Preserving

The decomposition of relation R with FDs , F into R1 and R2 with FDs F1 and F2 respectively, is said to be dependency preserving if , (F1 U F2) + = F+

i.e A relation R is divided into R1 & R2, similarly functional dependencies is divided into F1 & F2. later when it is rejoined like R1 U R2 = R and F1 U F2 = F , should get the original table.



Example 1:

- Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.

A relation R is decomposed into – R1 = (A, B, C) with FDs F1 = {A -> B, A -> C}, and R2 = (C, D) with FDs F2 = {C -> D}.

F ' = F1 ∪ F2 = {A -> B, A -> C, C -> D}

so, F ' = F. And so, F‘ + = F+ .

Hence, dependency preserved means functional dependency is preserved even after decomposing and rejoining of tables.

Example 2:

If a Relation R (ABC) , F = { A -> B , B-> C } is decomposed into R1(AB) R2(BC) is this decomposition preserving the dependency or not?

Sol: R1 (AB) and R2(BC)

F1 = { A->A , B->B, A->B}

F2 = { B->B , C-> C, B->C }

(tip: choose trivial functional dependency)

(F1 U F2) = { A->B , B->C }

So, dependency is preserved in this relation