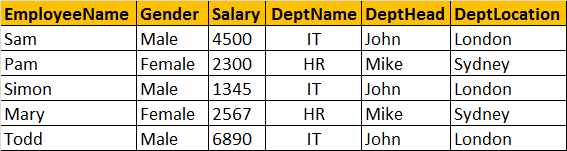
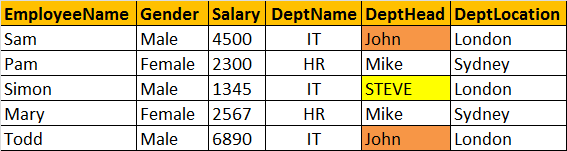
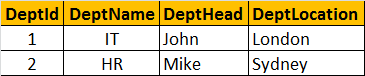
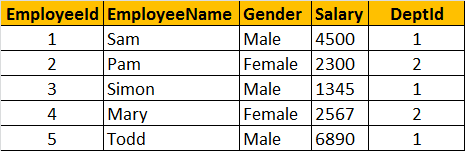
**Database Normalization - Part 52**

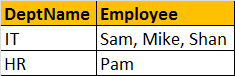
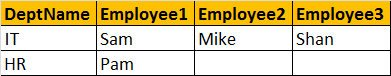
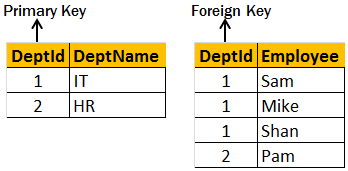
[**https://csharp-video-tutorials.blogspot.com/2012/09/database-normalization-part-52.html**](https://csharp-video-tutorials.blogspot.com/2012/09/database-normalization-part-52.html)

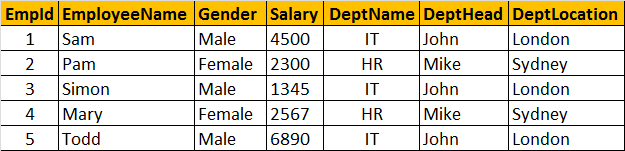
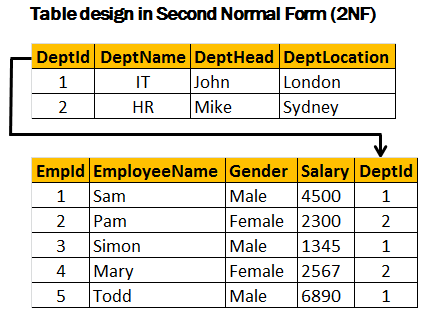
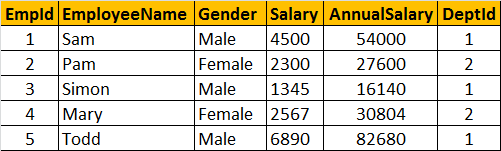
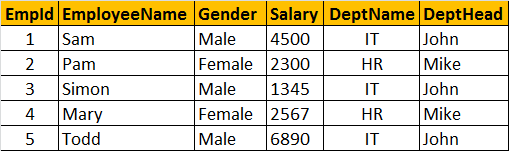
**https://csharp-video-tutorials.blogspot.com/2012/09/second-normal-form-and-third-normal.html**

[**https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql**](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql)

**What are the goals of database normalization**?  
or  
**Why do we normalize databases**?  
or   
**What is database normalization**?  
  
**Database normalization** is the process of organizing data to minimize data redundancy (data duplication), which in turn ensures data consistency.

**Let's understand with an example**, how**redundant data**can cause **data inconsistency**. Consider **Employees**table below. For every employee with in the same department, we are repeating, all the 3 columns (DeptName, DeptHead and DeptLocation). Let's say for example, if there 50 thousand employees in the IT department, we would have unnecessarily repeated all the 3 department columns (DeptName, DeptHead and DeptLocation) data 50 thousand times. The obvious problem with redundant data is the disk space wastage.  
  
  
**Another common problem, is that data can become inconsistent.** For example, let's say, JOHN has resigned, and we have a new department head (STEVE) for IT department. At present, there are 3 IT department rows in the table, and we need to update all of them. Let's assume I updated only one row and forgot to update the other 2 rows, then obviously, the data becomes inconsistent.  
  
  
**Another problem**, DML queries (Insert, update and delete), **could become slow**, as there could many records and columns to process.  
  
**So, to reduce the data redundancy**, we can divide this large badly organised table into two (Employees and Departments), as shown below. Now, we have reduced redundant department data. So, if we have to update department head name, we only have one row to update, even if there are 10 million employees in that department.  
  
**Normalized Departments Table**  
  
  
**Normalized Employees Table**  
  
  
**Database normalization is a step by step process.** There are 6 normal forms, First Normal form (1NF) thru Sixth Normal Form (6NF). Most databases are in third normal form (3NF). There are certain rules, that each normal form should follow.  
  
**Now, let's explore the first normal form** (1NF). A table is said to be in 1NF, if

1. The data in each column should be **atomic**. No multiple values, sepearated by comma.  
2. The table does not contain any **repeating column groups**  
3. Identify each record **uniquely using primary key**.  
  
**In the table below, data in Employee column is not atomic**. It contains multiple employees seperated by comma. From the data you can see that in the IT department, we have 3 employees - Sam, Mike, Shan. Now, let's say I want to change just, SHAN name.**It is not possible, we have to update the entire cell.** Similary it is not possible to select or delete just one employee, as the data in the cell is not atomic.  
  
  
**The 2nd rule of the first normal form is that, the table should not contain any repeating column groups**. Consider the Employee table below. We have repeated the Employee column, from Employee1 to Employee3. The problem with this design is that, if a department is going to have more than 3 employees, then we have to **change the table structure** to add Employee4 column. Employee2 and Employee3 columns in the HR department are NULL, as there is only employee in this department. The **disk space is simply wasted.**  
  
  
**To eliminate the repeating column groups, we are dividing the table into 2**. The repeating Employee columns are moved into a seperate table, with a foreign key pointing to the primary key of the other table. We also, introduced primary key to uniquely identify each record.  


In this video will learn about second normal form (2NF) and third normal form (3NF)  
**A table is said to be in 2NF, if**  
1. The table meets all the **conditions of 1NF**  
2. Move **redundant** data to a separate table  
3. Create **relationship** between these tables using foreign keys.  
  
**The table below violates second normal form**. There is lot of redundant data in the table. Let's say, in my organization there are 100,000 employees and only 2 departments (**IT & HR**). Since we are storing **DeptName, DeptHead and DeptLocation** columns also in the same table, all these columns should also be repeated 100,000 times, which results in unnecessary duplication of data.  
  
  
**So this table is clearly violating the rules of the second normal form**, and the redundant data can cause the following issues.  
1. Disk space wastage  
2. Data inconsistency  
3. DML queries (Insert, Update, Delete) can become slow  
  
**Now, to put this table in the second normal form**, we need to break the table into 2, and move the redundant department data (**DeptName, DeptHead and DeptLocation**) into it's own table. To link the tables with each other, we use the **DeptId** foreign key. The tables below are in 2NF.  
  
  
**Third Normal Form (3NF):**  
**A table is said to be in 3NF, if the table**  
1. Meets all the conditions of **1NF and 2NF**  
2. Does not contain columns (attributes) that are not fully **dependent upon the primary key**  
  
**The table below, violates third normal form**, because **AnnualSalary** column is not fully dependent on the primary key **EmpId**. The **AnnualSalary** is also dependent on the **Salary** column. In fact, to compute the **AnnualSalary**, we multiply the **Salary** by **12**. Since **AnnualSalary** is not fully dependent on the primary key, and it can be computed, we can remove this column from the table, which then, will adhere to 3NF.  
  
  
**Let's look at another example of Third Normal Form violation**. In the table below, **DeptHead** column is not fully dependent on **EmpId** column. **DeptHead** is also dependent on **DeptName**. So, this table is not in **3NF**.  
  
  
**To put this table in 3NF, we break this down into 2**, and then move all the columns that are not fully dependent on the primary key to a separate table as shown below. This design is now in 3NF.  
