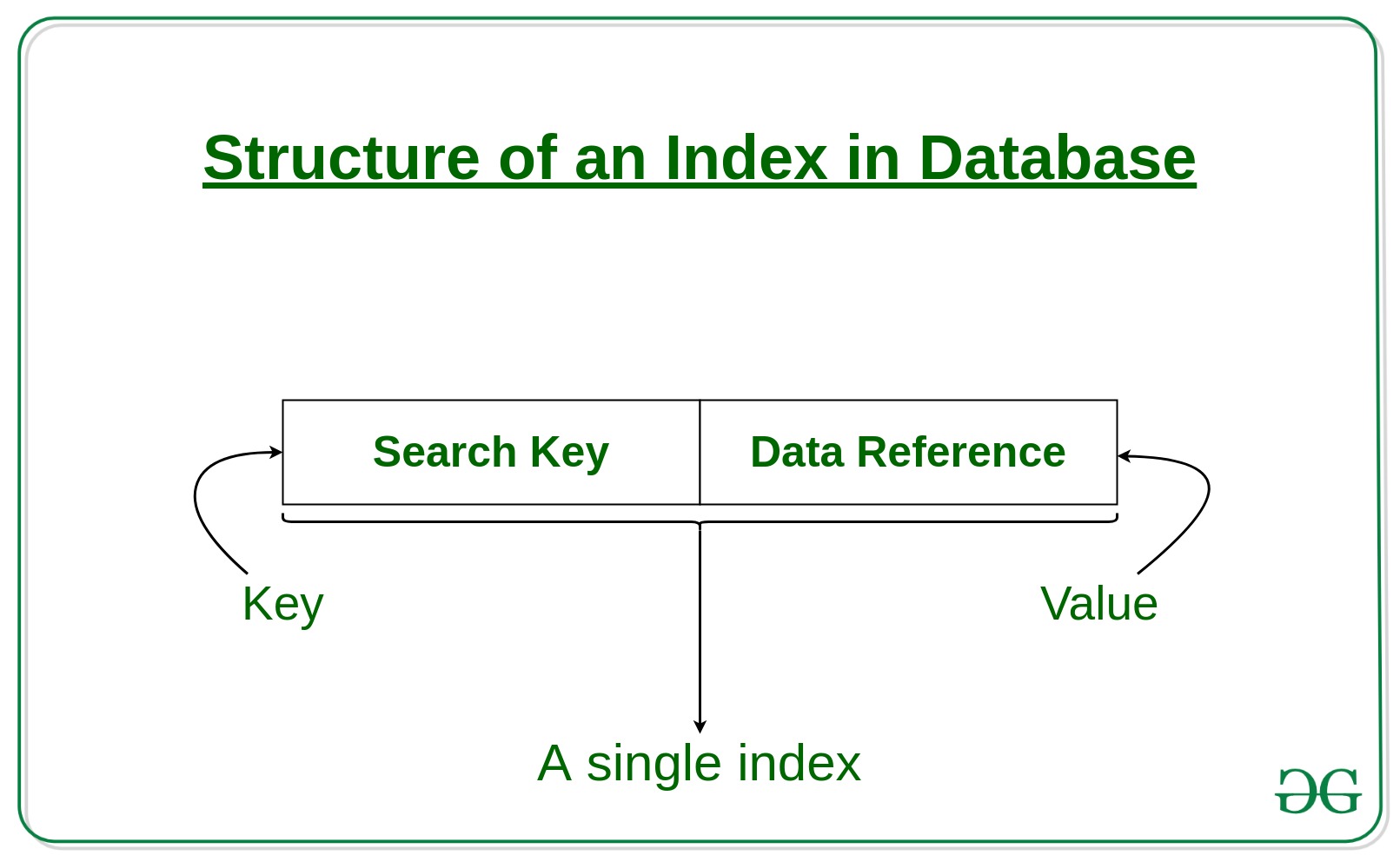
**Indexes**

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database

Indexes are created using a few database columns.

* The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.  
  *Note: The data may or may not be stored in sorted order.*
* The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183525/Structure-of-an-Index-in-Database.jpg)

The indexing has various attributes:

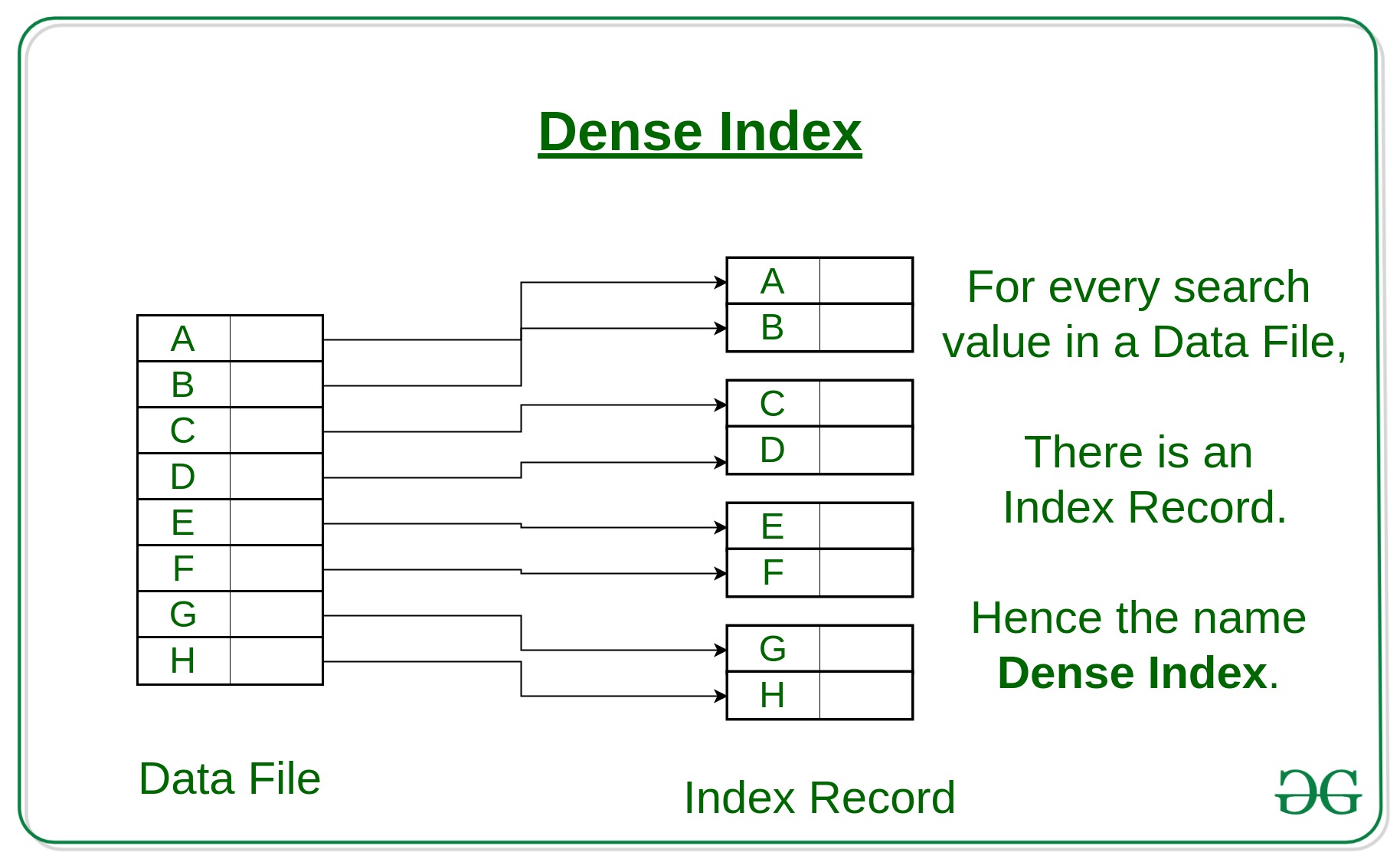
* **Access Types**: This refers to the type of access such as value based search, range access, etc.
* **Access Time**: It refers to the time needed to find particular data element or set of elements.
* **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
* **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
* **Space Overhead**: It refers to the additional space required by the index.

In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

**Sequential File Organization or Ordered Index File:** In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

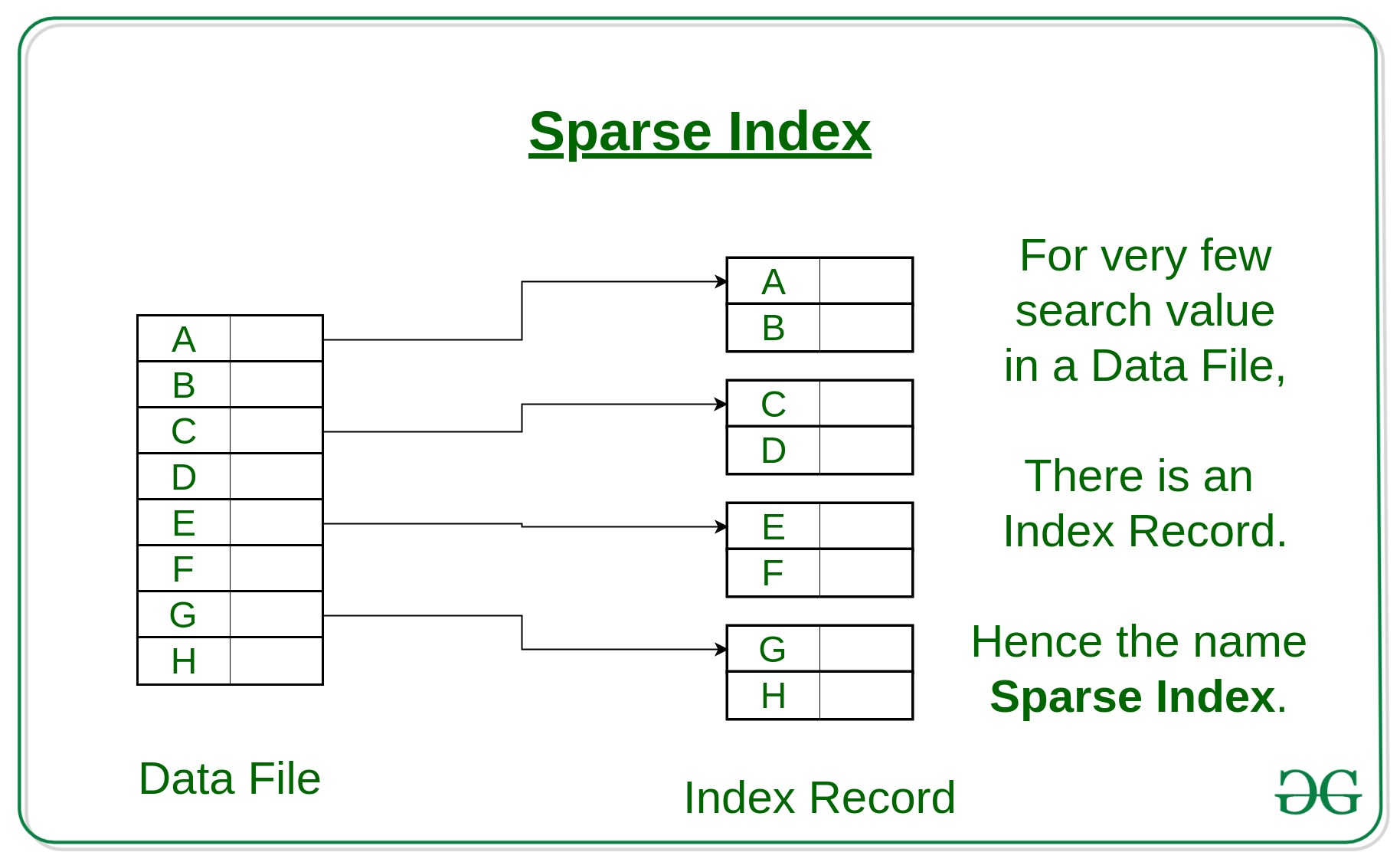
**Dense Index:**

* For every search key value in the data file, there is an index record.
* This record contains the search key and also a reference to the first data record with that search key value.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183521/Dense-Index.jpg)

**Sparse Index:**

* The index record appears only for a few items in the data file. Each item points to a block as shown.
* To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
* We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183518/Sparse-Index.jpg)

1. **Hash File organization:** Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by a function called a hash function.

**Indexes in DBMS**

[**https://www.javatpoint.com/indexing-in-dbms**](https://www.javatpoint.com/indexing-in-dbms)

**Normalization**

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let’s discuss about anomalies first then we will discuss normal forms with examples.

**Anomalies in DBMS**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

**Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp\_id for storing employee’s id, emp\_name for storing employee’s name, emp\_address for storing employee’s address and emp\_dept for storing the department details in which the employee works. At some point of time the table looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_dept |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

The above table is not normalized. We will see the problems that we face when a table is not normalized.

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

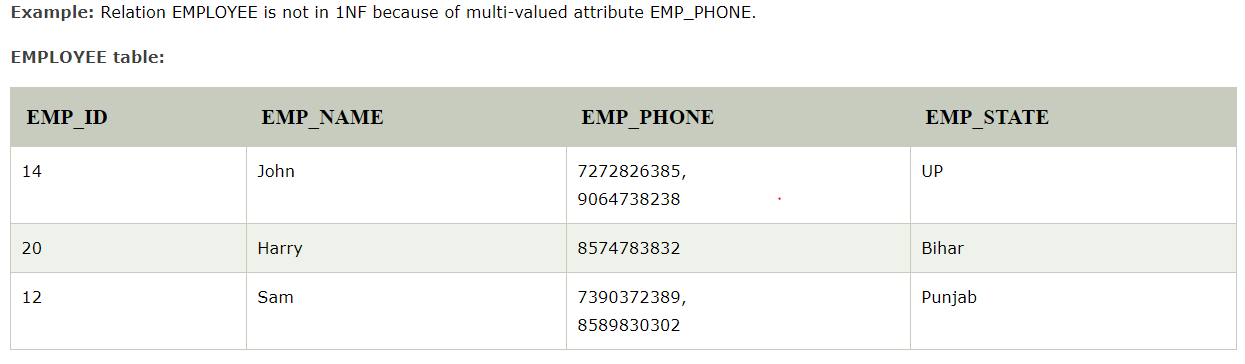
**Normalization**

Here are the most commonly used normal forms:

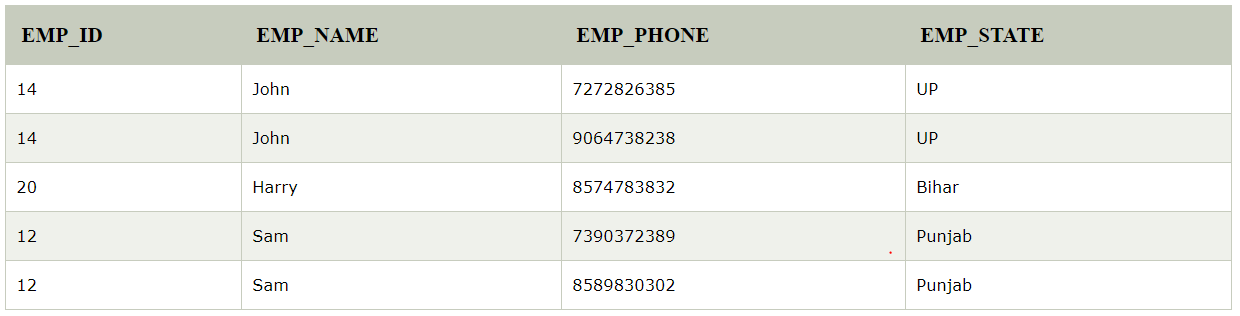
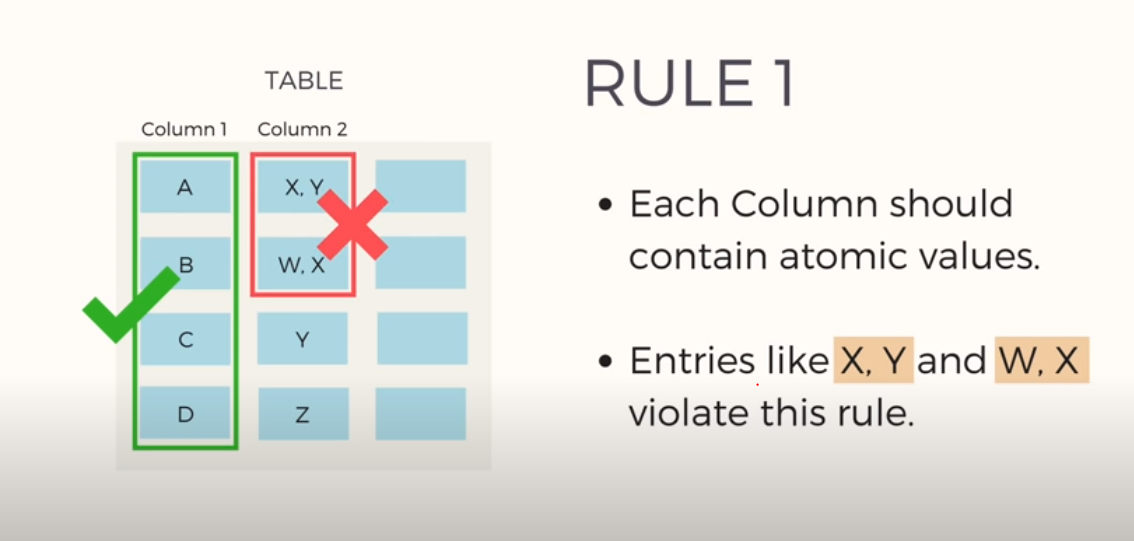
* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce & Codd normal form (BCNF)

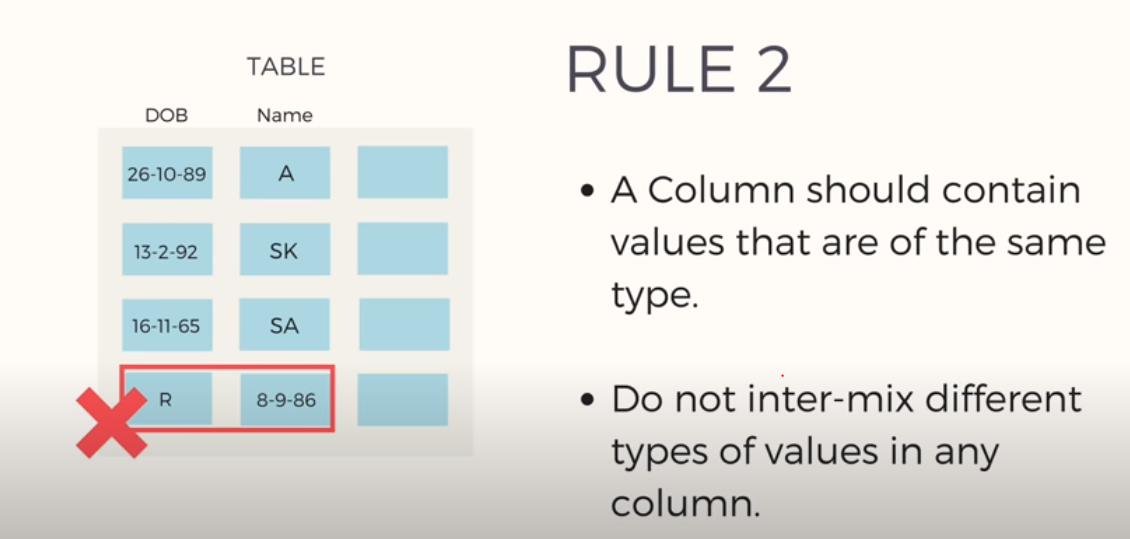
## First normal form (1NF)

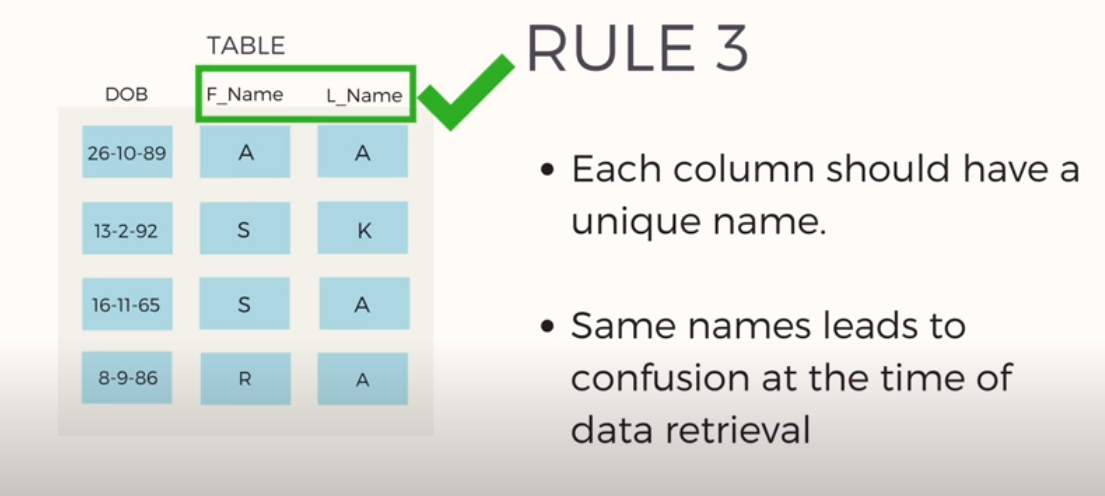
As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

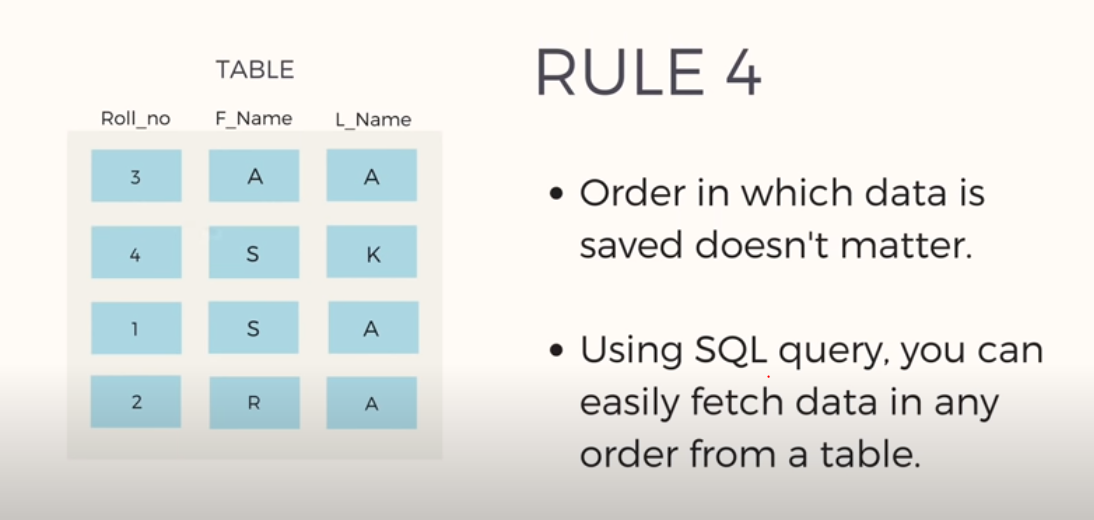


The decomposition of the EMPLOYEE table into 1NF has been shown below





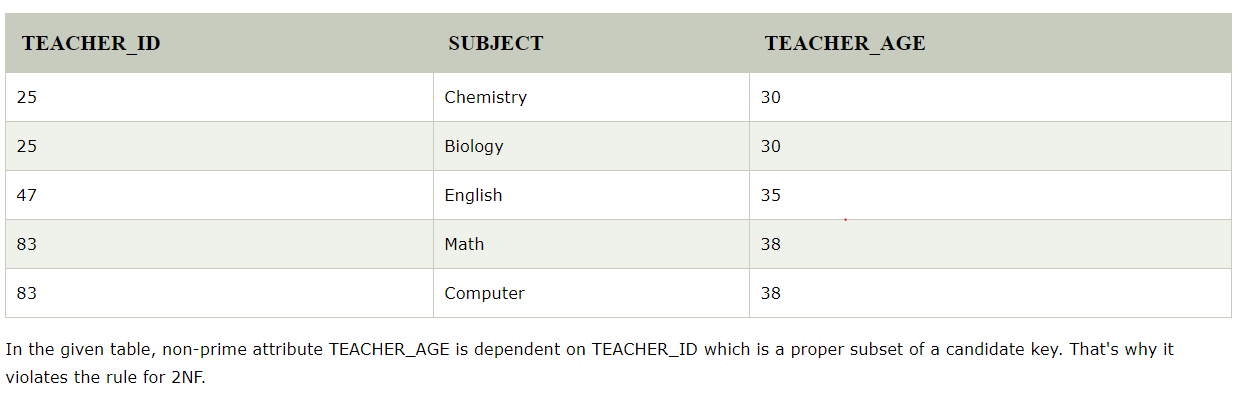


### 2. Second Normal Form –

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency.

**Example:** Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher..

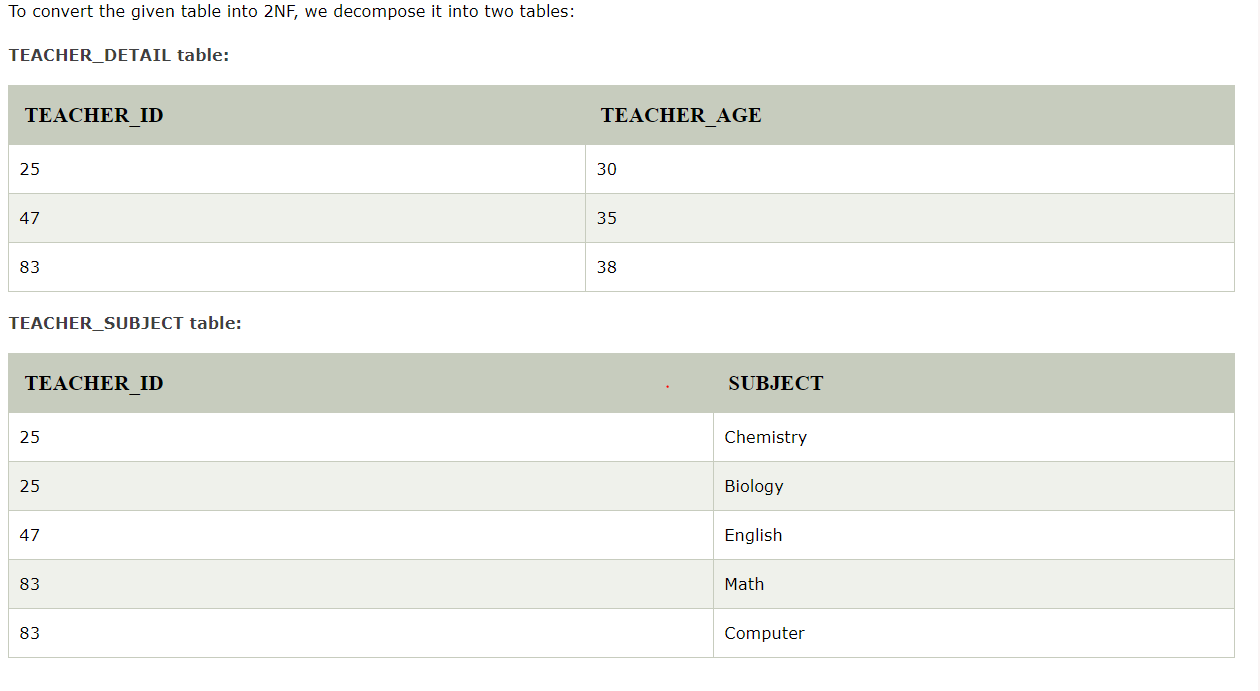
**TEACHER table**



**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”.

To make the table complies with 2NF we can break it in two tables like this



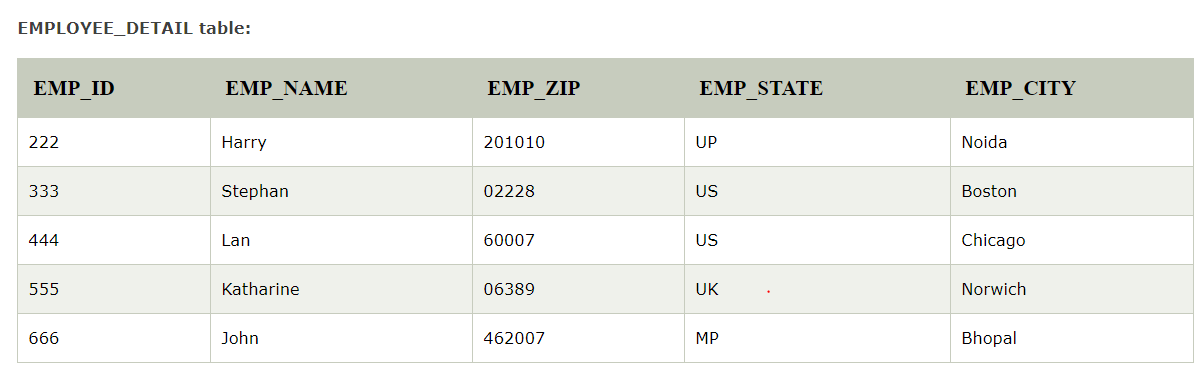
**Reference:**

[**https://www.youtube.com/watch?v=R7UblSu4744**](https://www.youtube.com/watch?v=R7UblSu4744)

### 3. Third Normal Form –

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form

An attribute that is not part of any [candidate key](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.



**Super key in the table above:**

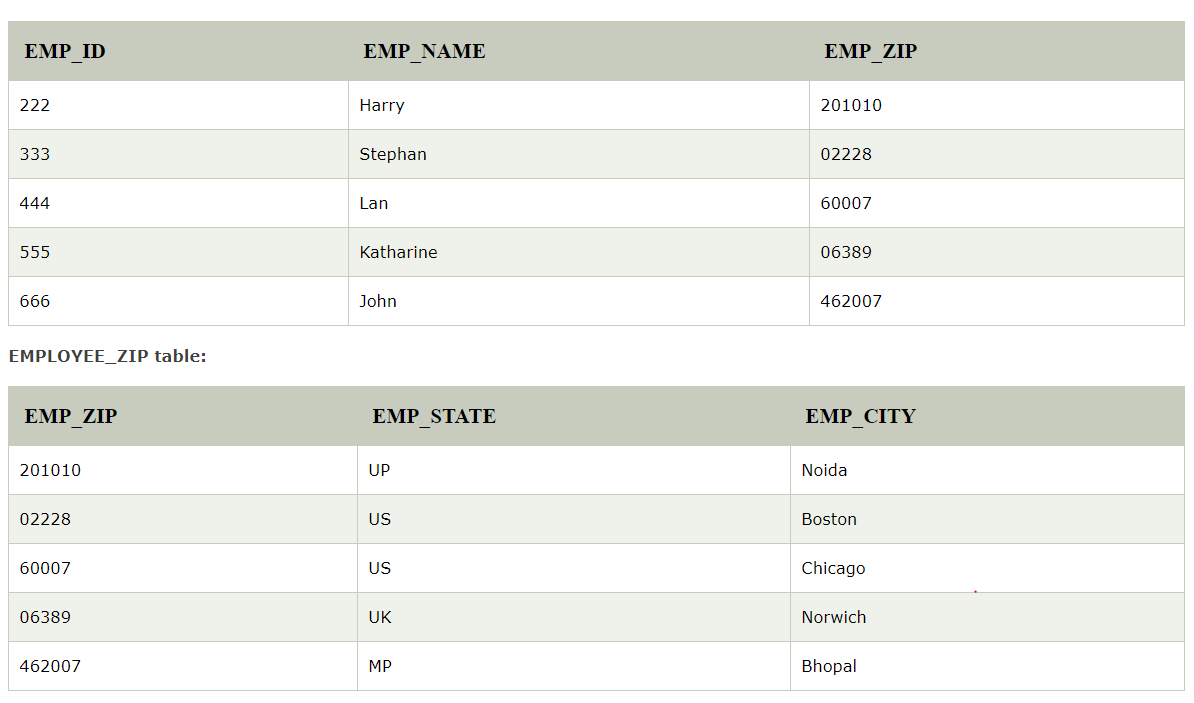
* 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

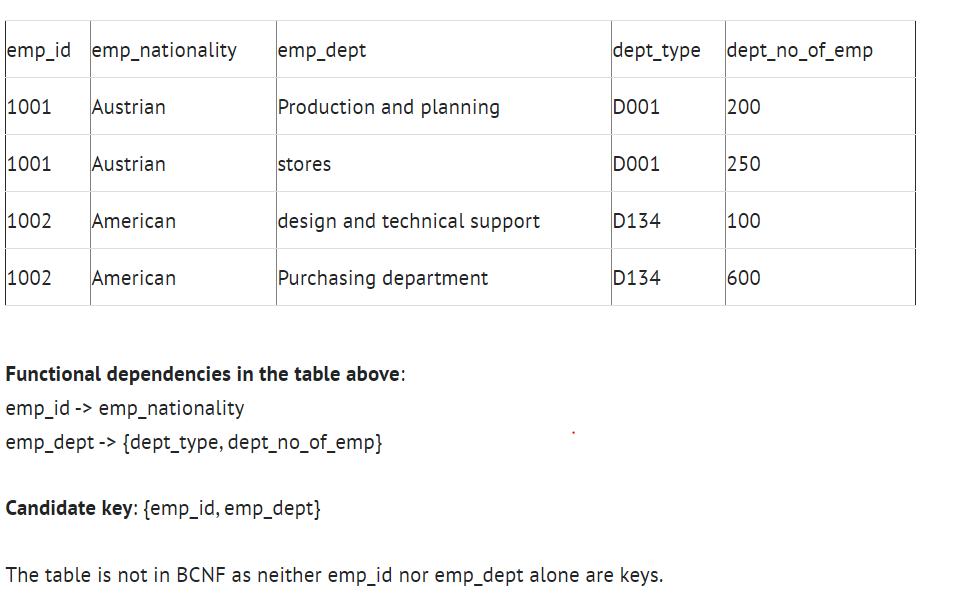
That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.



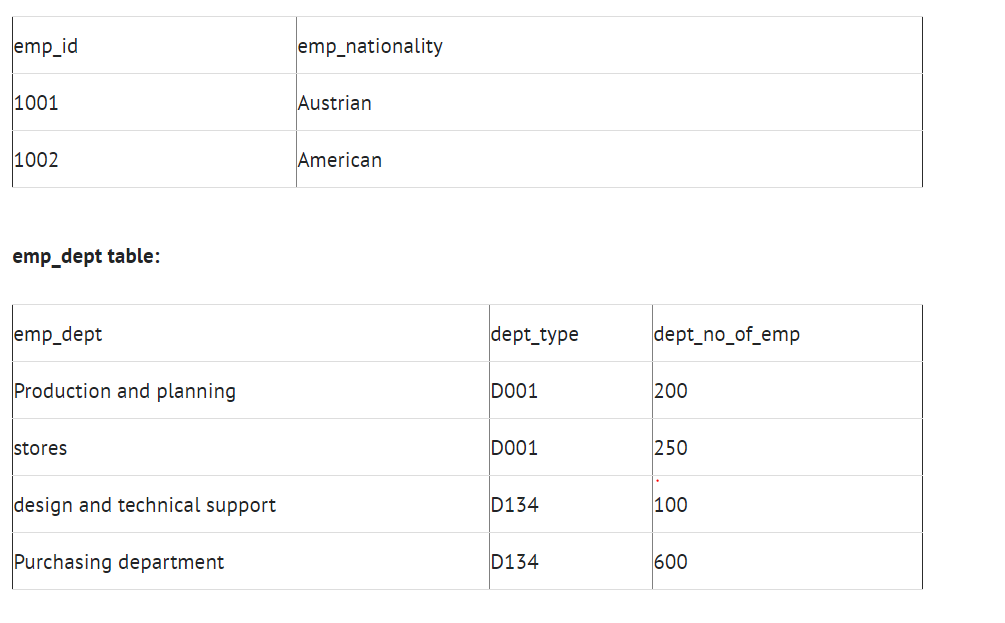
**Reference :** [**https://www.youtube.com/watch?v=aAx\_JoEDXQA**](https://www.youtube.com/watch?v=aAx_JoEDXQA)

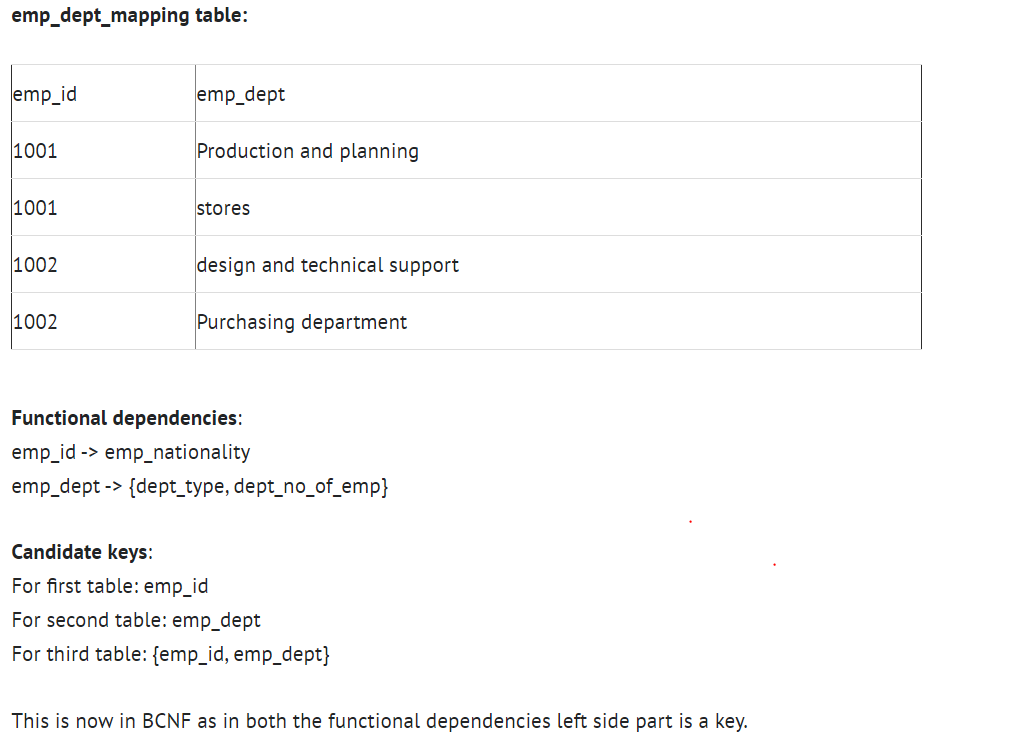
## Boyce Codd normal form (BCNF)

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: 

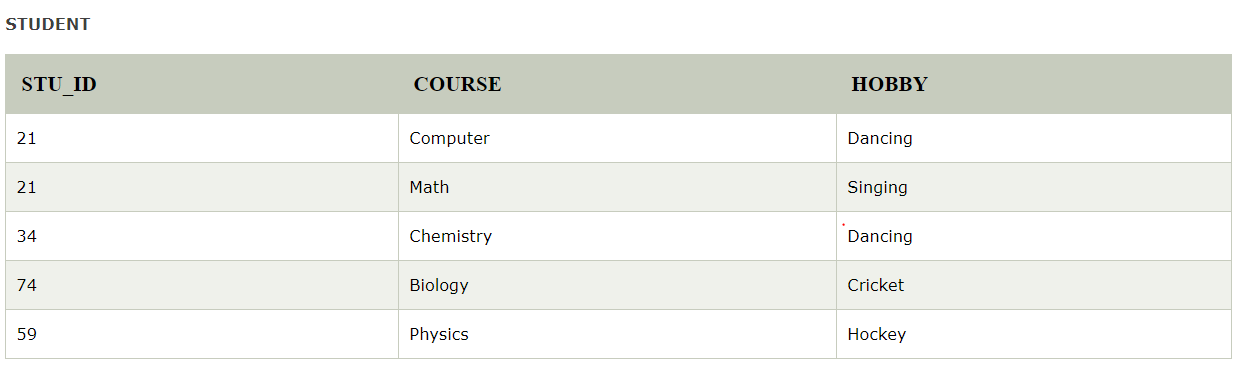
To make the table comply with BCNF we can break the table in three tables like this:





# **Fourth normal form (4NF)**

* A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
* For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.



The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

