\* Normalization (Suitable for online transaction processing systems (OLTP))

*Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity*

The process of taking a database design, and apply a set of formal criteria and rules, is called Normal Forms

The database normalization process is further categorized into the following types:

1. First Normal Form (1 NF)
2. Second Normal Form (2 NF)
3. Third Normal Form (3 NF)
4. Boyce Codd Normal Form or Fourth Normal Form ( BCNF or 4 NF)
5. Fifth Normal Form (5 NF)
6. Sixth Normal Form (6 NF)

One of the driving forces behind database normalization is to streamline data by reducing redundant data. Redundancy of data means there are multiple copies of the same information spread over multiple locations in the same database

A database is considered third normal form if it meets the requirements of the first 3 normal forms

A database is considered third normal form if it meets the requirements of the first 3 normal forms.

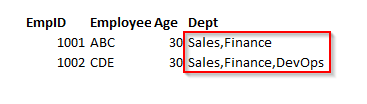
First Normal Form (1NF):

The first normal form requires that a table satisfies the following conditions:

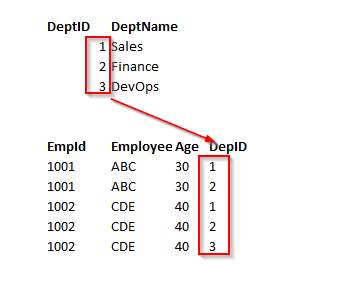
1. Rows are not ordered
2. Columns are not ordered
3. There is duplicated data
4. Row-and-column intersections always have a unique value
5. All columns are “regular” with no hidden values

In the following example, the first table clearly violates the 1 NF. It contains more than one value for the Dept column. So, what we might do then is go back to the original way and instead start adding new columns, so, Dept1, Dept2, and so on. This is what’s called a repeating group, and there should be no repeating groups. In order to bring this First Normal Form, split the table into the two tables. Let’s take the department data out of the table and put it in the dept table. This has the one-to-many relationship to the employee table.

Let’s take a look at the employee table:



Now, after normalization, the normalized tables Dept and Employee looks like below:



Second Normal Form and Third Normal Form are all about the relationship between the columns that are the keys and the other columns that aren’t the key columns.

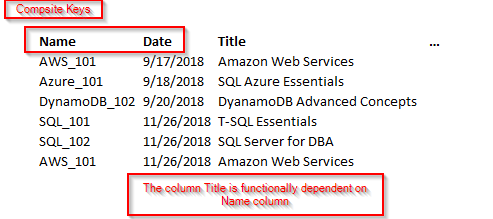
Second Normal Form (2NF):

An entity is in a second normal form if all of its attributes depend on the whole primary key. So this means that the values in the different columns have a dependency on the other columns.

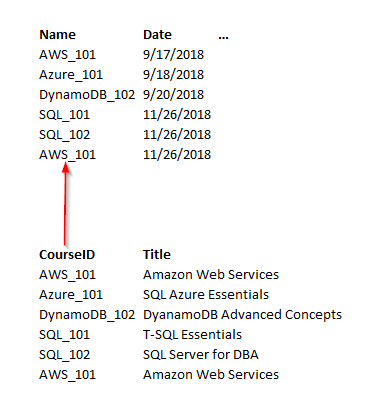
1. The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY
2. The partial dependencies are removed and placed in a separate table

Note: Second Normal Form (2 NF) is only ever a problem when we’re using a composite primary key. That is, a primary key made of two or more columns.

The following example, the relationship is established between the Employee and Department tables.



In this example, the *Title* column is functionally dependent on *Name and Date* columns. These two keys form a composite key. In this case, it only depends on Name and partially depndent on the Date column. Let’s remove the course details and form a separate table. Now, the course details are based on the entire key. We are not going to use a composite key.

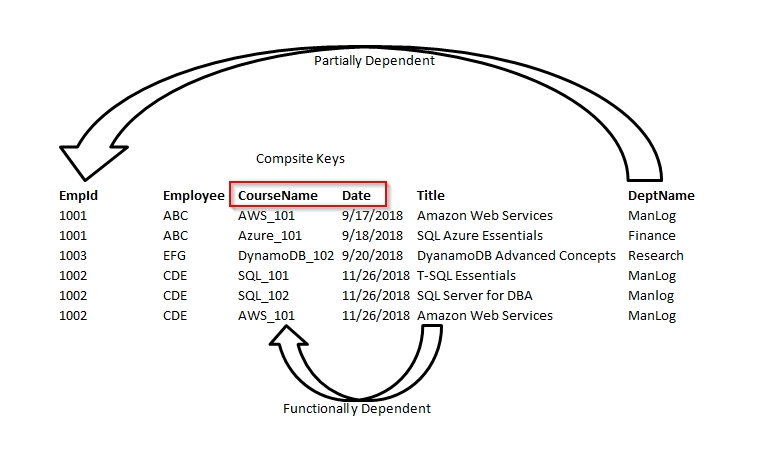


Third Normal Form (3NF):

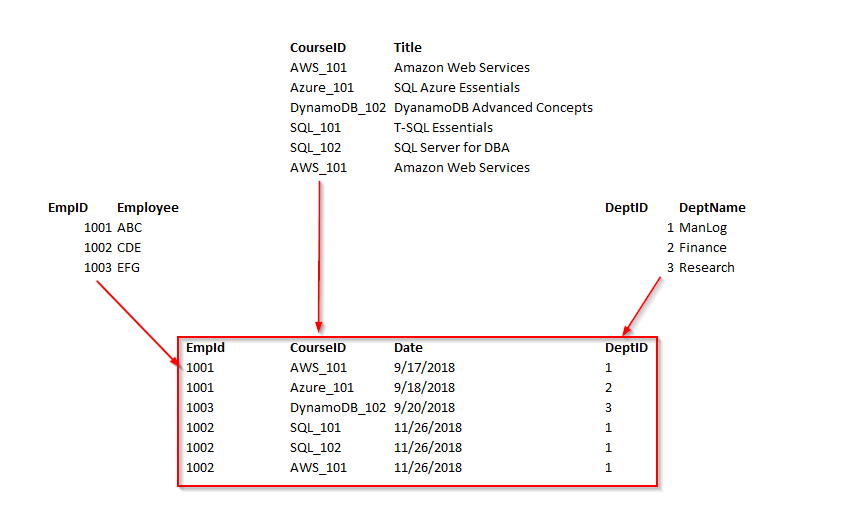
The third normal form states that you should eliminate fields in a table that do not depend on the key.

1. A Table is already in 2 NF
2. Non-Primary key columns shouldn’t depend on the other non-Primary key columns
3. There is no transitive functional dependency

Consider the following example, in the table employee; empID determines the department ID of an employee, department ID determines the department name. Therefore, the department name column indirectly dependent on the empID column. So, it satisfies the transitive dependency. So this cannot be in third normal form.



In order to bring the table to 3 NF, we split the employee table into two.



Now, we can see the all non-key columns are fully functionally dependent on the Primary key.

Although a fourth and fifth form does exist, most databases do not aspire to use those levels because they take extra work and they don’t truly impact the database functionality and improve performance.

**Denormalization** (Suitable for online analytical processing systems (OLAP))

According to [Wikipedia](https://en.wikipedia.org/wiki/Denormalization)…

*“Denormalization is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.[1][2] It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations. Denormalization should not be confused with Unnormalized form. Databases/tables must first be normalized to efficiently denormalize them.”*

Database normalization is always a starting point for denormalization. Denormalization is a type of reverse engineering process that can apply to retrieve the data in the shortest time possible.

\* Primary-key constraint

Enforces the uniqueness of rows and also disallows NULLs in the constraint attributes.

To enforce the uniqueness of the logical primary-key constraint, SQL Server will create

a unique index behind the scenes. A unique index is a physical mechanism used by

SQL Server to enforce uniqueness

\* Unique constraint

- SQL Server will create a unique index behind the scenes as the physical mechanism to enforce

the logical unique constraint.

- Two NULLs value were equal to each other

- If you create a unique constraint then by default a non-clustered Index has been created. You can create a maximum of 999 (Non-Clustered Indexes) per table as per limitations.

\* Foreign-key constraint

he FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

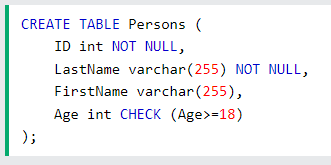
A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table

\* Check constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.



**\* Default constraint**

The [**DEFAULT**](https://www.geeksforgeeks.org/mysql-default-function/)**Constraint** is used to fill a column with a default and fixed value. The value will be added to all new records when no other value is provided.

**\* Inheritance**

**\* Table-per-Type**

Table-per-type inheritance uses a separate table in the database to maintain data for non-inherited properties and key properties for each type in the inheritance hierarchy

**\* Table-per-Hierarchy**

By default, EF maps the inheritance using the *table-per-hierarchy* (TPH) pattern. TPH uses a single table to store the data for all types in the hierarchy, and a discriminator column is used to identify which type each row represents.

**\* Index**

An index is a disk-based structure linked to a table or view that facilitates quicker row retrieval. A table or view’s table or view’s columns are used to create keys in an index. These keys are kept in a structure (B-tree) that enables SQL Server to quickly and effectively locate the row or rows that correspond to the key values.

**\* Clustered**

A clustered index is created only when both the following conditions satisfy –

1. The data or file, that you are moving into secondary memory should be in sequential or sorted order.
2. There should be a key value, meaning it can not have repeated values.

**\* Nonclustered**

A nonclustered index is an index structure separate from the data stored in a table that reorders one or more selected columns. Nonclustered indexes can often help you find data more quickly than searching the underlying table; queries can sometimes be answered entirely by the data in the nonclustered index, or the nonclustered index can point the Database Engine to the rows in the underlying table.

*Clustered indexes are faster for retrieving large ranges of sequential data, while non-clustered indexes are faster for retrieving small sets of data or for sorting and aggregating data*

**What is a table scan in SQL?**

A table scan is the easiest and simplest operation that can be performed against a table. It sequentially processes all the rows in the table to determine if they satisfy the selection criteria specified in the query. It does this processing in a way to maximize the I/O throughput for the table.

When performing a table scan, the query optimizer reads all the rows in the table, and extracts the rows that meet the criteria of the query. A table scan generates many disk I/O operations and can be resource intensive. However, a table scan could be the most efficient method if, for example, the result set of the query is a high percentage of rows from the table.

**\* Three-Valued predicate logic**

SQL’s three valued logic is a consequence of supporting null to mark absent data. If a null value affects the result of a logical expression, the result is neither true nor false but unknown.

The three-valued logic is an integral part of Core SQL and it is followed by pretty much every SQL database.

**\* Logical query processing**

[Explanation](https://www.sqlservercentral.com/blogs/sql-server-logical-query-processing)

\* All-at-once operations

\* Self-Contained sub query

\* Scalar

\* Multivalued

\* Correlated sub query

\* Table expression

\* Derived table

\* View

\* CTE

\* Recursive CTE

\* Inline table-valued function

\* Cross and outer apply operator

\* Window functions

\* Ranking

\* ROW\_NUMBER

\* RANK

\* DENSE\_RANK

\* NTILE

\* Offset

\* LEAD

\* LAG

\* FIRST\_VALUE

\* LAST\_VALUE

\* Inserting data

\* INSERT VALUES

\* INSERT SELECT

\* INSERT EXEC

\* SELECT INTO

\* BULK INSERT

\* The identity property and the sequence object

\* Deleting data

\* Delete statement

\* Truncate

\* Merge statement

\* The OUTPUT clause and nested DML

\* Transactions and isolation levels

\* ACID property

\* Isolation Levels

\* READ UNCOMMITTED

\* Dirty reads

\* READ COMMITTED

\* \*Non-Repeatable Reads\* or \*Inconsistent Analysis\*

\* REPEATABLE READ

\* Prevent phenomenon called a lost update

\* SERIALIZABLE

\* Prevent phenomenon called phantom reads

\* Isolation levels based on row versioning

\* SNAPSHOT

\* READ COMMITTED SNAPSHOT

\* Deadlocks

\* SQL Server chooses to terminate the transaction that did the least work

(based on the activity written to the transaction log)

\* DEADLOCK\_PRIORITY

\* Deadly embrace deadlock

\* Temporary tables

\* Local

\* Global

\* Table variable

\* Dynamic SQL

\* The \*EXEC\* command

\* The \*sp\_executesql\* stored procedure

\* User-defined functions

\* Scalar

\* table-valued

\* Stored procedure

\* trigger

\* DML

\* After

\* Permanent tables

\* Instead of

\* Permanent tables

\* Views

\* DDL

\* Database scope

\* For events with a database scope, such as CREATE TABLE

\* Server scope

\* For events with a server scope, such as CREATE DATABASE

\* SQL Server supports only after DDL triggers; it doesn’t support instead of DDL triggers

\* Can be used for

\* Auditing

\* Policy enforcement

\* Change management