***Database System***

1. **Database** (Collection of related data)
   1. Structured data (In form of relation, table)
   2. Unstructured data
2. **DBMS** (Operations on database)
   1. Sql, mySql, Oracle vers., etc.
3. **RDBMS** (Same as DBMS but in tables(relational) form)

***File System vs DBMS***

1. **Data to be accessed** – In file system, whole data to access for specific but in DBMS, only specified data will be accessed.
2. **Attributes** – Every data has data(meta data) which is not in DBMS.
3. **Concurrency** – DBMS is client server architecture which means clients can access data in the same time with some protocols.
4. **Security**–DBMS has role based access control which means clients with different roles can access different data.
5. **Redundancy** – DBMS doesn’t has redundancy, it has many contraints like primary, foreign keys.

***2-Tier Architecture***



Here, tier means layer.

1. It has 2 layers.
   1. **Client/User**
      1. It’s an iterface.
      2. **Uses API** to fatch data from server/database.
      3. Makes connection w/ server then sends query using DBMS for fatching data.
   2. **Server/Database**
      1. Processes the query then acts upon it and sends data.

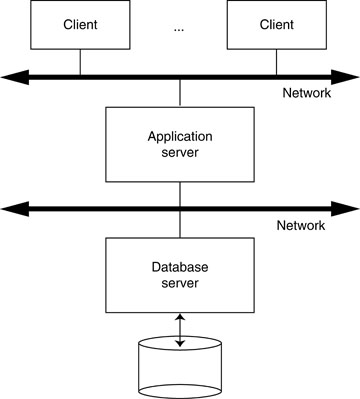
**Advantages:**

* **Maintainance** – Easy 2 maintain, as it has only two layer.

**Disadvantages:**

* **Scalability** – When client wants to connect it directly connects w/ database and database will have too much load. So, large no. of clients are difficult in 2-tier architechture.
* **Security** – Here, client directly connects with database without any mediator.

***3-Tier Architecture***

******

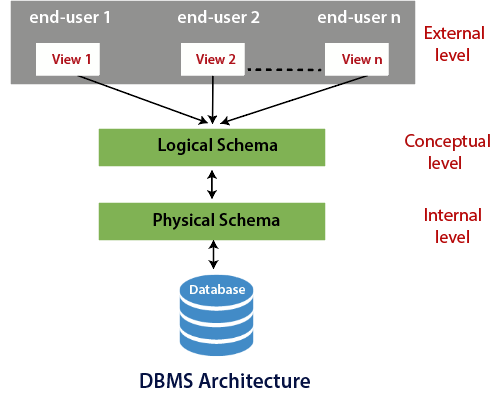
1. It has 3 layers.
   1. **Client/User**
      1. It makes connection with application layer.
      2. Sends query to application layer.
      3. Rest is same as in 2-tier architecture.
   2. **Application layer**
      1. It acts as mediator.
      2. Processes query and acts upon it.
      3. Fetches data from database.
2. **Server/Database**
3. Same as 2-tier database but sends data to application layer.

***Schema***

It’s a **logical representation of DBMS**.

e.g, Making tables to represent data like Student and its marks, roll\_no, etc.

***Three Schema Architecture***



By heading, it has 3 schemas,

It’s **theoraticel model**

It is based on the concept of **independence** of levels in databases

1. **External Schema / External Level**
   1. Every view has its own viewer like teacher can access different data while student can access different data or view.
   2. For users.
   3. Interface layer.
2. **Logical Schema / Conceptual Level**
   1. Blueprint of data.
   2. Connecting relational tables.
   3. Database desiging.
   4. No physical data into play.
3. **Physical Schema / Internal Level**
   1. Physical stuffs about data.
      1. Location
      2. Size
      3. File of data
      4. Which hard disk has data

***Data Independence***

Data change in one level doesn’t effect other levels. Take 3-schema architechture as example. There are 2 types of data independence:-

1. **Logical data Independence**
   1. Here, changes in logical/conceptual level doesn’t effect other levels.
2. **Physical data independence**
   1. Here, changes in physical level doesn’t effect other levels.

***Keys***

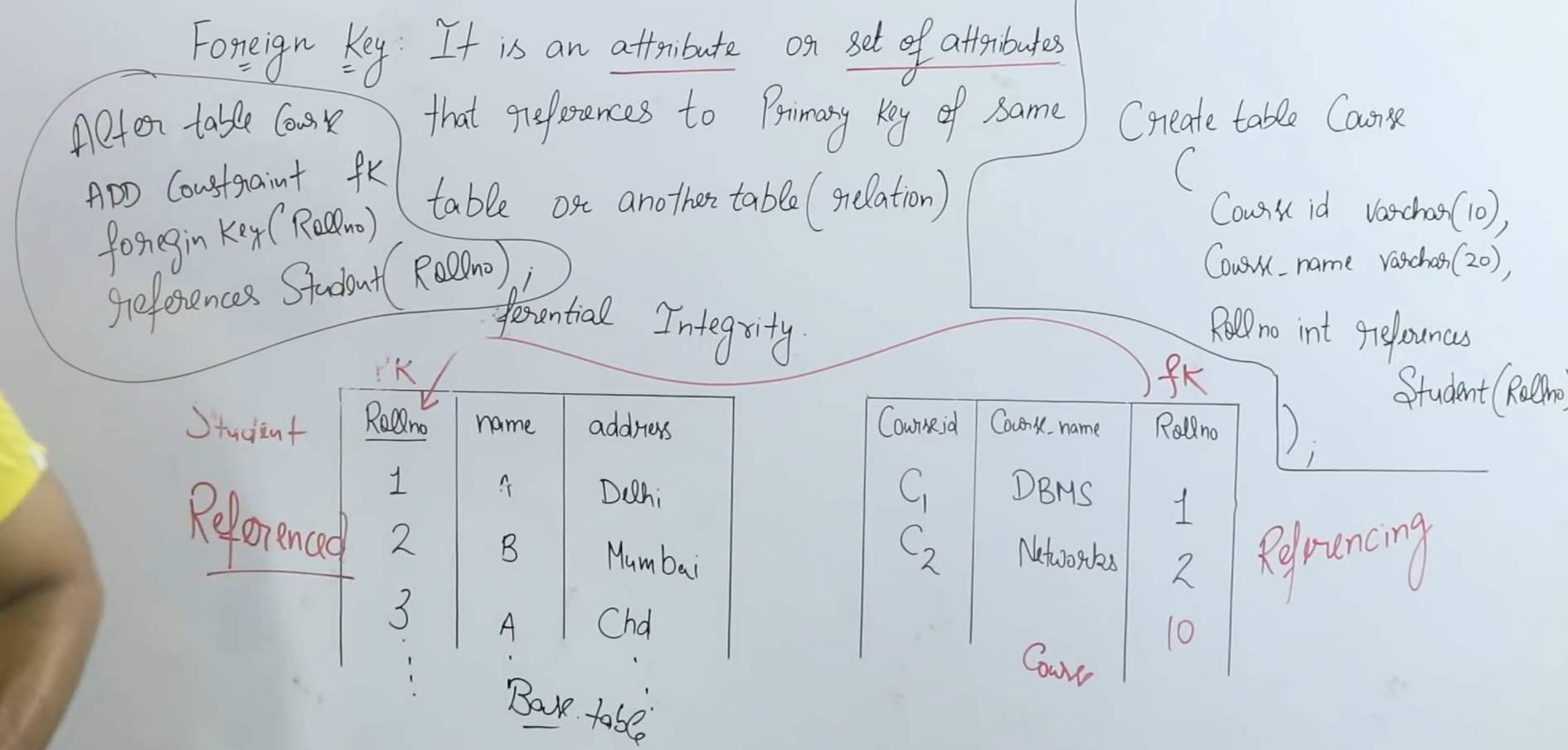
1. Keys are nothing but attribute or set of attributes used to uniquely identify tuples in the table.
2. Attributes used to uniquely identify tuple in table are candidate keys.
   1. Candidate keys are of two types:
      1. Primary key
      2. Alternate key

**Primary Key:**

* Must be unique attribute and not null.
* Only one primary key allowed.

**Foreign Key**

* Attribute or set of attribute.
* Reference to primary key of same or different table.
* Can be more than one foreign key.
* Maintains *referential integrity* (relation b\w table)
* The table which has primary key is referenced/base table.
* The table which has foreign key is referencing table.



1. INSERT, DELETE, UPDATE
   1. In referenced/base table
      1. **INSERT** – Won’t violate
      2. **DELETE** – May cause violation, as referencing table has its reference

Codes to correct it:

ON DELTETE CASCADE (will delete all referencing value on other table)

ON DELETE SET NULL (sets null in referencing values in other table)

ON DELETE NO ACTION (no action if reference is present)

* + 1. **UPDATE** – May cause violation, as values may present in other table.
  1. In referencing table
     1. **INSERT** – May cause violation, as inserting value which is different in base table.
     2. **DELETE** – Won’t violate.
     3. **UPDATE** – May cause violation, as updating value which is different or not present in base table.

**Super key**

1. It’s a combination of all the possible attributes which identifies two unique tuples in table.
2. Super set of any candidate keys is super key.
   1. For eg, *rollNo(CK) + name = super key*

***Entity-Relationship Model (ER Model)***

1. A representation of relationship between entities.
   1. For eg, *Student* ------study------- *Course*

Here, student and course are objects ie, entity.

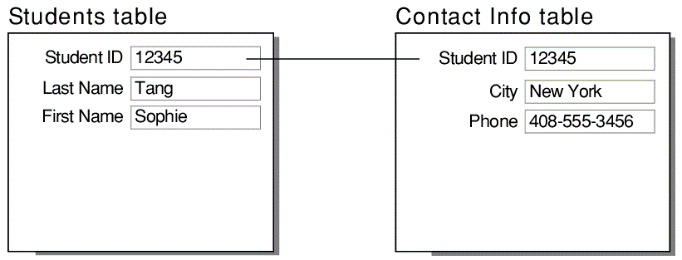
**Types of attributes in ER Model**

1. Single valued attribute
   1. Can’t have more than one value.
      1. For eg, Student->*rollNo*
2. Multi valued attribute
   1. Can have more than one value.
      1. For eg, Student->*gmail*
3. Simple attribute
   1. Can’t be further divided
      1. For eg, Student->*rollNo*
4. Composite attribute
   1. Can be further divided
      1. For eg, Student->*subjects->.maths/phy/chem*
5. Stored attribute
   1. Value can’t be derieved from it.
6. Derieved attribute
   1. Value can be derieved
      1. For eg, your age from from you DOB.
   2. **Will have dotted circle in ER Model**
7. Key attribute
   1. Uniquely identify any tuple.
   2. **Will have underline in ER Model**
8. Non-key attribute
   1. Being unique in unnecessary
9. Required attribute
   1. Must be filled
      1. For eg, Student->*name*
10. Optional attribute
    1. Can be skipped
11. Complex attribute
    1. (composite + multivalued) attributes

***Types/Degree of relationship(Cardinality)***

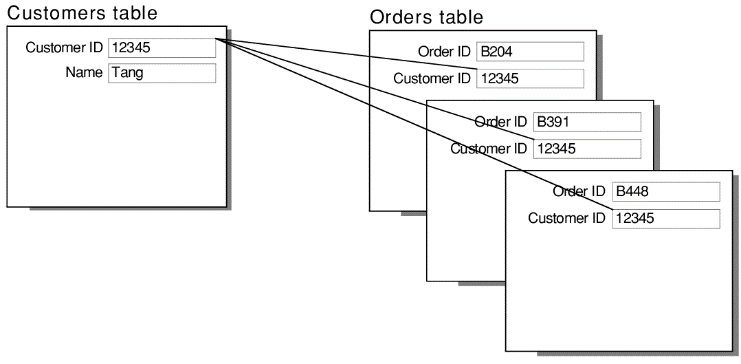
* It represents the number of entity types that associate in a relationship.
* For example, we have two entities, one is a student and the other is a bag and they are connected with the primary key and foreign key.

1. **One-to-one**



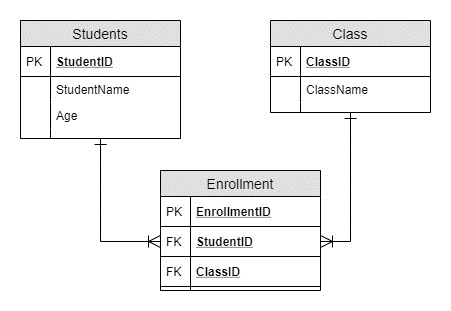
* 
  + A third table is formed by their relationship
  + Taking PK of both the tables and refrensing each other with foreign key
  + We can make either of them PK, in third table
  + **\*Third table called as referancing or relationship table\***
  + **Table is reduced towards many side**
    - **Reduced means merging here. ,ll=oll l**
  + Reduce:
    - We can reduce the tables by PK of third table
    - So, 1st table will have extra row which is DID

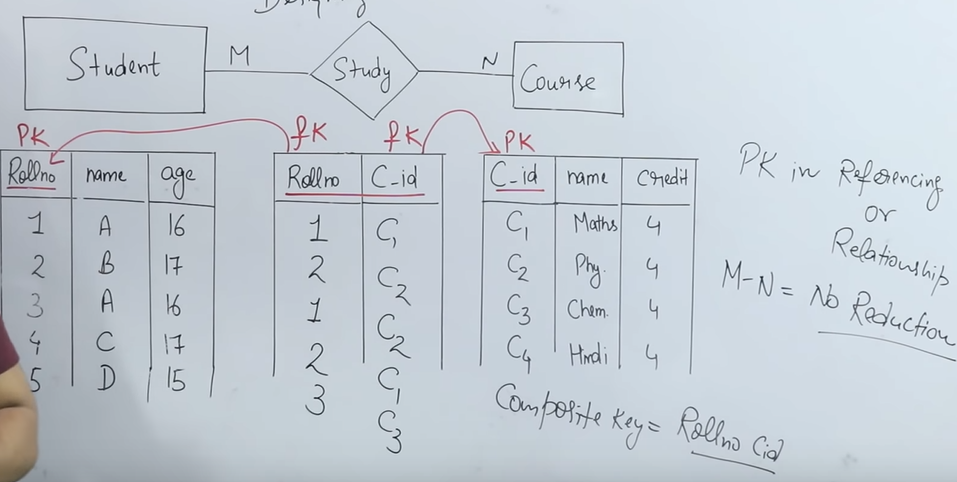
1. **One-to-many**



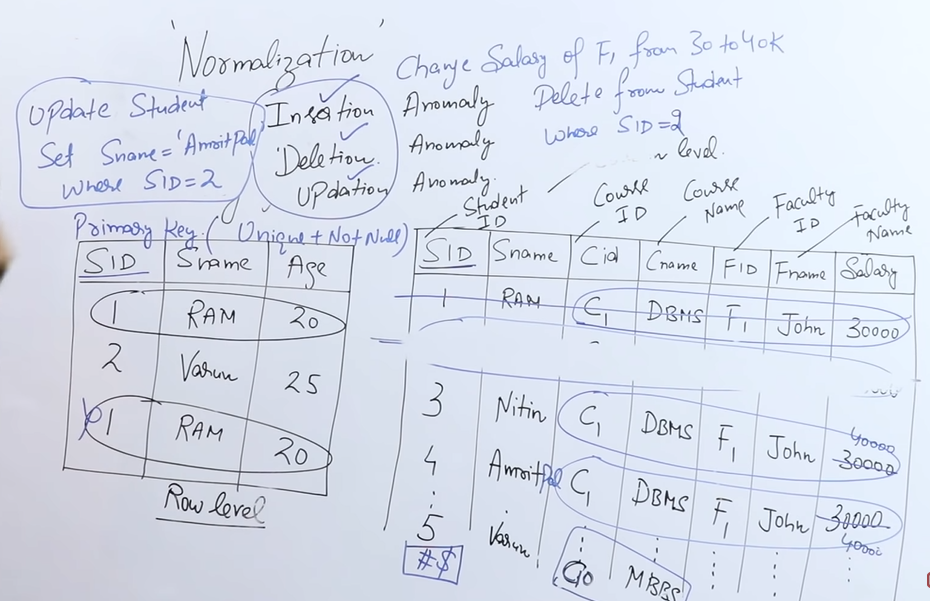
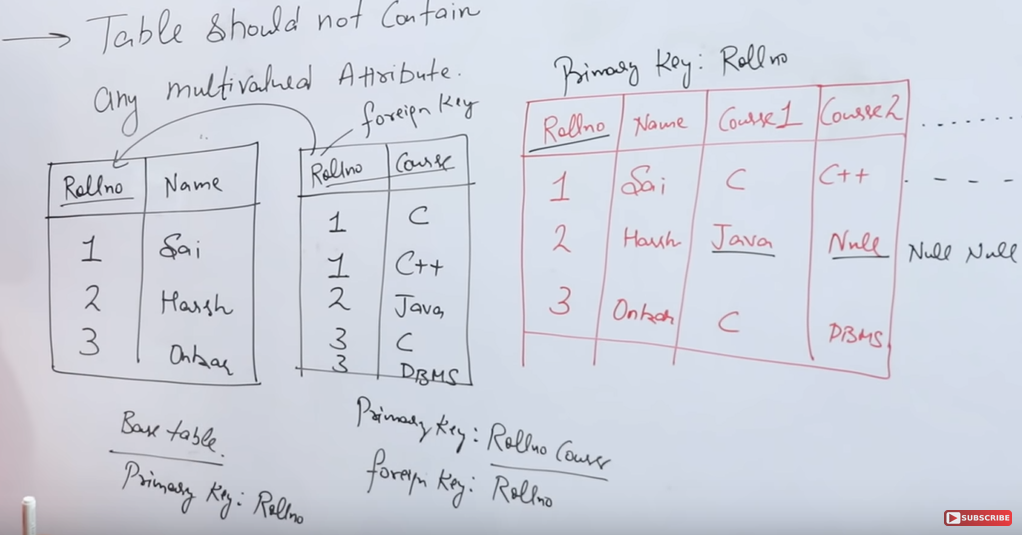
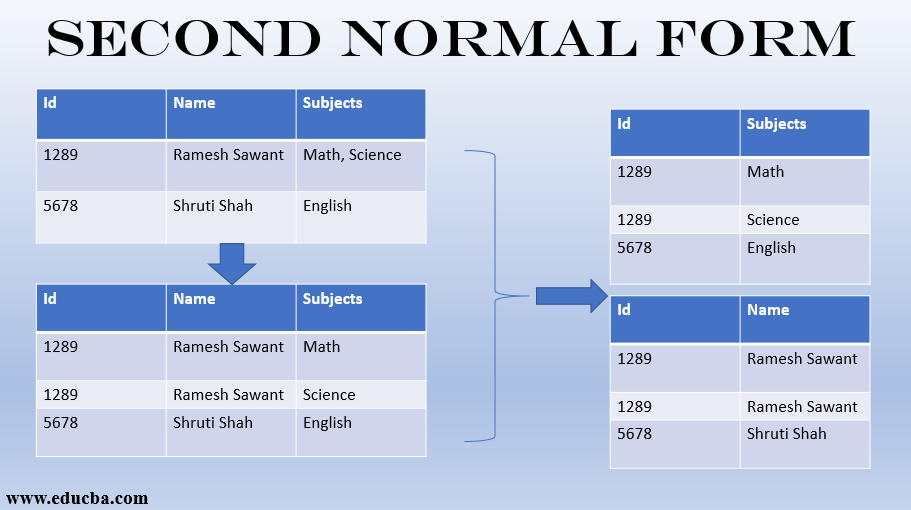
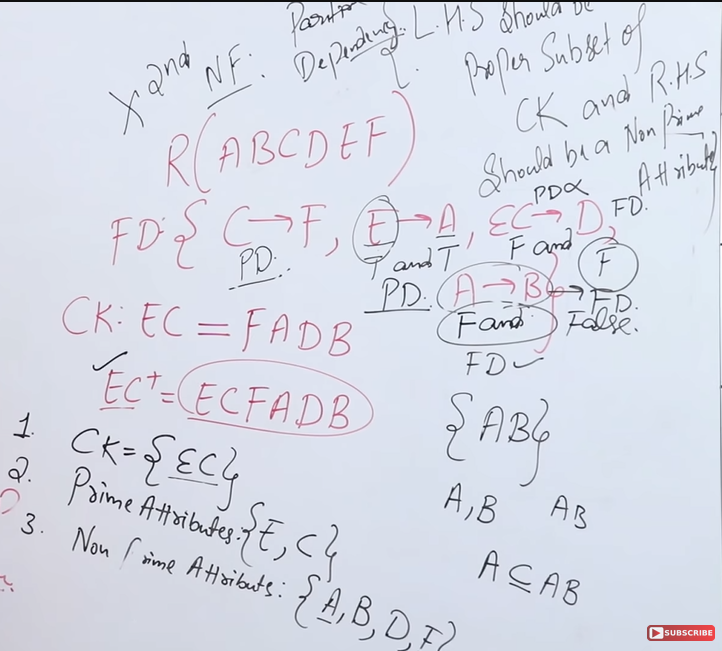
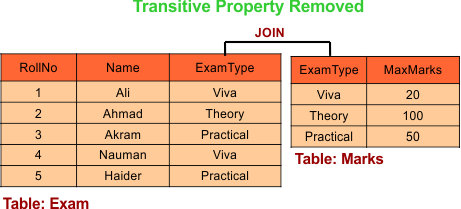
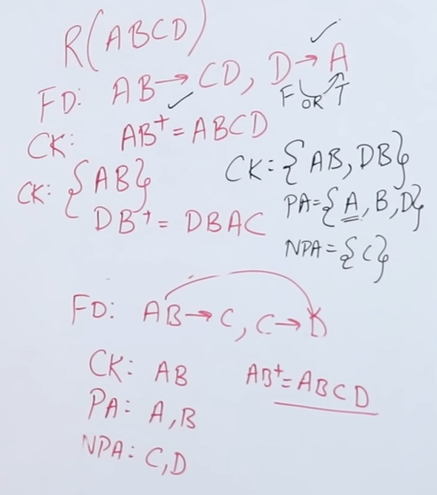
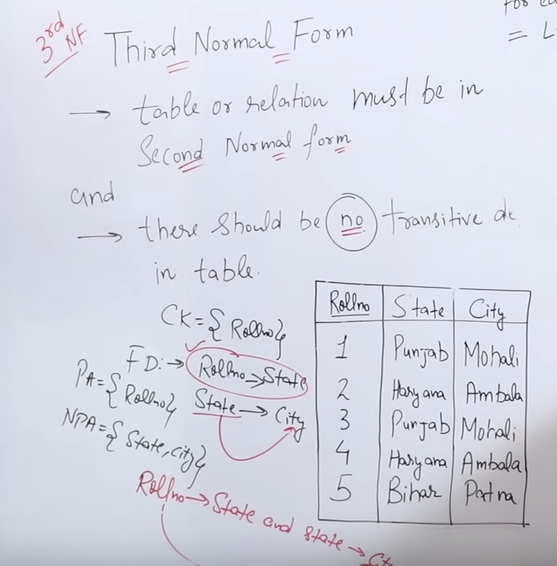
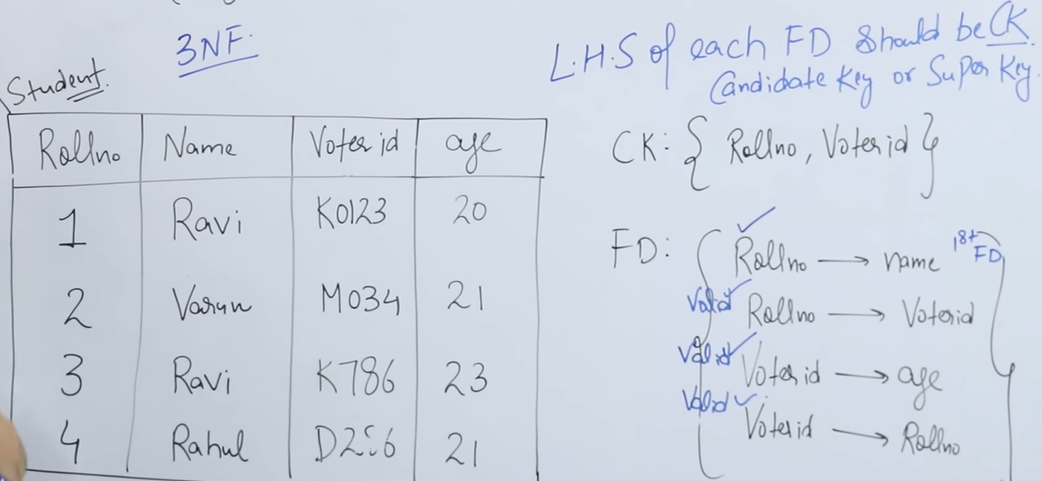
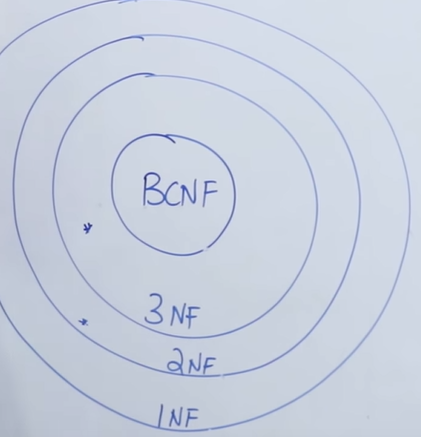
* No. of table can be reduced with relationship table.
  + But only reduced to the many side
* 

1. **Many-to-one**
   1. Same as one-to-many
2. **Many-to-many**

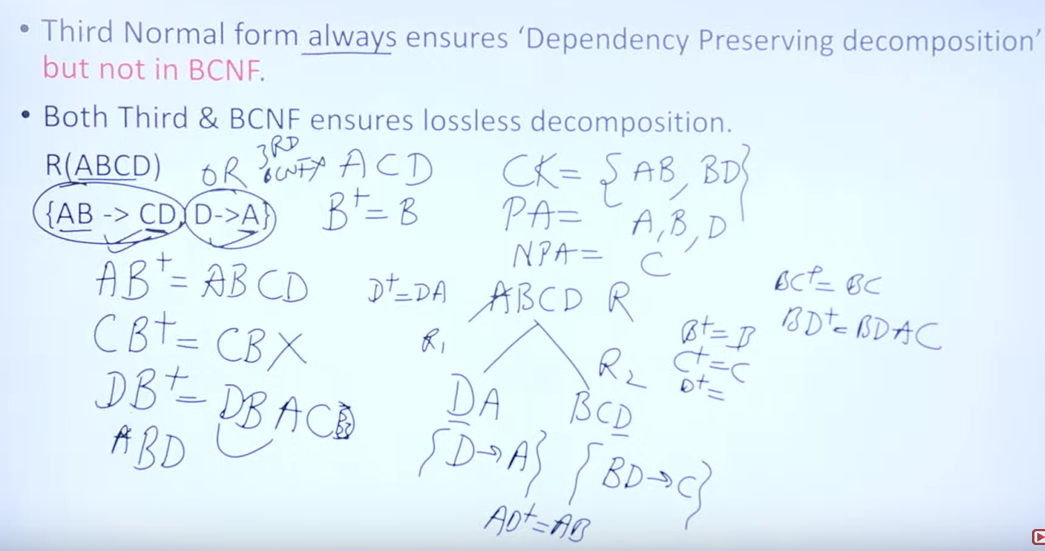


* No. of table can’t be reduced.
* 

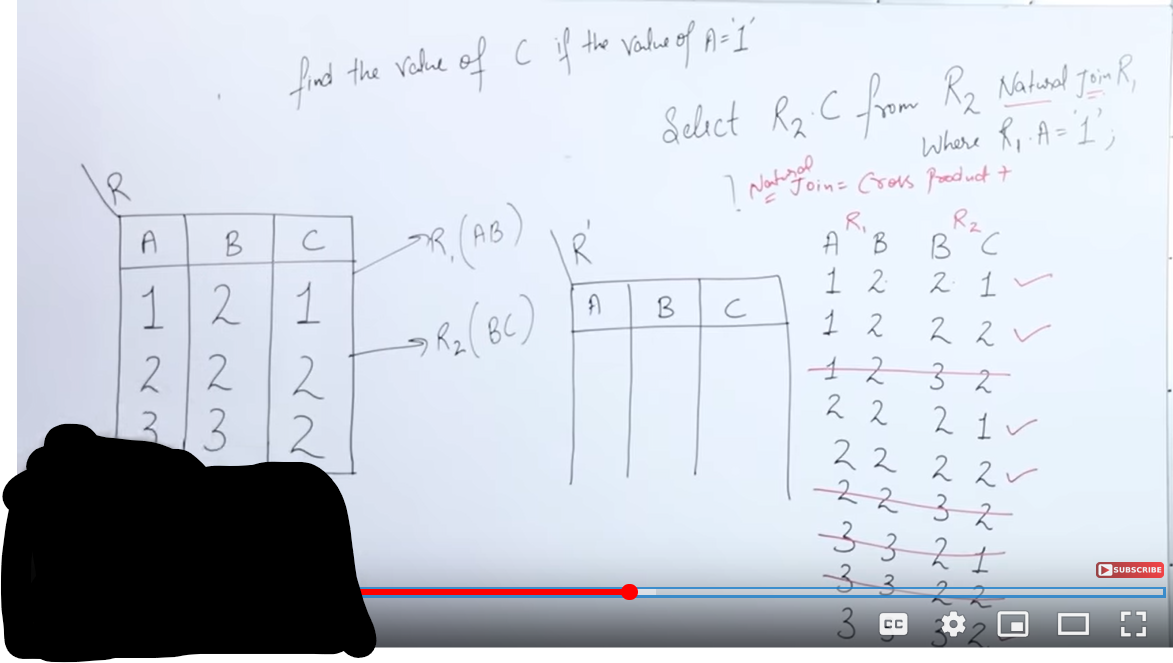
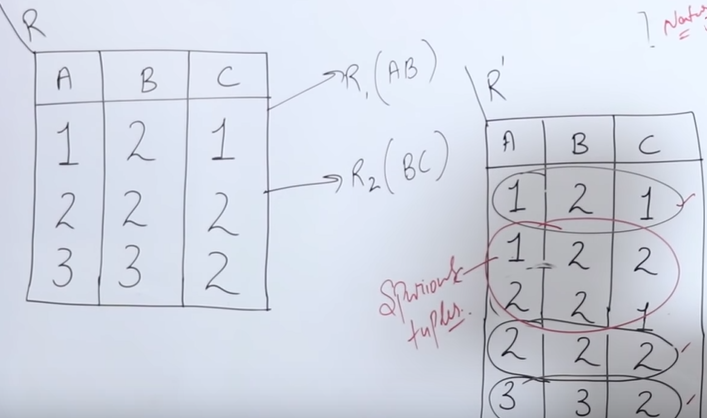
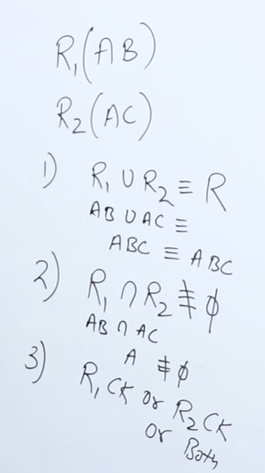
**Normalization**

* It’s **a technique to remove or reduce the duplicacy or redundancy from the table**.
* While execution, that normalization removes:
  + **Insertion anomaly**(abnormal, a problem)
    - Problem while inserting data
  + **Deletion anomaly**
    - Problem while deleting data
  + **Updation anomaly**
    - Problem while updating data
  + 
* It has two Types:
  + Row level
  + Column level
* **First Normal Form (1NF)**
  + No multivalue in an attribute
  + 
    - Separate table if there is multiple value
    - RollNo from T2 is referencing from RollNo in T1
    - And, it has different values in T3
* **Second Normal Form (2NF)**
  + All non-prime attributes must be full dependent on candidate key.
  + If a non-prime attribute is determined by the candidate part is partial dependency.
  + There should not be **partial dependency(PD)**.
    - Partial dependency means when a non-prime attribute is functionally dependent on part of CK
  + 
    - Here non-prime attribute ‘subjects’ depends on ‘id’
  + 
    - LHS should be proper subset of CK and RHS should be a non-prime attribute
* **Third Normal Form (3NF)**
  + Table or relation must be in 2NF.
  + There should not be partial dependency.
  + There should not be transitive dependency.
  + 
  + 
    - Non-prime attribute should not be determined by non-prime attribute
    - LHS should be CK/SK or RHS should be prime attribute
* Boyce Codd Normal Form (BCNF)
  + Some consider it as special case of 3NF
  + Conditions:
    - Should be a 3NF
    - LHS should be CK/SK
  + 
    - In above fig,
      * CK is rollno, voterid
      * In FD, LHS is CK, hence it’s BCNF
  + 
    - If a table in 2NF it doesn’t mean its 3NF but its possible in vice-versa
    - Above fig. suggests that,
      * BCNF consist of 1,2,3 NF
      * 3NF consist of 1,2 NF
      * 2NF consist of 1NF

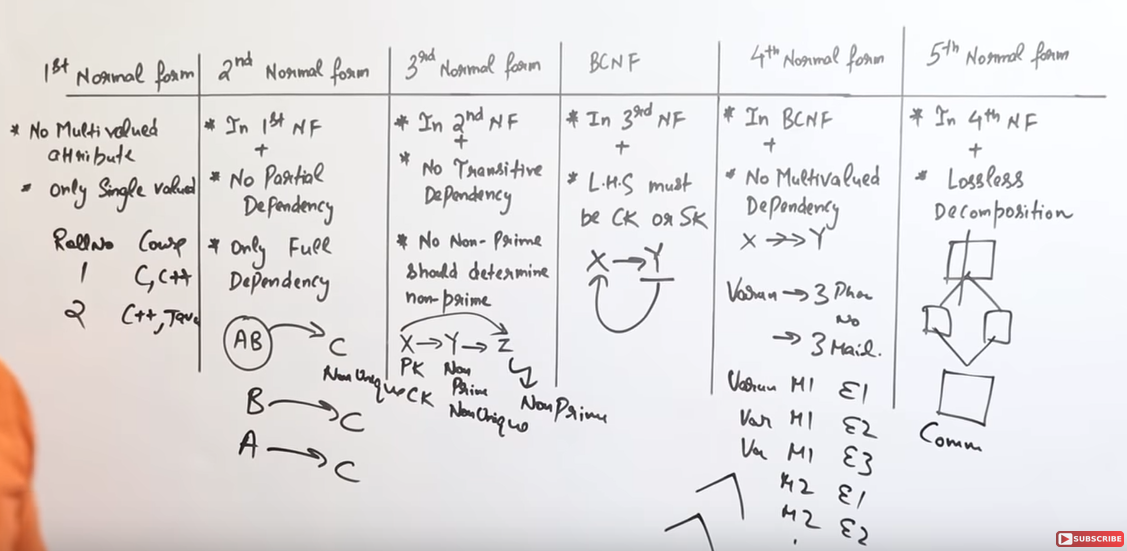
**Dependency Preserving Decomposition**

* While decomposition or dividing a table
  + A table must be **lossless**
  + **Dependency** must be **preserved**
  + **All NF ensures losslessness**
  + Dependency is only preserved completely by 3NF
  + BCNF also preserves dependency but its not always the case
  + ****
    - In the above example,
      * AB determines CD and D determines A
      * AB is CK but D is not (acc. to BCNF)
        + So we’ve to divide the table
        + From above image, we get AB from AB not ABCD
        + Hence, dependency is not preserved

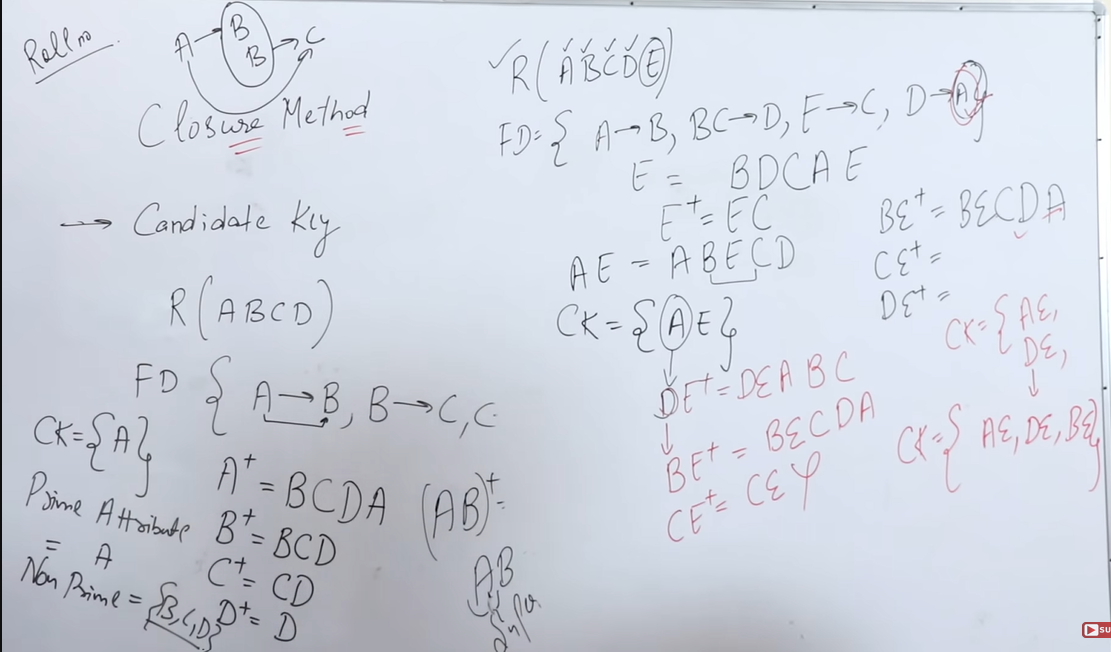
**Lossless and Lossy decomposition**

* ****
  + According to question:
    - We need to find value of C if value of A=1
    - So we’ll divide table in 2
      * i.e, R1(AB) and R2(BC)
    - Query:
      * Select R2.C from R2 Natural Join R1 Where R1.A=’1’
    - We have to do Cross product of both the table by taking value of R1 with all values of R2
    - From above image,
      * Remove rows having different B values in R1 and R2
      * And, join the table with same B into B
    - Joined table will look like this, (in below image)
* ****
  + Joined table has 2 more tuples than original table
    - This extra tuples are called, **Spurious tuples**
  + This extra tuples are lossy decomposition
    - **Lossy in terms of consistancy**
* In order, to remove losslessness we’ve to divide table based on:
  + **We’ve to table common a CK or SK from first table or second table or both**
  + Which is R1(AB) R2(AC) where common is A
* ****
  + In union, we should get all attributes
  + In intersection, it shouldn’t be null
  + CK can be from R1, R2 or both.

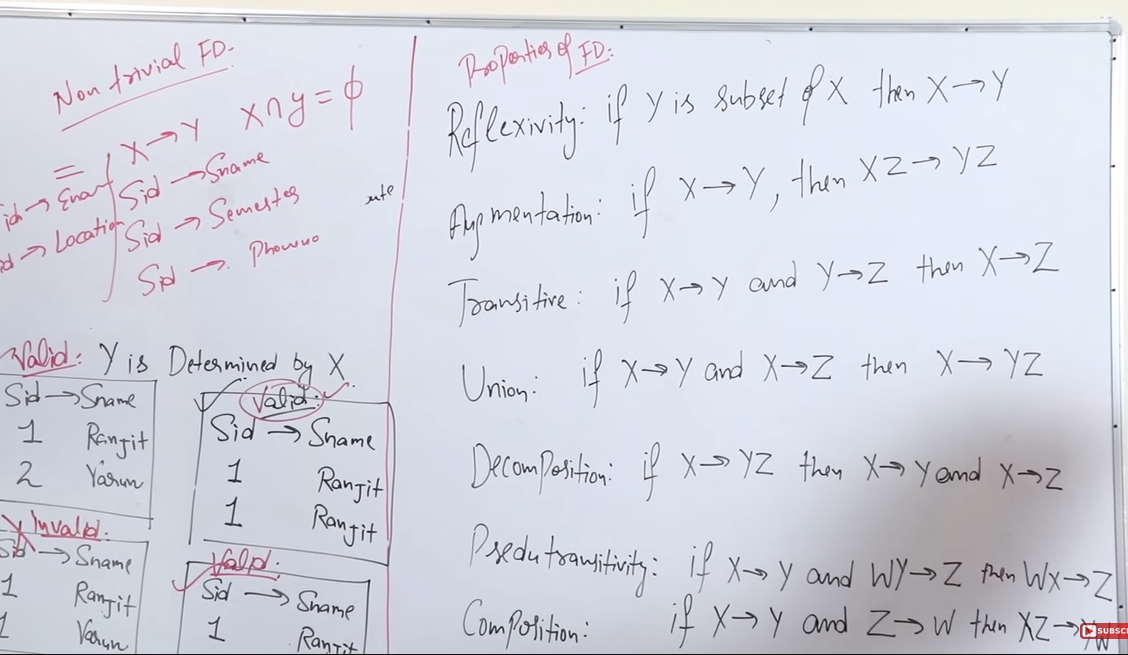
**Summary of NFs**

* ****
  + **1NF**
    - No multi-valued attribute
    - Only single value
  + **2NF**
    - No partial dependency
  + **3NF**
    - No transitive dependency
    - No non-prime attribute should determine non-prime attribute
  + **BCNF**
    - LHS must be CK or SK
  + **4NF**
    - No multivalued dependency
  + **5NF**
    - Lossless decomposition

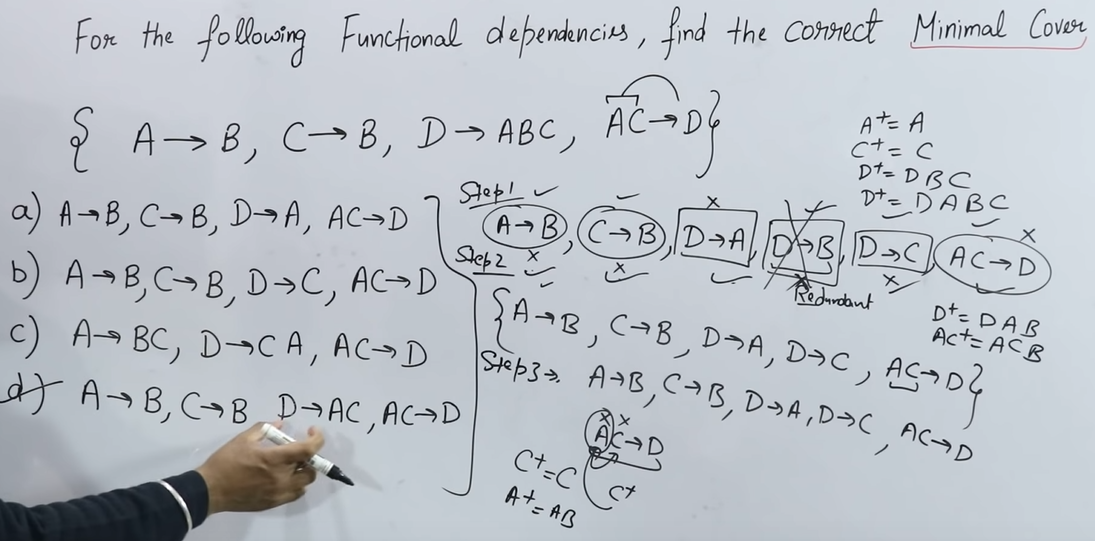
**Closure method**

* It’s a method to find all the candidate keys in a table.
  + Candidate key must be a minimal value like A, B
    - If not, it’ll be a super key like AB, BC
* 
  + A->B, means A can determine B
  + A+ = BCDA, means A can determine BCDA
  + **Candidate key has all the attributes**
    - All attributes in CK are **prime attributes**
  + In second example,
    - R(ABCDE), there is no E in right side so we’ve to add E in left side also
    - So CK is BE+
      * Having prime attribute, B,E,C,D,A

**Functional Depandancy**

* It’s a relationship between attributes of a table dependent on each other.
* 

**Minimal Cover**

* 
  + We’ll be given a FD
  + We just have to find minimal cover(irreducible set of attributes)
  + For example(from above image),
    - In step 1, we wrote all the attributes
    - In step 2, we wrote all the attributes which is not a CK(closure of any attribute that finds all the attributes)
    - In step 3, again we further removed all CK
    - Remaining is the minimal cover