* **Problem Without Normalization:**

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not normalized.

* **Anomaly:** A problems that can occur in poorly planned un-normalized database where all data is stored in one tables.

|  |  |  |  |
| --- | --- | --- | --- |
| S\_id | S\_Name | S\_Address | Subject\_opted |
| 401 | Adam | Noida | Bio |
| 402 | Alex | Panipat | Maths |
| 403 | Stuart | Jammu | Maths |
| 404 | Adam | Noida | Physics |

1. **Updation Anomaly:** exists when one or more instances of duplicated data is updated but not all.

**Ex:**To update address of a student who occurs twice or more than twice in a table, we will have to update **S\_Address** column in all the rows, else data will become inconsistent.

1. **Insertion Anomaly: Certain attributes cannot be inserted in to database without presence of another attribute.**

**Ex:**Suppose for a new admission, we have a Student id(S\_id), name and address of a student but if student has not opted for any subjects yet then we have to insert NULL there, leading to Insertion Anomaly.

1. **Deletion Anomaly: Certain attributes are lost because of the deletion of other attributes.**

**Ex:**If (S\_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

* **Normalization**: is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data and ensuring data dependencies make sense.

There are several benefits for using Normalization in Database.

**Benefits:**

* Eliminate data redundancy
* Improve performance
* Query optimization
* Faster update due to less number of columns in one table
* Index improvement

There are different types of Normalizations form available in the Database.

* **First Normal Form (1NF):** As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Harshal | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Manish | Bangalore | 9990000123 8123450987 |

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Harshal | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Manish | Bangalore | 9990000123 |
| 104 | Manish | Bangalore | 8123450987 |

* **Second normal form (2NF):** A table is said to be in 2NF if both the following conditions hold:

1. Table is in 1NF (First normal form).
2. No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

|  |  |  |
| --- | --- | --- |
| teacher\_id | subject | teacher\_age |
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys**: {teacher\_id, subject} **Non prime attribute**: teacher\_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher\_age is dependent on teacher\_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

**1. Teacher\_details table:**

|  |  |
| --- | --- |
| teacher\_id | teacher\_age |
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

**2. Teacher\_subject table:**

|  |  |
| --- | --- |
| teacher\_id | subject |
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

Now the tables comply with Second normal form (2NF).

* **Third Normal form (3NF):** A table design is said to be in 3NF if both the following conditions hold:

1. Table must be in 2NF
2. [Transitive functional dependency](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

**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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_zip | emp\_state | emp\_city | emp\_district |
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

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 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

* **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:

**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\_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 |

**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.

