# SQL SERVER – How to an Add Identity Column to Table in SQL Server

Here is the question I received on SQLAuthority Fan Page.

“How do I add an identity column to Table in SQL Server? “

Sometime the questions are very very simple but the answer is not easy to find.

### Scenario 1:

If you are table does not have identity column, you can simply add the identity column by executing following script:

ALTER TABLE MyTable

ADD ID INT IDENTITY(1,1) NOT NULL

### Scenario 2:

If your table already has a column which you want to convert to identity column, you can’t do that directly. There is a workaround for the same which I have discussed in depth over the article [**Add or Remove Identity Property on Column**](https://blog.sqlauthority.com/2009/05/03/sql-server-add-or-remove-identity-property-on-column/).

### Scenario 3:

If your table has already identity column and you can want to add another identity column for any reason – that is not possible. A table can have only one identity column. If you try to have multiple identity column your table, it will give following error.

Msg 2744, Level 16, State 2, Line 2  
Multiple identity columns specified for table ‘MyTable‘. Only one identity column per table is allowed.

How to Reset IDENTITY Column Values in SQL Server

Do you want to reset an identity column in your table in SQL Server so the numbers are in order?

And you want to do it without recreating the table?

In this article, I’ll show you how you can do this, and a few things to be aware of.

Let’s take a look.

What is an Identity Column?

An identity column is a feature in [SQL Server](https://www.databasestar.com/sql-server/) that lets you auto-increment the column’s value. This is helpful for [primary keys](https://www.databasestar.com/database-keys/), where you don’t care what the number is, as long as it’s unique.

You can specify the word IDENTITY as a property after the data type when creating or altering a table.

For example:

CREATE TABLE product (

product\_id INT IDENTITY,

product\_name VARCHAR(200),

price INT

);

This statement will create a new table called product.

The product\_id has the word IDENTITY after it, which means new records will have an automatically generated value.

Let’s see an example of this.

INSERT INTO product (product\_name, price)

VALUES ('Chair', 100);

We can check the value was inserted by selecting data from the table.

SELECT product\_id, product\_name, price

FROM product;

|  |  |  |
| --- | --- | --- |
| **product\_id** | **product\_name** | **price** |
| 1 | Chair | 100 |

We can insert a second row into the table to see what the product\_id identity column will be set to.

INSERT INTO product (product\_name, price)

VALUES ('Desk', 250);

To see the records in the table, we can select from it.

SELECT product\_id, product\_name, price

FROM product;

|  |  |  |
| --- | --- | --- |
| **product\_id** | **product\_name** | **price** |
| 1 | Chair | 100 |
| 2 | Desk | 250 |

Without specifying the product\_id in the INSERT statements, the IDENTITY feature has generated a number and populated the row. A new number is created for each row.

Why Reset an Identity Column?

So what’s the issue? You may want to reset an identity column if you delete records from the table, or if you get an error when inserting a row.

Let’s delete a record and insert a new one.

DELETE FROM product

WHERE product\_id = 2;

INSERT INTO product (product\_name, price)

VALUES ('Large Desk', 300);

Once we delete a row and insert a new one, here’s what our table looks like.

SELECT product\_id, product\_name, price

FROM product;

|  |  |  |
| --- | --- | --- |
| **product\_id** | **product\_name** | **price** |
| 1 | Chair | 100 |
| 3 | Large Desk | 300 |

We can see that the new row has a product\_id of 3, and not 2.

Our ID values are not in order.

So, if you want them to be in order, you’ll have to reset the identity column.

Identity Values Don’t Matter

**If you’re here because you want your primary key identity values to be in numerical order without any gaps, then I would suggest it’s not necessary.**

A primary key value should hold no significance to any user or application outside of the database. Its only purpose is to uniquely identify a row and therefore be used to relate to other rows in other tables.

It doesn’t matter to the database if the values are in the order of 1, 2, 3, or if there are missing values like 1, 4, 8. As long as they are unique, the database will still operate in the same way.

If someone looks at an output of a database (e.g. a report that has IDs) and wonders why they are not in order or that there are gaps, then either:

* you can explain to them that the ID is OK to have gaps
* if they want numbers with no gaps, you can change the query to generate a row number instead

If someone is using the identity values to determine how many rows are in a table, and the column is “skipping” values like this, then they may think there are more rows in the table than there actually are.

However, the right way to count records is to use the [COUNT function](https://www.databasestar.com/sql-count/), not by looking at the identity column as a row count.

*So, to summarise, it doesn’t matter what the value of the identity column is. As long as it’s unique, it’s OK.*

People don’t need to see the value.

Having said that, if you really want to reset the identity values (e.g. for a test you’re doing or for a university project), then you can do it in SQL Server.

Check the Current Value Using the DBCC CHECKIDENT Procedure

You can run the DBCC CHECKIDENT procedure in SQL Server to see what the current value of the identity column is. You just specify the name of the table as the parameter.

Here’s an example:

DBCC CHECKIDENT('product');

This will show you the current value of this table’s identity column.

Checking identity information: current identity value '3', current column value '3'.

DBCC execution completed. If DBCC printed error messages,

contact your system administrator.

The value of 3 is the most recent value of the identity column. The next row that’s inserted will have the value of 4.

So, how do we reset the value?

Reset the Identity Value Using the DBCC CHECKIDENT Procedure

If you want to reset the identity column in SQL Server, you can use the DBCC CHECKIDENT procedure with extra parameters:

DBCC CHECKIDENT ('table\_name', RESEED, new\_value);

Resetting our produce table to use a value of 1 is done using this command:

DBCC CHECKIDENT ('product', RESEED, 0);

However, there are existing records in the table, so if we reset it and insert a new record, there will be an error.

So, what do we do? We need to:

1. Delete all data from the table
2. Reset the identity
3. Re-insert all data

Our script would look like this:

DELETE FROM product;

DBCC CHECKIDENT ('product', RESEED, 0);

INSERT INTO product (product\_name, price)

VALUES ('Chair', 100);

INSERT INTO product (product\_name, price)

VALUES ('Large Desk', 300);

This will reset the identity value to 0, causing the two inserted rows to have an ID of 1 and 2.

The result is:

|  |  |  |
| --- | --- | --- |
| **product\_id** | **product\_name** | **price** |
| 1 | Chair | 100 |
| 2 | Large Desk | 300 |

So that’s how you can reset an identity column in SQL Server.

Reset the Identity Value and Keep Table Data

What if you have a large table? It can be hard or impossible to re-insert all of the values from a script.

One way you can do this is to store the values in another table. The process would be:

1. Create a new table using the values from the real table
2. Delete the data from the real table
3. Reset the identity
4. Re-insert from the new table

Your script could look like this:

CREATE TABLE product\_backup AS

SELECT product\_id, product\_name, price

FROM product;

DELETE FROM product;

DBCC CHECKIDENT ('product', RESEED, 0);

INSERT INTO product (product\_name, price)

SELECT product\_name, price

FROM product\_backup

ORDER BY product\_id ASC;

There are a few things to note here. The product\_backup table stores the product\_id, which has the gaps, so it can be used in the order by later in the script. This is important if you want the data in the same order. This may not actually be important as you can just order your data in any future queries, but it’s useful if that’s what you want.

Also, in the [INSERT](https://www.databasestar.com/sql-insert-into/) statement, you select from the product\_backup table, but you don’t insert the product\_id from that table. The identity column is generated.

Your data will look the same as the earlier example:

|  |  |  |
| --- | --- | --- |
| **product\_id** | **product\_name** | **price** |
| 1 | Chair | 100 |
|  |  |  |
| 2 | Large Desk | 300 |

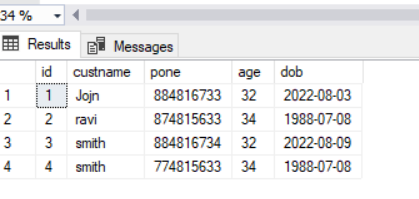
 select \* from Customers

delete Customers where id=2

insert into Customers values('ravi',874815633,34,'1988-07-08')

this will insert as 4 th record but we want to insert as 2 record

dbcc checkident('Customers',Reseed,1)



Reset the Identity Value: Delete vs Truncate

One issue with using this method is that the value that the CHECKIDENT procedure requires is different depending on if you TRUNCATE or DELETE.

In short:

* When you use [DELETE](https://www.databasestar.com/sql-delete/) to delete all rows in the table, the next assigned value for the identity column is the new reseed value + the current increment
* When you use [TRUNCATE](https://www.databasestar.com/sql-truncate-table/), the next assigned value for the identity column is the initial seed value.

Take a look at the article above for more details on the tests that were run.

Conclusion

Using an identity column for an auto-generating primary key is a great feature. The values of the identity column shouldn’t matter so it doesn’t matter if they are not continuous. But if you really need to make them continuous, you can reset them without dropping and recreating the table using the steps

# Insert Values in Identity Column in SQL Server

entity Column Values are identical in SQL. When creating such a column the auto-increment property can be set that creates a numeric sequence.  
  
I created a table named Customer and for the CustomerID column, "IDENTITY(1,1)" is specified. This means that, as each row is inserted into the table, SQL Server will automatically increment this value by 1 starting with the number 1.

CREATE TABLE Customer

(

ID int IDENTITY(1,1),

Name varchar(100),

Address varchar(200)

)

This is a point of interest because as we begin to write our INSERT statements, we will need to ensure we do not include this column because SQL Server will manage it for us.

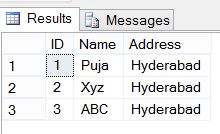
Insert INTO Customer (NAME,ADDRESS) Values('Puja','Hyderabad')

Insert INTO Customer (NAME,ADDRESS) Values('Xyz','Hyderabad')

Insert INTO Customer (NAME,ADDRESS) Values('ABC','Hyderabad')

Select \* from customer

Result:

  
  
So now let' see what happens if we want to insert a record with a specific CustomerID.

Insert INTO Customer (ID,NAME,ADDRESS) Values(4,'SSS','Hyderabad')

Result:

https://csharpcorner-mindcrackerinc.netdna-ssl.com/UploadFile/biswapinky/insert-values-in-identity-column-in-sql-server/Images/SQLIdentityColumn1.jpg  
  
SQL Server has thrown the preceding error when I attempt to insert a value into the ID column. For example, let's say a customer was deleted by mistake and you want to retain their original CustomerID. What would you do? If you inserted the record like we have in the examples above then the next highest number will be inserted, not the original value. Let's check out the example below to show how to solve this issue.  
  
**Insert Value to Identity field**  
  
Now, let's see how to insert our own values to identity the field ID within the Customer table.

SET IDENTITY\_INSERT Customer ON

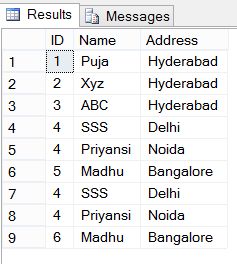
INSERT INTO Customer(ID,Name,Address) VALUES(4,'SSS','Delhi')

INSERT INTO Customer(ID,Name,Address) VALUES(4,'Priyansi','Noida')

SET IDENTITY\_INSERT Customer OFF

INSERT INTO Customer(Name,Address) VALUES('Madhu','Bangalore')

Result:

**  
  
Note**  
  
We usually use this trick when we have deleted some rows from the table and we want the data in a sequence.  
  
After inserting your own value into the identity field don't forget to set IDENTITY\_INSERT OFF.

# SQL SERVER – Having Two Identity Columns for A Single Table

The last section of my [**Comprehensive Database Performance Health Check**](https://blog.sqlauthority.com/comprehensive-database-performance-health-check/) is always questions and answers. In yesterday’s session, a DBA asked an interesting question that if it is possible to have two identity columns for a single table. The answer to this question – **Yes and No**.

Technically a table can have only one identity column and if you try to add another identity column to it, it will give you an error.

Here is an example

|  |  |
| --- | --- |
| 1  2  3  4 | USE TempDB  GO  CREATE TABLE dbo.TwoIdentity (ID INT IDENTITY (1,1) NOT NULL,  ID2 INT IDENTITY (1,1) NOT NULL) |

When you run the above syntax, it will give you the following error:

Msg 2744, Level 16, State 2, Line 3  
Multiple identity columns specified for table ‘TwoIdentity’. Only one identity column per table is allowed.

However, there is a workaround for the same. You can run using the **computed column** to create another identity.

Let us see a unique question which I had received from my client:

***“We want two identity columns in a single table. One identity column starting from 1 and increasing by an interval of 1 and another one starting from 9999 and decreasing by 1, can you help us to do that?”***

The answer is yes, it is possible to do that. Let us see the code for the same.

First, let us create a table with a computed column which actually uses the original identity column and decreases it by 1.

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE dbo.TwoIdentity (ID INT IDENTITY (1,1) NOT NULL,  SecondID AS 10000-ID,  TextDesc VARCHAR(100))  GO |

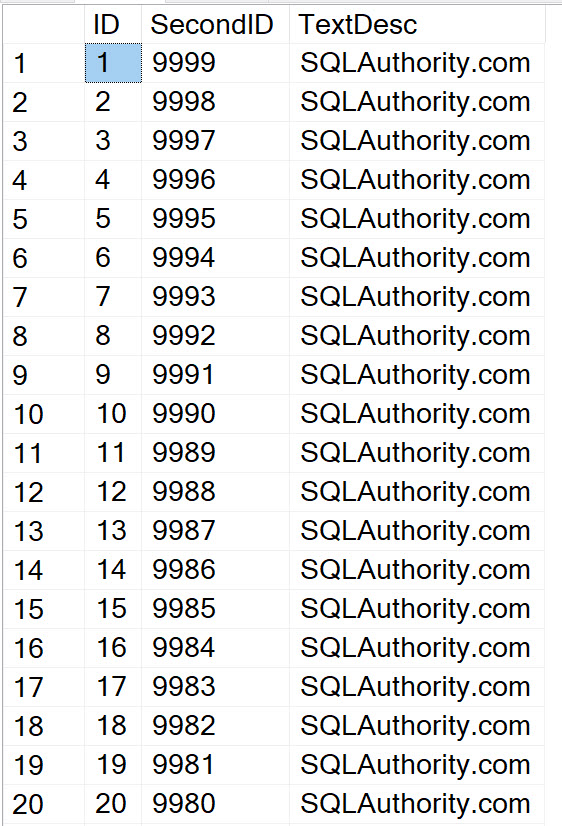
Next, let us populate the table with some sample data.

|  |  |
| --- | --- |
| 1  2  3 | INSERT INTO TwoIdentity (TextDesc)  VALUES ('SQLAuthority.com')  GO 100 |

Now let us select the data from the table.

|  |  |
| --- | --- |
| 1  2  3 | SELECT \*  FROM dbo.TwoIdentity  GO |

Here is the screenshot of the data.



In the screenshot, you can clearly see how the SecondID column acts as a second identity column which is decreasing from an initial number of 9999.

Here is how you can drop the table which you had created earlier.

|  |  |
| --- | --- |
| 1  2 | DROP TABLE dbo.TwoIdentity  GO |