Database Normalization

**Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints. It is accomplished by applying some formal rules either by a process of synthesis (creating a new database design) or decomposition (improving an existing database design)."**

**Database Normalization is a set of rules that are applied to a database, such that the schema of the database ensures that all the rules are being followed. These rules are also known as Normal Forms and are widely used while designing database solutions.**

The database normalization process can be divided into following types:

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

**First Normal Form (1NF)**

* **Data is stored in tables with rows that can be uniquely identified by a Primary Key.**
* **Data within each table is stored in individual columns in its most reduced form.**
* **There are no repeating groups**.

**Second Normal Form (2NF)**

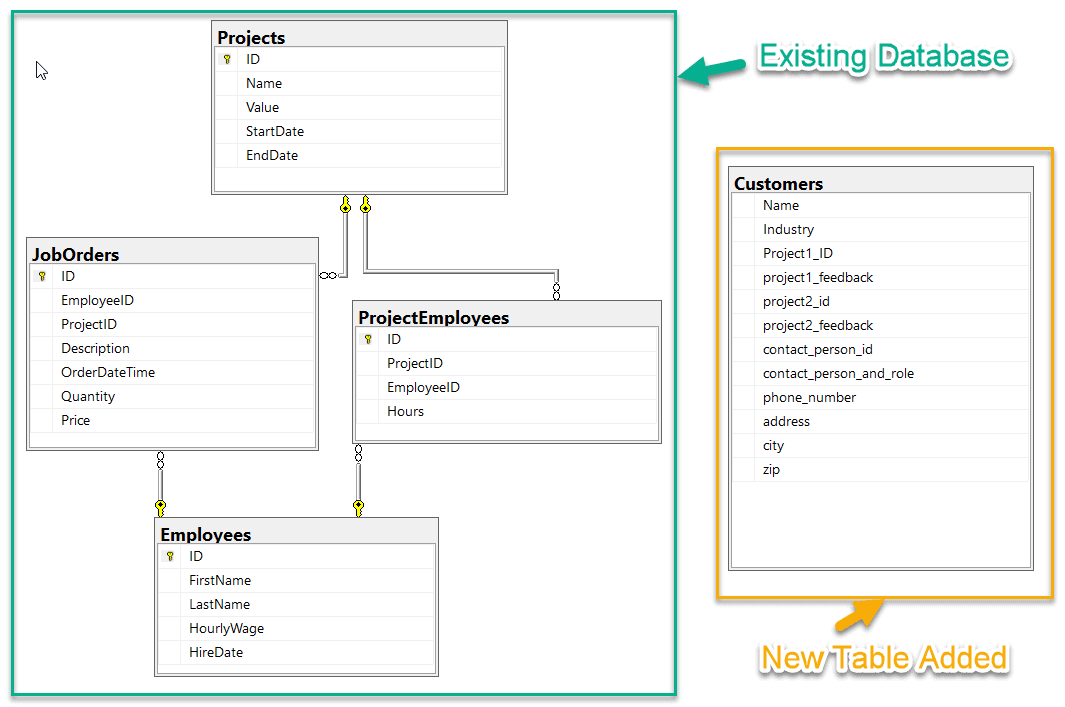
* **All the rules from 1NF must be satisfied.**
* **Only those data that relates to a table’s primary key is stored in each table.**

Third Normal Form (3NF)

* All the rules from 2NF must be satisfied.
* There should be no intra-table dependencies between the columns in each table.

Example.

***Let us consider the following database schema. As you can see in Fig 1, there are four tables (Existing Database) - Projects, Employees, ProjectEmployees, and JobOrders. Recently, the Customers table has also been added to the database to store the customers' information. As you can see in the diagram below, the Customers table has not been designed in a proper way to support the normal forms, let's go ahead and fix it.***

[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e365b819bde8.png)Fig 1 - Initial Database Schema

CREATE TABLE Projects(

[ID] INT PRIMARY KEY IDENTITY,

[Name] VARCHAR(100),

[Value] DECIMAL(5,2),

StartDate DATE,

EndDate DATE

)

GO

CREATE TABLE Employees(

[ID] INT PRIMARY KEY IDENTITY,

[FirstName] VARCHAR(50),

[LastName] VARCHAR(50),

[HourlyWage] DECIMAL(5,2),

[HireDate] DATE

)

GO

CREATE TABLE ProjectEmployees(

[ID] INT PRIMARY KEY IDENTITY,

[ProjectID] INT,

[EmployeeID] INT,

[Hours] DECIMAL(5,2),

CONSTRAINT FK\_ProjectEmployees\_Projects FOREIGN KEY ([ProjectID]) REFERENCES [Projects] ([ID]),

CONSTRAINT FK\_ProjectEmployees\_Employees FOREIGN KEY ([EmployeeID]) REFERENCES [Employees] ([ID])

)

GO

CREATE TABLE JobOrders(

[ID] INT PRIMARY KEY IDENTITY,

[EmployeeID] INT,

[ProjectID] INT,

[Description] TEXT,

[OrderDateTime] DATETIME,

[Quantity] INT,

[Price] DECIMAL(5,2),

CONSTRAINT FK\_JobOrders\_Projects FOREIGN KEY ([ProjectID]) REFERENCES [Projects] ([ID]),

CONSTRAINT FK\_JobOrders\_Employees FOREIGN KEY ([EmployeeID]) REFERENCES [Employees] ([ID])

)

GO

CREATE TABLE Customers (

[Name] VARCHAR(100),

[Industry] VARCHAR(100),

[Project1\_ID] INT,

[Project1\_Feedback] TEXT,

[Project2\_ID] INT,

[Project2\_Feedback] TEXT,

[ContactPersonID] INT,

[ContactPersonAndRole] VARCHAR(255),

[PhoneNumber] VARCHAR(12),

[Address] VARCHAR(255),

[City] VARCHAR(255),

[Zip] VARCHAR(5)

)

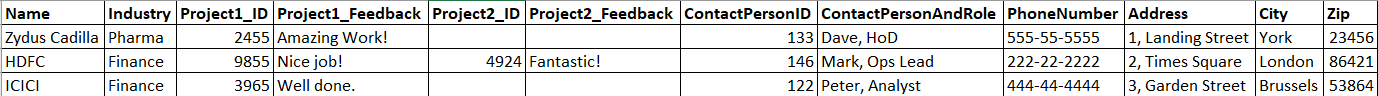
GO

First Normal Form.

The *Customers* table in the diagram violates all the three rules of the first normal form.

1. We do not see any Primary Key in the table.
2. The data is not found in its most reduced form. For example, the column *ContactPersonAndRole* can be divided further into two individual columns - *ContactPerson* and *ContactPersonRole*.
3. Also, we can see there are two repeating groups of columns in this table - (*Project1\_ID*, *Project1\_FeedBack*) and (*Project2\_ID*, *Project2\_Feedback*). We need to get these removed from this table.

The diagram below shows dummy data stored in the *Customers* table.

[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e365f3ce0bf5.png)Fig 2 - Customers Table Data

**The first thing that we need to do is to add a primary key to this table. For this, we can add a new column *ID* with datatype as *INT* and also assign it as an *Identity* column.** The script is given below.

ALTER TABLE [Customers]

ADD [ID] INT IDENTITY PRIMARY KEY

GO

When you execute this script, a new column gets added at the end of all the columns. This is the primary key of the table and now it satisfies the first rule of the First Normal Form.

**Secondly, we need to split the column *ContactPersonAndRole* into two individual columns.** This can be done in two steps as follows:

1. Rename the original column from *ContactPersonAndRole* to *ContactPerson*.
2. Add a new column for *ContactPersonRole*.

The script below, when executed, will rename the original column and add a new column to store the *ContactRole* information.

sp\_rename 'Customers.[ContactPersonAndRole]', 'ContactPerson', 'COLUMN'

GO

ALTER TABLE [Customers]

ADD [ContactPersonRole] VARCHAR(20)

GO

**Finally, in order to satisfy the third rule of the First Normal Form, we need to move the columns *Project1\_ID*, *Project1\_Feedback*, *Project2\_ID*, and *Project2\_Feedback* into a new table. This can be done by creating a new table *ProjectFeedbacks* and link it back with the *Customers* and the *Projects* table.**

When the script below is executed, it will remove the above-mentioned columns from the *Customers* table and create a new table *ProjectFeedbacks* with Foreign Key references to the *Customers* and *Projects* table.

ALTER TABLE [Customers]

DROP COLUMN Project1\_ID

ALTER TABLE [Customers]

DROP COLUMN Project1\_Feedback

ALTER TABLE [Customers]

DROP COLUMN Project2\_ID

ALTER TABLE [Customers]

DROP COLUMN Project2\_Feedback

GO

CREATE TABLE ProjectFeedback(

[ID] INT PRIMARY KEY IDENTITY,

[ProjectID] INT,

[CustomerID] INT,

[Feedback] TEXT,

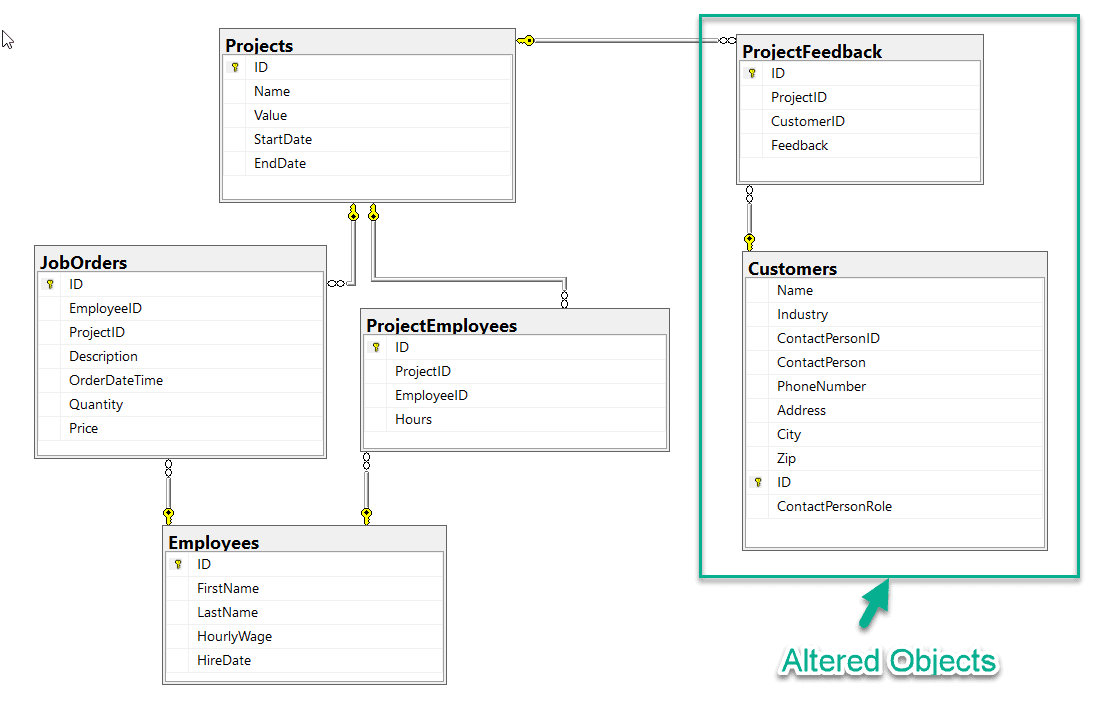
CONSTRAINT FK\_ProjectFeedbacks\_Projects FOREIGN KEY ([ProjectID]) REFERENCES [Projects] ([ID]),

CONSTRAINT FK\_ProjectFeedbacks\_Customers FOREIGN KEY ([CustomerID]) REFERENCES [Customers] ([ID])

)

GO

The database schema after applying all the rules of the first normal form is as below.

[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e37ac3fb5912.png)Fig 3 - First Normal Form Diagram

As you can see, the *Customers* table has been altered and a new table *ProjectFeedbacks* has been added into the schema. Thus, there are no repeating groups in the *Customers* or the *ProjectFeedbacks* table. We can also know about the feedbacks as it refers to both the *Customers* and the *Projects* table.

Now, that the Customers table supports 1NF, let's go ahead and apply the second normal form.

Second Normal Form

To satisfy the conditions of the second normal form, all the rules of the first normal form should satisfy. And along with that, we also need to ensure that all the columns in the table relate directly to the primary key of the record in the table.

However, if you see the database schema diagram above (*Fig 3*), you can see that the *ContactPerson*, *ContactPersonRole* and the *PhoneNumber* do not directly relate to the *ID* of the *Customers* table. That is because the primary key refers to a customer and not to any person or role or the phone number of the contact person. If ever, the contact person for a customer changes, we would have to update all of these columns, running the risk that we will update the values in one of the columns but forget to modify the other.

So, in order to satisfy this rule, we need to remove these three columns from the *Customers* table and put them in a separate table. This table should contain data that is related only to the contact person and not the customer.

Let us remove all these columns from the *Customers* table which do not relate to the primary key of the table directly. The script below removes the three columns from the table as these are not related to the customer, instead of to the contact person only.

ALTER TABLE [Customers]

DROP COLUMN ContactPerson

ALTER TABLE [Customers]

DROP COLUMN ContactPersonRole

ALTER TABLE [Customers]

DROP COLUMN PhoneNumber

GO

Once, the columns are removed from the *Customers* table, we need to create a new table that'll store the data for the contact persons. Let us create a new table *ContactPersons* and relate it to the *Customers* table with a foreign key relation. The script is provided below.

CREATE TABLE ContactPersons(

[ID] INT PRIMARY KEY IDENTITY,

[ContactPerson] VARCHAR(100),

[ContactPersonRole] VARCHAR(20),

[PhoneNumber] VARCHAR(12)

)

GO

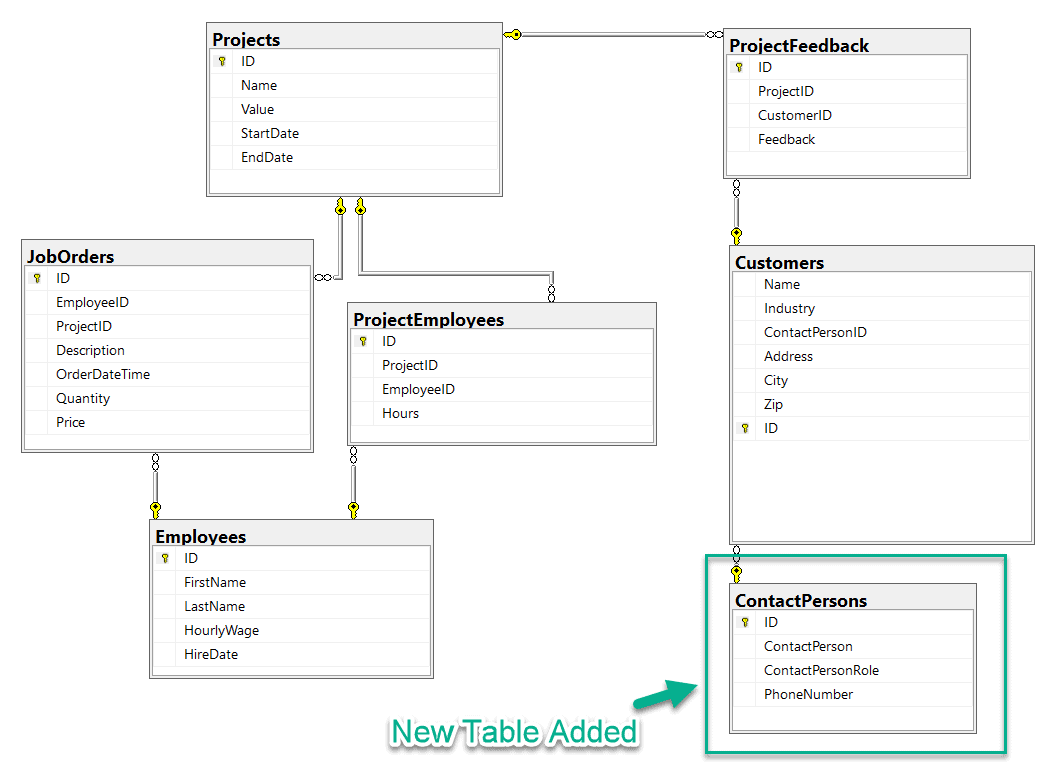
ALTER TABLE [Customers]

ADD CONSTRAINT FK\_Customers\_ContactPersons FOREIGN KEY ([ContactPersonID])

REFERENCES ContactPersons([ID])

GO

Once this script is executed, you can see in the diagram below (*Fig 4*) that a new table has been added to the schema and now it satisfies the second normal form of the database.

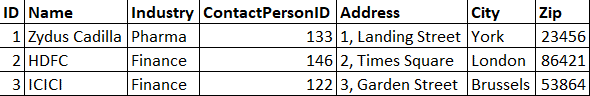
[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e37abd765696.png)Fig 4 - Second Normal Form Diagram

Now, if the contact person for customer changes, we just need to insert a record into the *ContactPersons* table and change the *ContactPersonID* in the *Customers* table.

Third Normal Form

To satisfy the conditions of the third normal form, all the rules of the second normal form must satisfy. And with that, we also need to ensure that each column must be **non-transitively dependent** on the primary key of the table. This means that all columns in a table should rely only on the primary key and no other column. If ColumnA relies on the primary key and also on the ColumnB, then ColumnA is known to be transitively dependent on the primary key and it violates the third normal form.

After applying 1NF and 2NF, below is what the Customers table looks like now (*Fig 5*).

[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e3669a372c30.png)Fig 5 - Customers table after 2NF

If you look carefully, there are transitive dependent columns in this table and it violates the 3NF. The transitive dependent relationship is between the columns - *City* and *Zip*. The city in which a customer is situated relates to the primary key of the customer, so this satisfies the second normal form. However, the city also depends on the zip code. If a customer changes its location, there may be a chance we update one column but not the other. Because of this relationship between the *City* and *Zip*, the database is not in 3NF.

In order to fix this and bring the table to satisfy the third normal form, we need to remove the *City* from the *Customers* table and create a new table *ZipCodes* to store the *Zip* and *City*. This new table can be related to the *Customers* table via a foreign key relation. The script is provided below.

ALTER TABLE [Customers]

DROP COLUMN City

GO

CREATE TABLE ZipCodes(

[ZipID] VARCHAR(5) PRIMARY KEY,

[City] VARCHAR(255)

)

GO

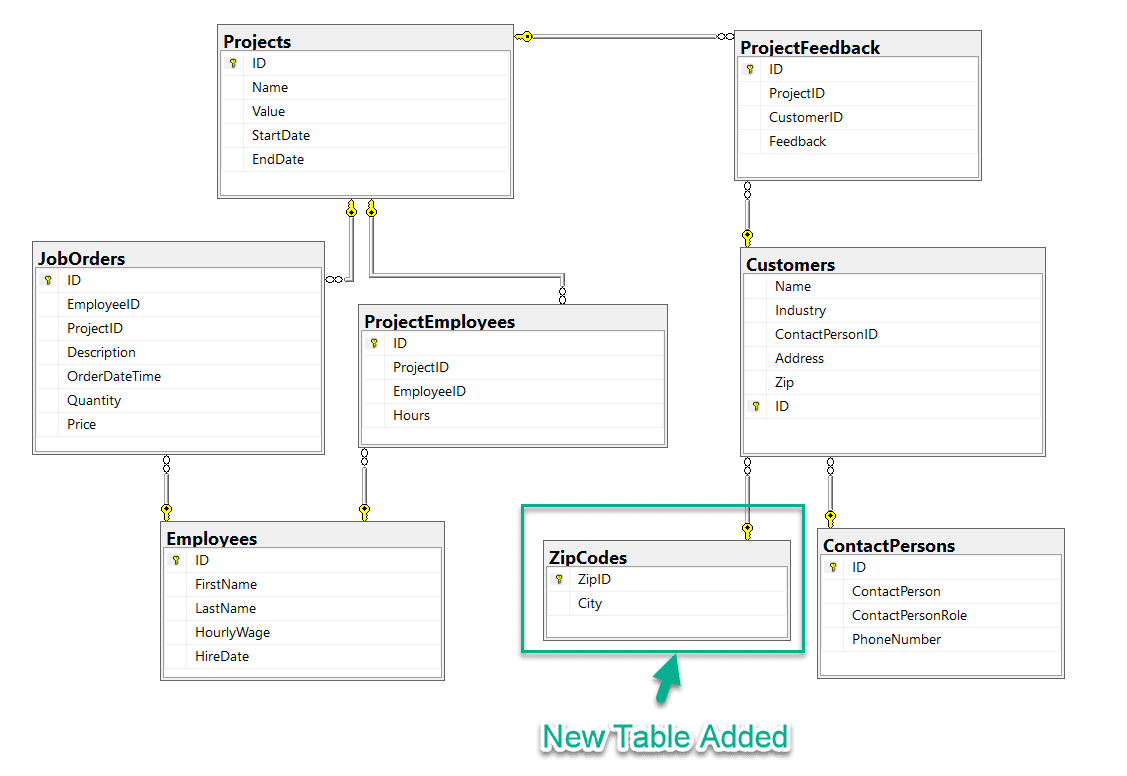
ALTER TABLE [Customers]

ADD CONSTRAINT FK\_Customers\_ZipCodes FOREIGN KEY ([Zip])

REFERENCES ZipCodes([ZipID])

GO

Now that all the changes are performed, lets look at the schema after the third normal form has also been satisfied (*Fig 6*).  As you can see, the new table *ZipCodes* has been added and it relates to the *Customers* table.

[](https://www.sqlservercentral.com/wp-content/uploads/2020/02/img_5e37ab33158c5.png)Fig 6 - Third Normal Form Diagram

That's all for the third normal form. The *Customers* table now supports all the three normal forms and can be used as required. It is always tricky to find issues that are caused by a violation of the third normal form. However, for good database design, these are quite essential that all the normal forms are satisfied.

Summary

In this article, we have seen what is database normalization and how can we implement it in a SQL Server database. To learn more please follow the link below.

* [Database normalization article from Microsoft](https://support.microsoft.com/en-in/help/283878/description-of-the-database-normalization-basics)
* [Database Denormalization](https://www.vertabelo.com/blog/denormalization-when-why-and-how/)