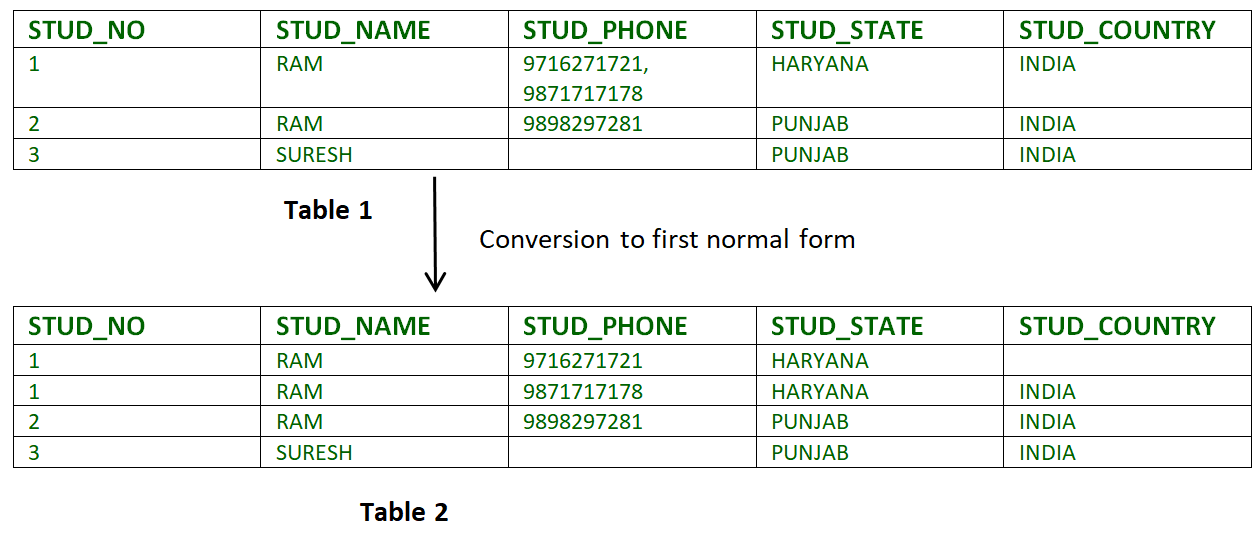
Database Normalization | Normal Forms

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

**1. First Normal Form –**

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

* **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been shown in table 2.  
  [](https://www.geeksforgeeks.org/wp-content/uploads/Normalisation_normalforms_1.png)
* **Example 2 –**
* ID Name Courses
* ------------------
* 1 A c1, c2
* 2 E c3
* 3 M C2, c3

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

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

ID Name Course

------------------

1 A c1

1 A c2

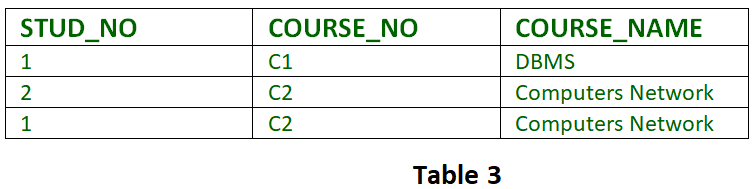
2 E c3

3 M c1

3 M c2

**2. Second Normal Form –**

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

[](https://www.geeksforgeeks.org/wp-content/uploads/Normalisation_normalforms_2.png)

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

* **Example 1 –** In relation STUDENT\_COURSE given in Table 3,
* FD set: {COURSE\_NO->COURSE\_NAME}
* Candidate Key: {STUD\_NO, COURSE\_NO}

In FD COURSE\_NO->COURSE\_NAME, COURSE\_NO (proper subset of candidate key) is determining COURSE\_NAME (non-prime attribute). Hence, it is partial dependency and relation is not in second normal form.  
To convert it to second normal form, we will decompose the relation STUDENT\_COURSE (STUD\_NO, COURSE\_NO, COURSE\_NAME) as :

STUDENT\_COURSE (STUD\_NO, COURSE\_NO)

COURSE (COURSE\_NO, COURSE\_NAME)

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

* **Example 2 –** Consider following functional dependencies in relation  R (A,  B , C,  D )
* AB -> C [A and B together determine C]

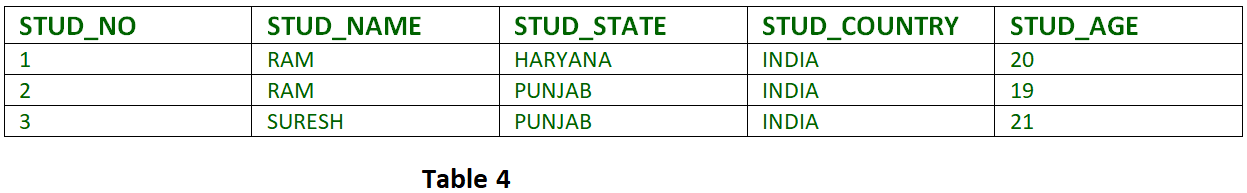
BC -> D [B and C together determine D]

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

**3. Third Normal Form –**

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes is it is in second normal form.  
A relation is in 3NF iff **at least one of the following condition holds** in every non-trivial function dependency X –> Y

* 1. X is a super key.
  2. Y is a prime attribute (each element of Y is part of some candidate key).

[](https://www.geeksforgeeks.org/wp-content/uploads/Normalisation_normalforms_3.png)  
**Transitive dependency –**If A->B and B->C are two FDs then A->C is called transitive dependency.

* 1. **Example 1 –** In relation STUDENT given in Table 4,

FD set: {STUD\_NO -> STUD\_NAME, STUD\_NO -> STUD\_STATE, STUD\_NO -> STUD\_COUNTRY, STUD\_NO -> STUD\_AGE, STUD\_STATE -> STUD\_COUNTRY}  
Candidate Key: {STUD\_NO}

For this relation in table 4, STUD\_NO -> STUD\_STATE and STUD\_STATE -> STUD\_COUNTRY are true. So STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY\_STUD\_AGE) as:  
STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)  
STATE\_COUNTRY (STATE, COUNTRY)

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

**4. Boyce-Codd Normal Form (BCNF) –**

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

* + - **Example 1 –** Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}  
      Step 1. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can’t be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.  
      Step 2. Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.  
      Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.  
      The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).  
      The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.  
      So the highest normal form of relation will be 2nd Normal form.
    - **Example 2 –**For example consider relation R(A, B, C)  
      A -> BC,  
      B ->  
      A and B both are super keys so above relation is in BCNF.

**Key Points –**

* + - BCNF is free from redundancy.
    - If a relation is in BCNF, then 3NF is also also satisfied.
    - If all attributes of relation are prime attribute, then the relation is always in 3NF.
    - A relation in a Relational Database is always and at least in 1NF form.
    - Every Binary Relation ( a Relation with only 2 attributes ) is always in BCNF.
    - If a Relation has only singleton candidate keys( i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF( because no Partial functional dependency possible).
    - Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
    - There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it’s generally not required to go beyond BCNF.

**Functional Dependency**  
Functional Dependency is a constraint between two sets of attributes in a relation from a database. Functional dependency is denoted by arrow (→). If an attributed A functionally determines B, then it is written as A → B.  
For example employee\_id → name means employee\_id functionally determines name of employee. As another example in a time table database, {student\_id, time} → {lecture\_room}, student ID and time determine the lecture room where student should be.

**What does functionally dependent mean?**  
A function dependency A → B mean for all instances of a particular value of A, there is same value of B.

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

A B

------

1 3

2 3

4 0

1 3

4 0

**Trivial Functional Dependency**  
X –> Y is trivial only when Y is subset of X.  
Examples

ABC --> AB

ABC --> A

ABC --> ABC

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

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

Id --> Name,

Name --> DOB

## **Armstrong's Axioms**

If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

* **Reflexive rule** − If alpha is a set of attributes and beta is\_subset\_of alpha, then alpha holds beta.
* **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
* **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

## **Boyce Codd normal form (BCNF)**

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every [functional dependency](https://beginnersbook.com/2015/04/functional-dependency-in-dbms/) X->Y, X should be the super key of the table.

**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 |

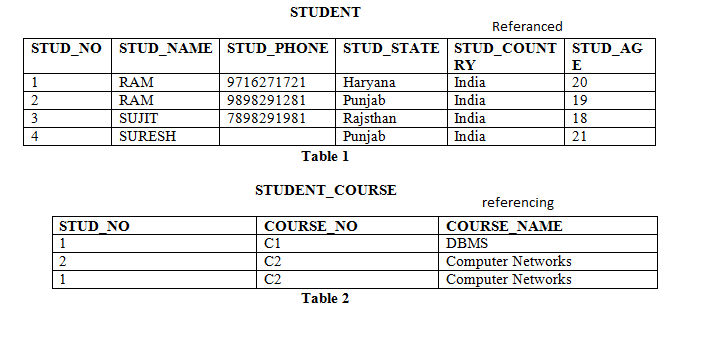
**Functional dependencies**:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

**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

DBMS | Anomalies in Relational Model

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



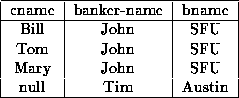
**Insertion anomaly:** If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation. For Example, If we try to insert a record in STUDENT\_COURSE with STUD\_NO =7, it will not allow.

**Deletion and Updation anomaly:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will not allow deleting the tuple from referenced relation. For Example, If we try to delete a record from STUDENT with STUD\_NO =1, it will not allow. To avoid this, following can be used in query:

* **ON DELETE/UPDATE SET NULL:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and set the value of referenced attribute to NULL.
* **ON DELETE/UPDATE CASCADE:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and referencing relation as well

## **Comparison of BCNF and 3NF**

1. We have seen BCNF and 3NF.
   * It is always possible to obtain a 3NF design without sacrificing lossless-join or dependency-preservation.
   * If we do not eliminate all transitive dependencies, we may need to use null values to represent some of the meaningful relationships.
   * Repetition of information occurs.
2. These problems can be illustrated with Banker-schema.
   * As banker-name *tex2html_wrap_inline1526* bname , we may want to express relationships between a banker and his or her branch.

     
**Figure 7.4:**   An instance of Banker-schema.

* + Figure [7.4](https://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter7/node12.html#fig68banker) shows how we must either have a corresponding value for customer name, or include a null.
  + Repetition of information also occurs.
  + Every occurrence of the banker's name must be accompanied by the branch name.

1. If we must choose between BCNF and dependency preservation, it is generally better to opt for 3NF.
   * If we cannot check for dependency preservation efficiently, we either pay a high price in system performance or risk the integrity of the data.
   * The limited amount of redundancy in 3NF is then a lesser evil.
2. To summarize, our goal for a relational database design is
   * BCNF.
   * Lossless-join.
   * Dependency-preservation.
3. If we cannot achieve this, we accept
   * 3NF
   * Lossless-join.
   * Dependency-preservation.
4. **A final point:** there is a price to pay for decomposition. When we decompose a relation, we have to use natural joins or Cartesian products to put the pieces back together. This takes computational time

## **ACID Properties**

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
* **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
* **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
* **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

**SQL**

**Tables** − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

**Tuple** − A single row of a table, which contains a single record for that relation is called a tuple.

**Relation instance** − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

**Relation schema** − A relation schema describes the relation name (table name), attributes, and their names.

**Relation key** − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

**Attribute domain** − Every attribute has some pre-defined value scope, known as attribute domain.

## **Constraints**

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

* Key constraints
* Domain constraints
* Referential integrity constraints

### Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key**for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that −

* in a relation with a key attribute, no two tuples can have identical values for key attributes.
* a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

### Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

### Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

* The principal end of the constraint. (An entity type whose entity key is referenced by the dependent end.)
* The entity key of the principal end.
* The dependent end of the constraint. (An entity type that has a property or properties that reference the entity key of the principal end.)
* The referencing property or properties of the dependent end.