Problems without Normalization

- If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss.
- Insertion, Updation and Deletion Anamolies are very frequent if database is not normalized.
- To understand these anomalies let us take an example of a **Student** table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

- In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

1. Insertion Anomaly

- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.
- Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but **Insertion anomalies**.

2. Updation Anomaly

- What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.
- This is **Updation anomaly**.

3. Deletion Anomaly

- In our Student table, two different information are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information.
- This is **Deletion anomaly**.

Basics of Normalization

- Database normalization is the process of removing redundant data from your tables to improve **storage efficiency, data integrity, and scalability**.
- In the relational model, methods exist for quantifying how efficient a database is. These classifications are called normal forms (or NF), and there are algorithms for converting a given database in normal form.
- **Definition:** In relational database design, the process of organizing data to minimize redundancy. Normalization usually involves dividing a database into two or more tables without losing information and defining relationships between the tables.
- The goal of normalization process are:
    1) To minimize data redundancy.
    2) To minimize update, deletion and insertion anomalies.
    3) Improve data integrity, scalability and data consistency.
    4) Reduces disk space.
- The **Normal Form** is a state of a relation that results from applying some criteria on that relation.
- Various normal forms are given below:
    1) **First Normal Form (1NF)**
    2) **Second Normal Form (2NF)**
    3) **Third Normal Form (3NF)**
    4) Boyce-Codd Normal Form (1NF)
    5) Forth Normal Form (4NF)
    6) Fifth Normal Form (5NF)

Normal Forms

First Normal Form (1NF)

- A relation **R** is in **first normal form** (1NF) if and only if all **domains** contain **atomic** values only.

OR

- A relation **R** is in **first normal form** (1NF) if and only if it does not contain any **composite** or **multi valued attributes** or their combinations.

- Every tuple (row) in the relation schema contains only one value of each attribute and no repeating groups.
- 1 NF disallows multi-valued attributes that are themselves composites.
- 1NF disallows having a set of values.

Example:

Customer:

<u>Cid</u>	Name	Address		Contact_no
		Society	City	
C01	Emily	Nana Bazar, Anand		9879898798,7877855416
C02	Jeniffer	C.G.Road, Ahmedabad		9825098254
C03	Peter	M.G.Road, Rajkot		9898787898

- Above relation has four attributes **Cid, Name, Address, Contact_no.** Here **Address** is **composite** attribute which is further divided in to sub attributes as **Society** and **City**. Another attribute **Contact_no** is **multi valued attribute** which can store more than one values. So above **relation is not in 1NF**.

Problem:

- Suppose we want to find all customers for some particular city then it is difficult to retrieve.
- Reason is city name is combined with society name and stored whole as address.

Solution:

- Insert separate attribute for each sub attribute of **composite** attribute.

❖ **First Approach:**

- Determine maximum allowable values for **multi-valued** attribute.
- Insert separate attribute for **multi valued** attribute and insert only one value on one attribute and other in other attribute.
- So, above table can be created as follows:

Customer:

<u>Cid</u>	Name	Society	City	Contact_no1	Contact_no2
C01	Emily	Nana Bazar	Anand	9879898798	7877855416
C02	Jeniffer	C.G.Road	Ahmedabad	9825098254	
C03	Peter	M.G.Road	Rajkot	9898787898	

❖ **Second Approach:**

- Remove multi valued attribute and place it in a separate relation along with the primary key of a given original relation.
- So, above table can be created as follows:

Customer:

<u>Cid</u>	Name	Society	City
C01	Emily	Nana Bazar	Anand
C02	Jeniffer	C.G.Road	Ahmedabad
C03	Peter	M.G.Road	Rajkot

Customer_Contact:

Cid	<u>Contact_no</u>
C01	9879898798
C01	7877855416
C02	9825098254
C03	9898787898

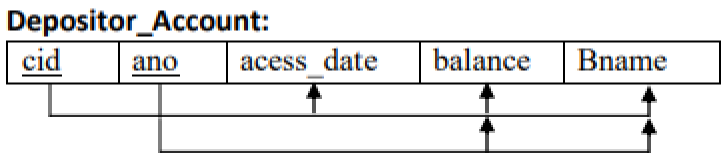
Second Normal Form (2NF)

- A relation **R** is in second normal form (2NF) if and only if **it is in 1NF** and every **non-prime** attribute of relation is fully dependent on the **primary key**.

OR

- A relation **R** is in second normal form (2NF) if and only if **it is in 1NF** and no any **non-prime** attribute is **partially** dependent on the **primary key**.

Example:



- Above relation has five attributes **cid, ano, acess_date, balance, bname** and two FDS

  FD1: {cid,ano}→ {acess_date, balance, bname} and

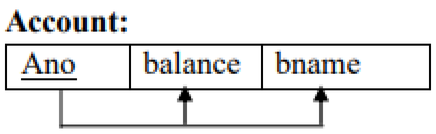
  FD2: ano→ {balance, bname}
- We have **cid and ano as primary key.** As per FD2 **balace** and **bname** are only depend on **ano** not **cid**.
- In above table **balance** and **bname** are not fully dependent on primary key but these attributes are partial dependent on primary key. So above relation is not in 2NF.

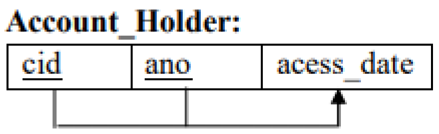
Problem:

- For example in case of joint account **multiple customers** have common **account**. If some account says '**A02**' is jointly by two customers says '**C02**' and '**C04**' then data values for attributes **balance** and **bname** will be duplicated in two different tuples of customers '**C02**' and '**C04**'.

Solution:

- Decompose relation in such a way that resultant relation does not have any **partial FD**.
- For this purpose remove partial dependent attributes that violets 2NF from relation. Place them in **separate new relation** along with the **prime** attribute on which they are **full dependent**.
- The **primary key** of new relation will be the attribute on which they are fully dependent.
- Keep other attributes same as in that table with same primary key.
- So, above table **Depositor_Account** can be decomposed into **Account** and **Account_Holder** table as per following.





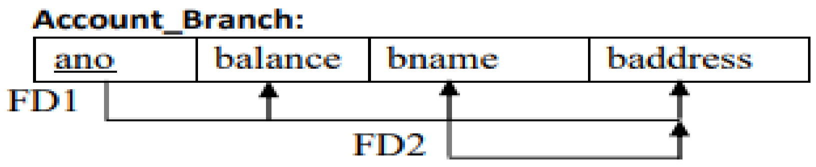
- So, Both **Account** and **Account_Holder** relations are in **second normal form**.

Third Normal Form (3NF)

- A relation **R** is in third normal form (3NF) if and only if **it is in 2NF** and **no any non-prime** attribute of a relation is transitively dependent on the primary key.
- An attribute **C** is transitively dependent on attribute **A** if there exist an attribute **B** such that:

  A → B and **B → C**.

Example:



- Above relation has four attributes **ano, balance, bname, baddress** and two FDS

  FD1 : ano → {balance, bname, baddress} and

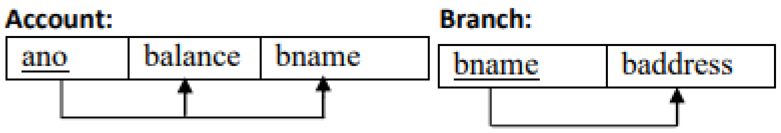
  FD2 : bname → baddress
- So, from **FD1** and **FD2** and using transitivity rule we get **ano→ baddress**.
- So, there is transitively dependency from **ano** to **baddress** using **bname** in which **baddress** is **non-prime** attribute.
- So, there is a non-prime attribute **baddress** which is transitively dependent on primary key **ano**.
- So, above relation is not in 3NF.

Problem:

- Transitively dependency results in **data redundancy**.
- In this relation **branch address** will be stored repeatedly for each account of same branch which occupy more space.

Solution:

- Decompose relation in such a way that resultant relation does not have any **non-prime** attribute that are **transitively dependent on primary key**.
- For this purpose remove **transitively dependent attribute** that violets 3NF from relation. Place them in separate **new relation** along with the non-prime attribute due to which transitive dependency occurred. The **primary key** of new relation will be this non-prime attribute.
- Keep other attribute same as in that table with same primary key.
- So, above table **Account_Branch** can be decomposed into **Account** and **Branch** table as per following:



- Now, there is no any transitive dependency in **account** and **branch** relations.
- So, these both relations are in **third normal form**.