# Lab Exercise: Working with MS SQL and LocalDB

This document is a walkthrough through the process of **installing and configuring MS SQL Server 2016 LocalDB (MSSQL LocalDB)** and **Microsoft SQL Server Management Studio (SSMS)**. After following all steps you will have fully installed and configured MSSQL LocalDB and SSMS.

|  |
| --- |
| If you use **old Windows** (e.g. Windows 7 or Windows XP), use **the other lab**, designed for Windows 7 and SQL Server 2014! |

This lab is part of the [“Software Technologies” course @ SoftUni](https://softuni.bg/courses/software-technologies).

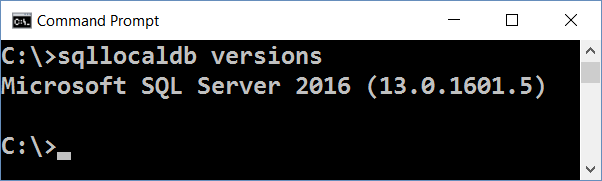
# Part I: SQL Server Installation

## Check If You Have LocalDB Already Installed

Open the command prompt (cmd.exe) and run the following command from the console:

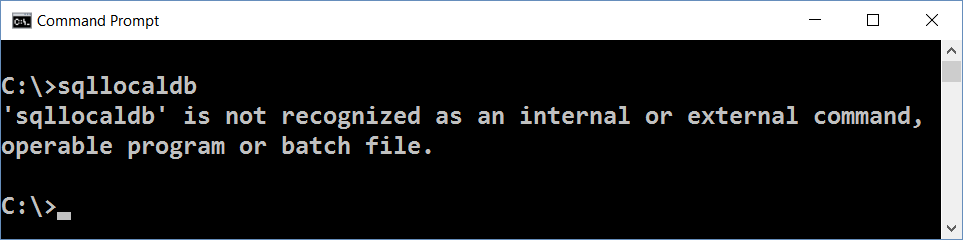
|  |
| --- |
| **sqllocaldb versions** |

This command will show you the versions of **LocalDB** that you have already installed:



In most cases when you install Visual Studio, it silently installs SQL Server LocalDB as well. The LocalDB version depends on the Visual Studio version. You might have or not have SQL LocalDB.

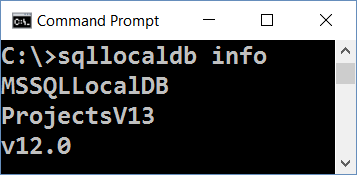
If you **do not have LocalDB installed**, an **error** like this will be shown:



Now you can list the available **LocalDB instances** with this command:

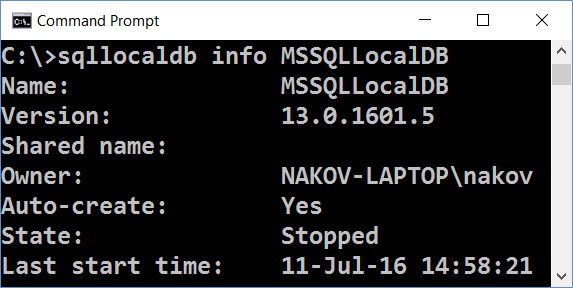
|  |
| --- |
| **sqllocaldb info** |

The **result** should be like at the example below. The list of LocalDB instances could vary depending on your installed software and instances created during the time:



Finally, you can **check the version** and other info for certain instance:

|  |
| --- |
| **sqllocaldb info MSSQLLocalDB** |

****

If you have LocalDB installed, skip to SQL Server installation steps. You **don’t need** to install MS SQL Server.

Note that you might have different **versions** of LocalDB in the same time:

* 13.0 means SQL Server 2016
* 12.0 means SQL Server 2014
* 11.0 means SQL Server 2012

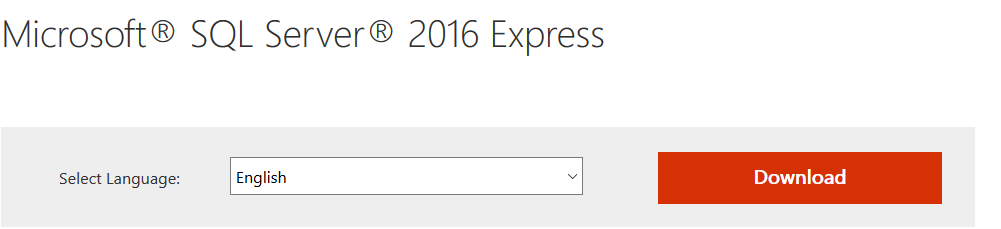
Any of the above versions will work correctly for the SQL Server exercises.

### Important Note

|  |
| --- |
| If you **have already installed SQL Server LocalDB** (some version), **you may** **skip the download and install steps** for SQL Server LocalDB. Install only SQL Server Management Studio (SSMS). |

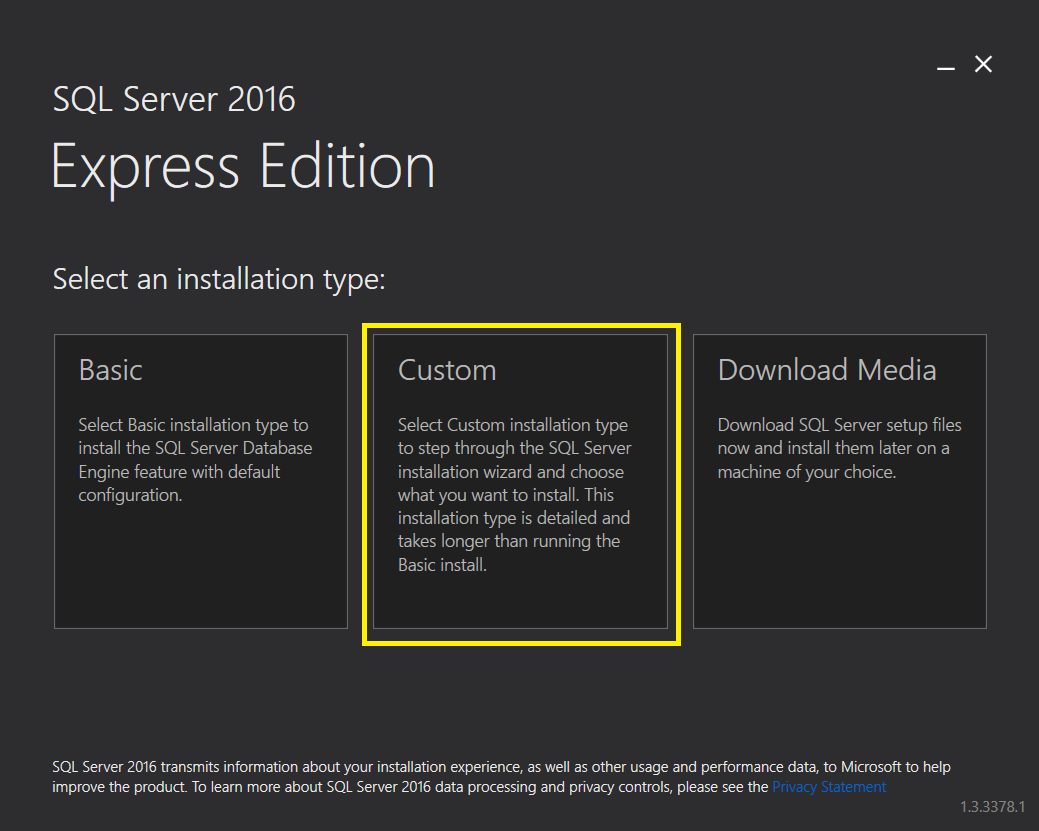
## Download MS SQL Server 2016 Express

Download **Microsoft SQL Server 2016 Express** from: <https://www.microsoft.com/en-us/download/details.aspx?id=52679>.

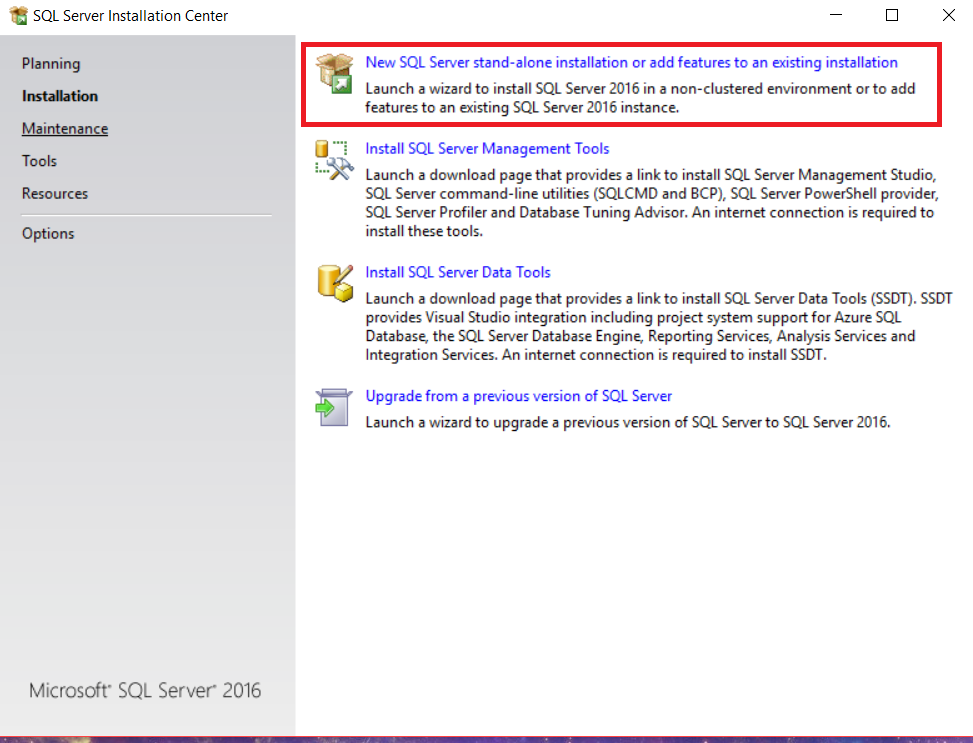


## Install MS SQL Server 2016 LocalDB

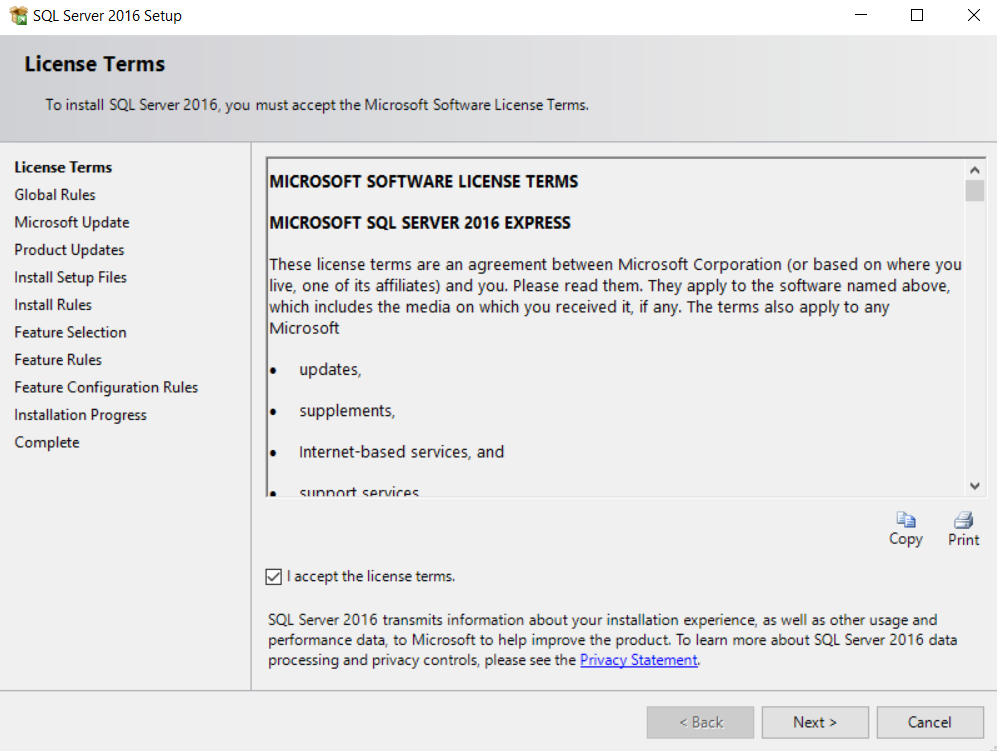
In this course we shall use [SQL Server 2016 Express LocalDB](https://msdn.microsoft.com/en-us/library/hh510202.aspx?f=255&MSPPError=-2147217396). This is a lightweight **free** version of SQL Server.



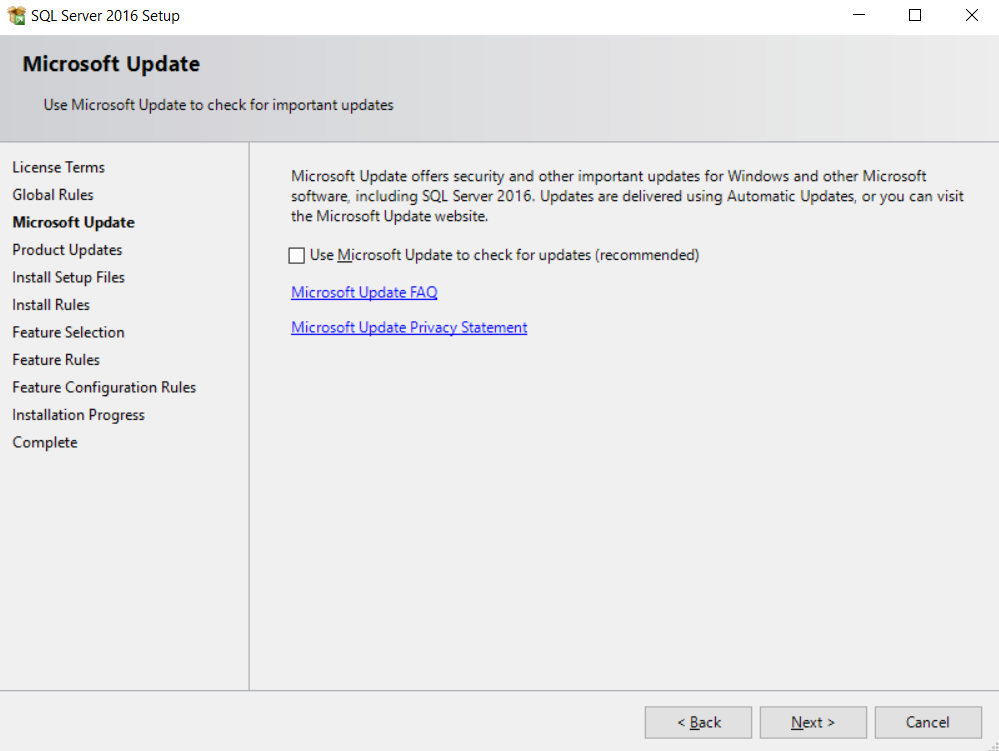
After choosing **Custom** the installer will start the SQL Server Installation Center, that looks like this:



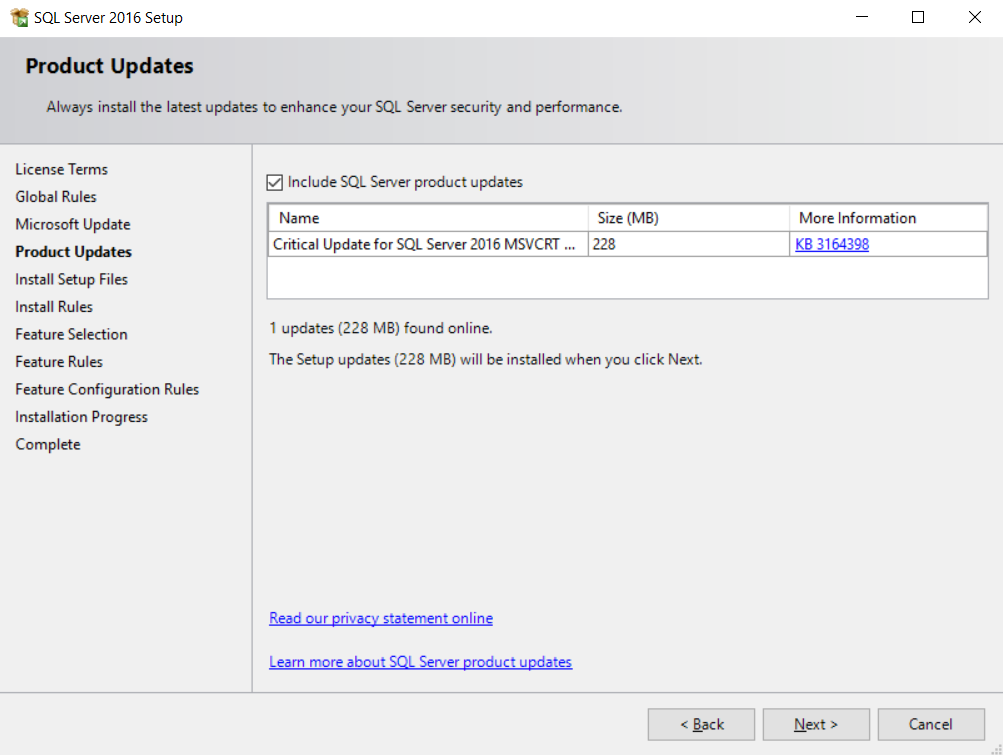
Click on the first option. A new installer will come up. It should look like this:

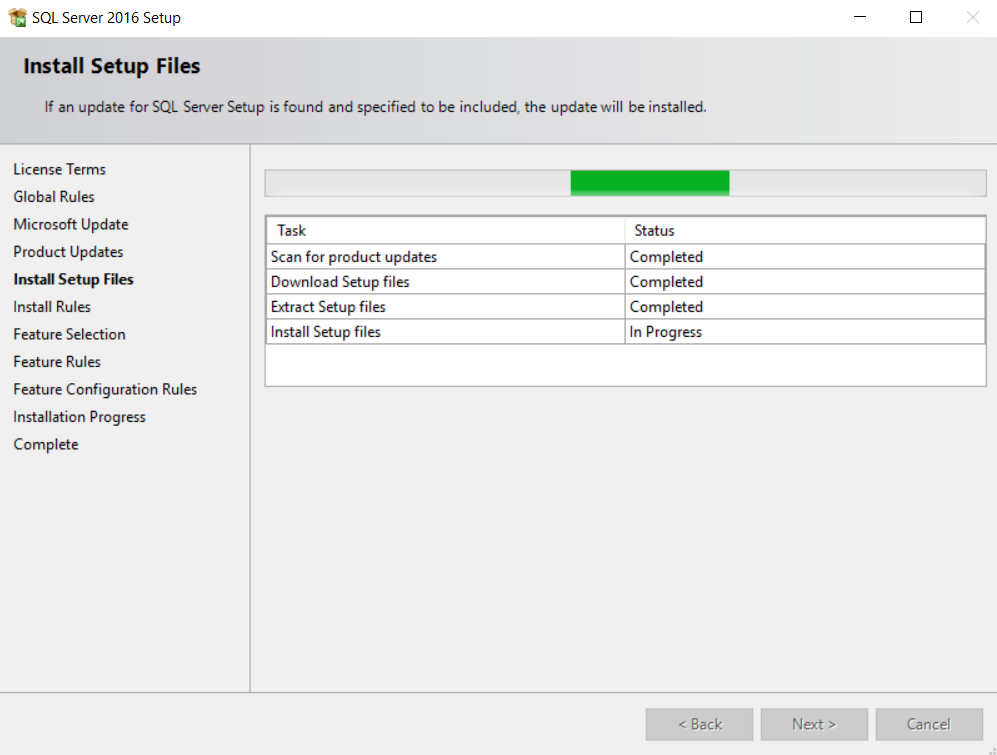


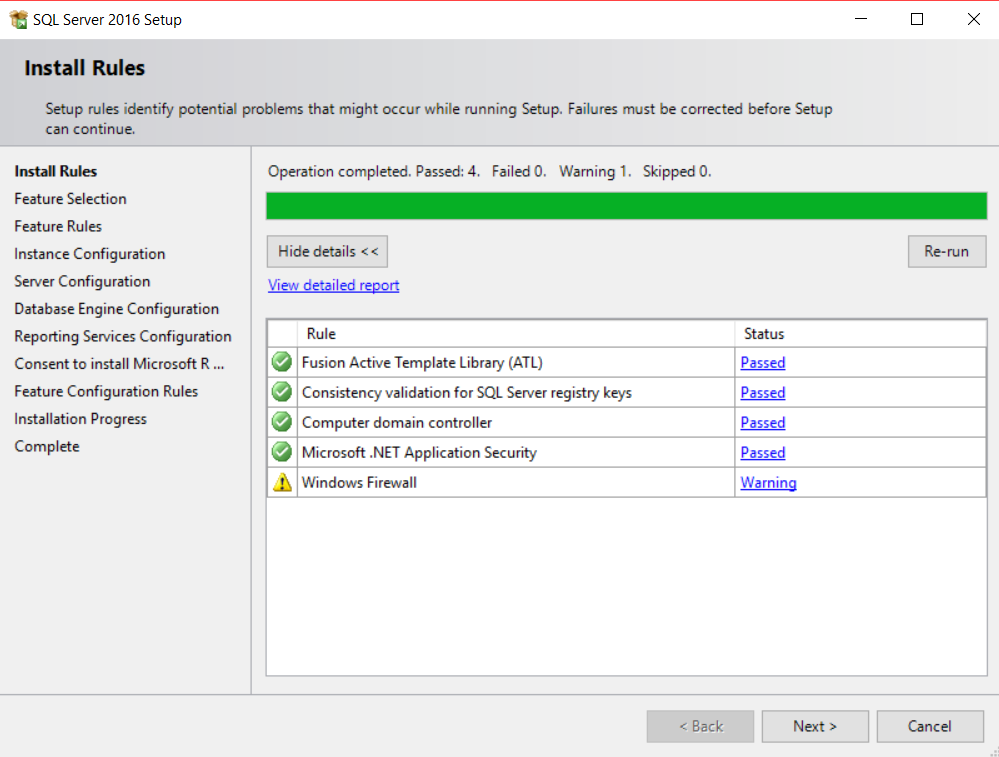
Click [**Next**].



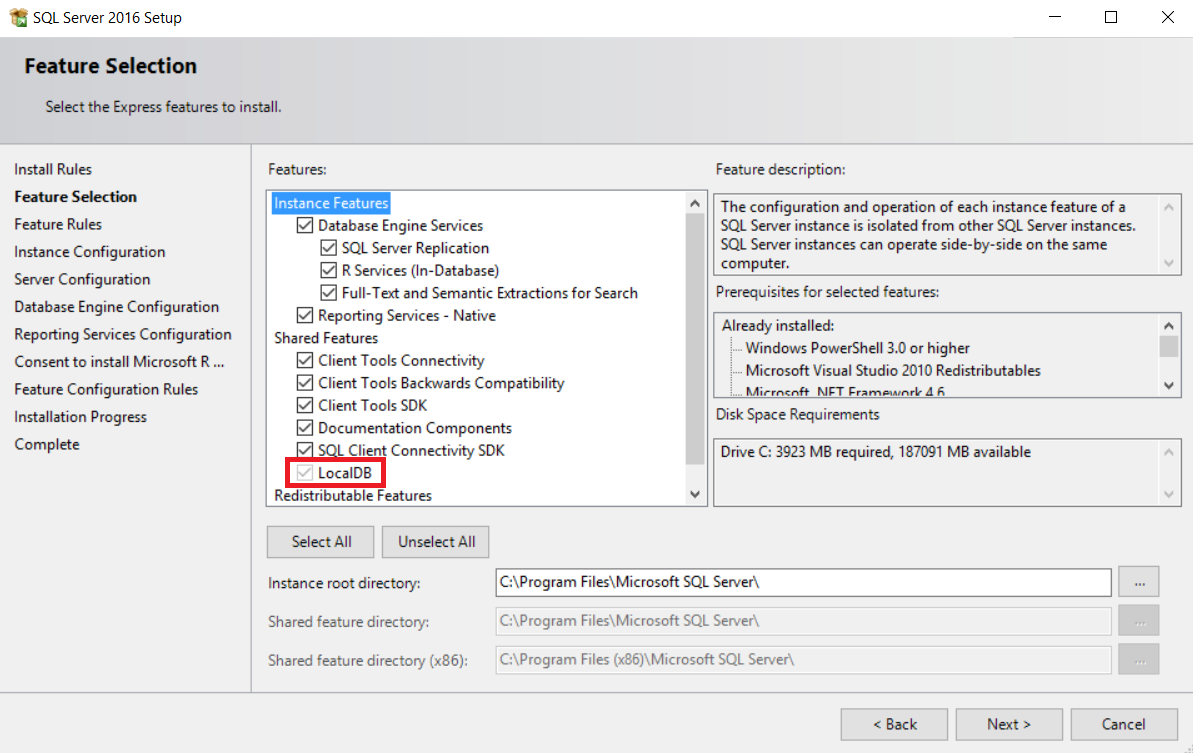
Click [**Next**] again.



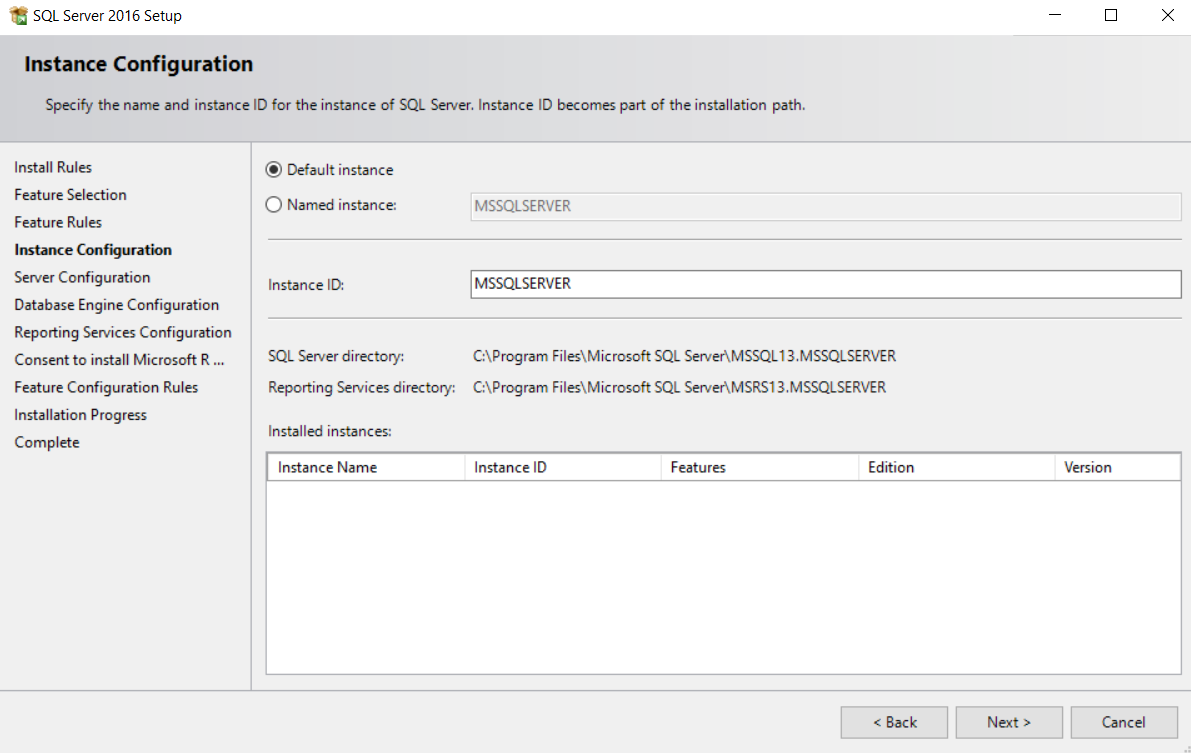




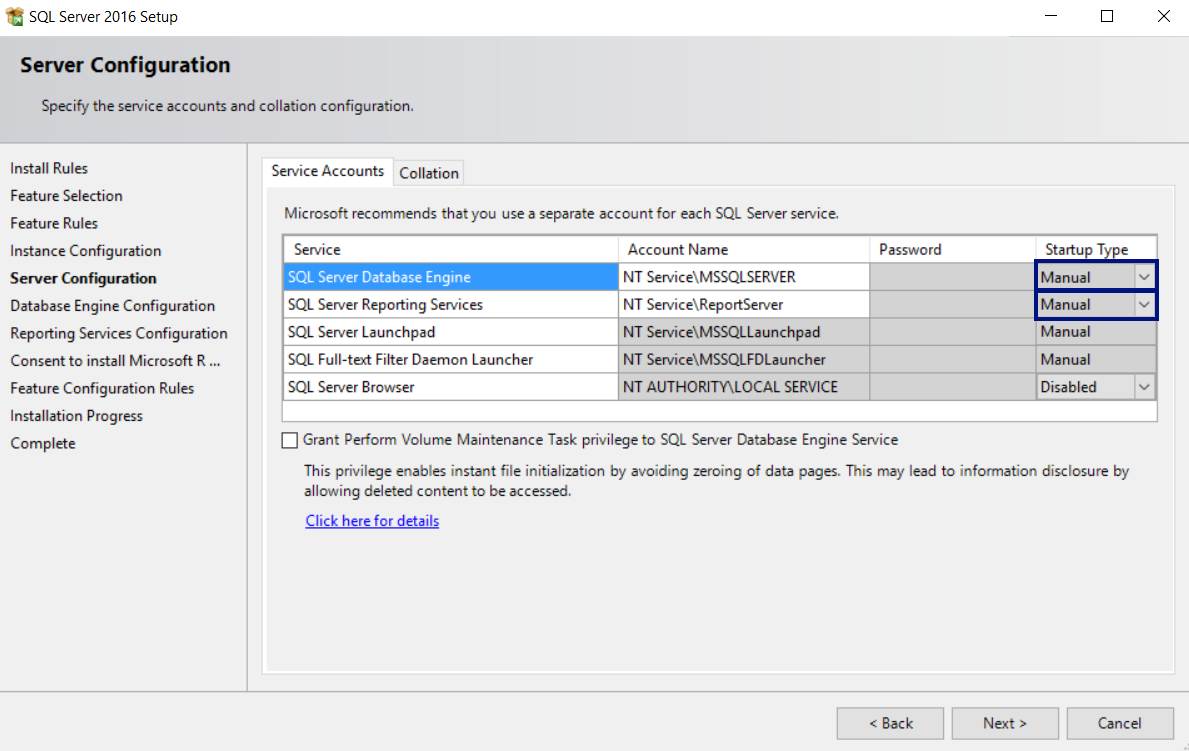
Check the **LocalDB**. It is the only required component for our training course.

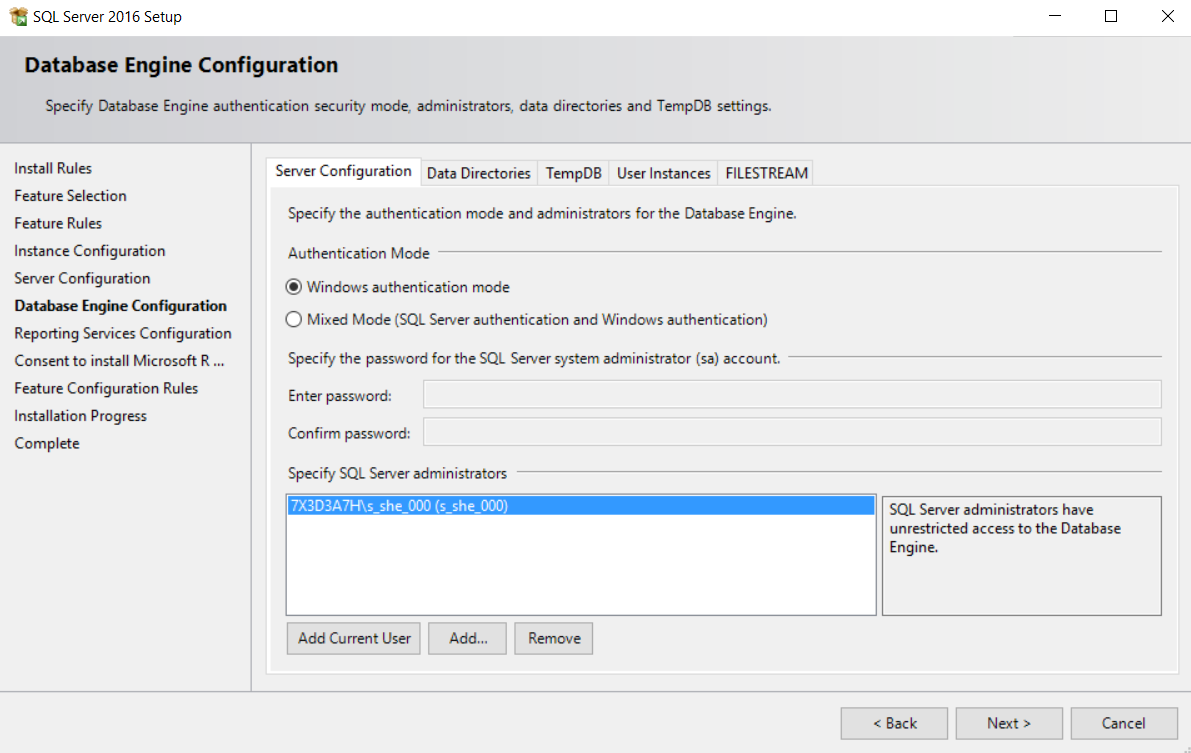


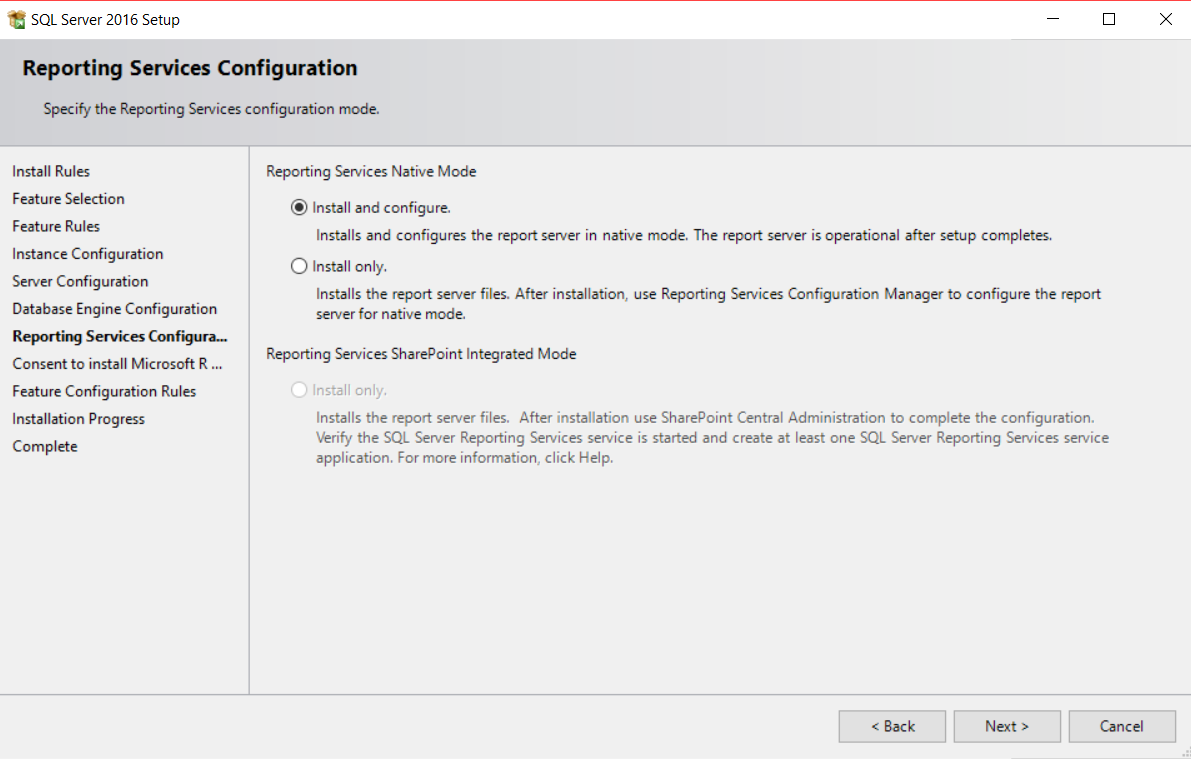
The next few screens will come when you install the full SQL Server Express version (not only the LocalDB):

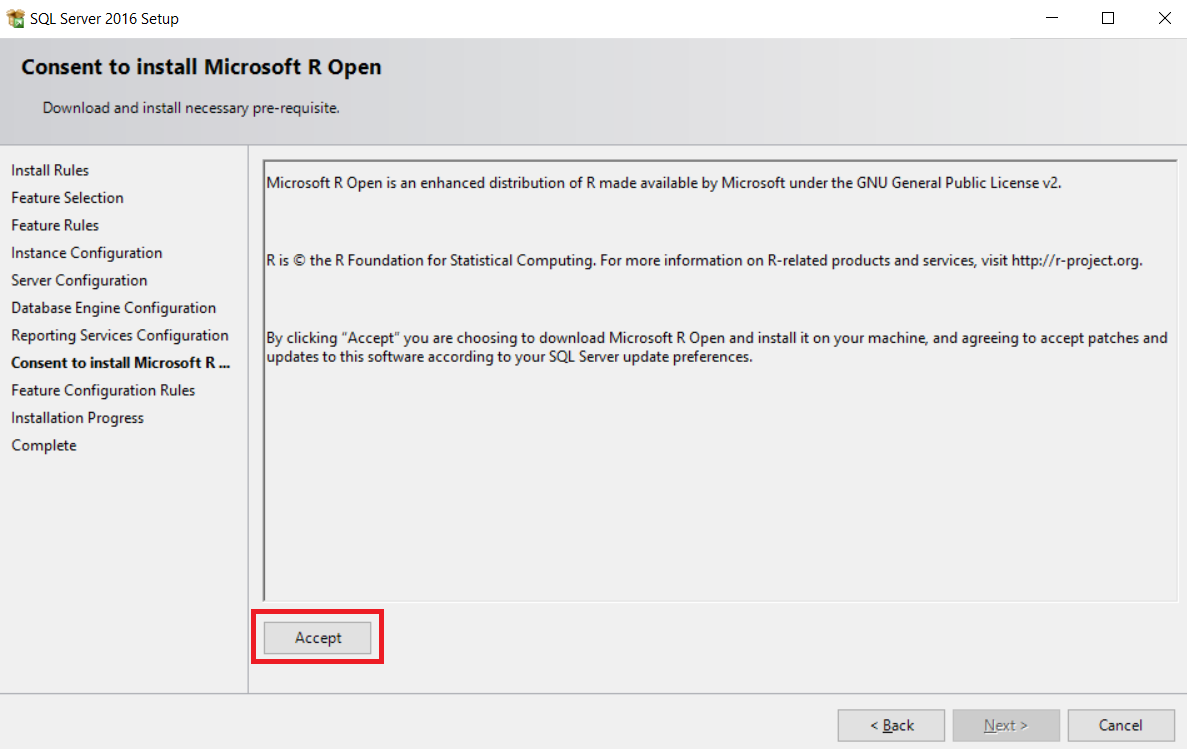


To save memory and Windows startup time, choose **[Manual] Startup Type** for all components.









After clicking on **Next** the installation will start.

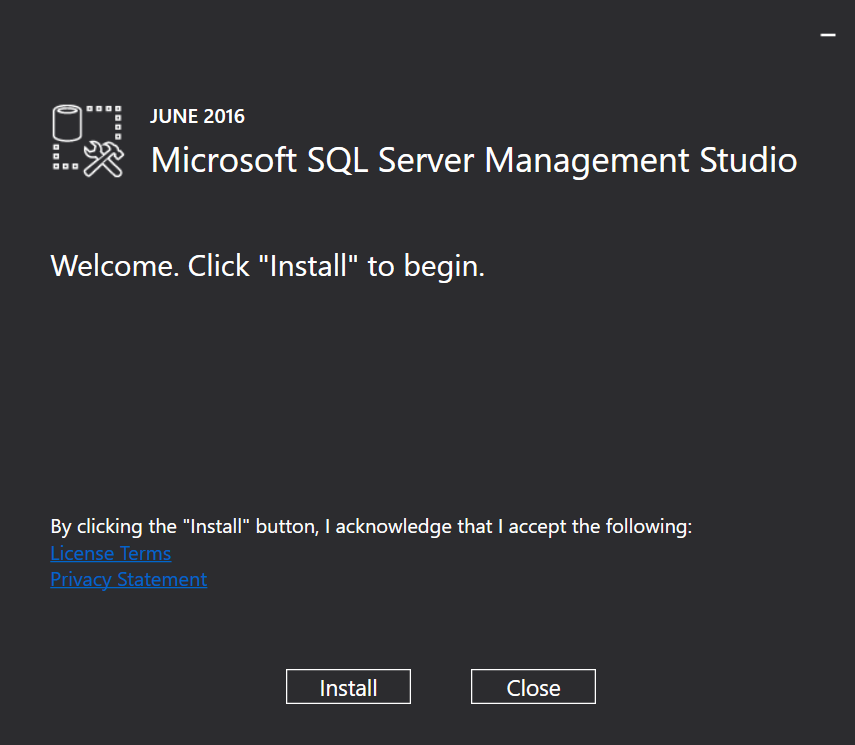
**Note**: During the installation, you may receive error, which you can simply ignore. It will result in installation fail for the reporting services, but don’t worry, we don’t need them.

That is all, now you have **installed MS SQL Server 2016**.

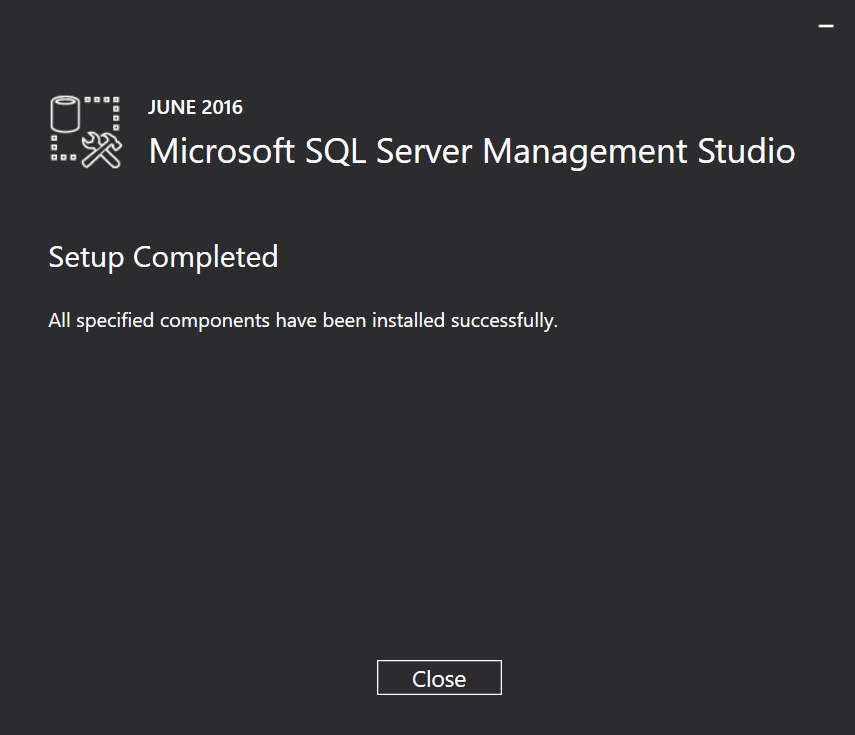
## Download and Install SQL Server Management Studio (SSMS)

Download **Microsoft SQL Server Management Studio** from here: <https://msdn.microsoft.com/en-us/library/mt238488.aspx>. Don’t use “**SSMS July 2016**” version. It is **broken** and does not work!

The installation will start from this window:



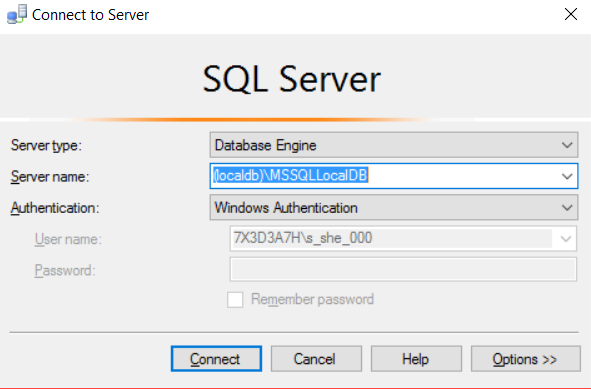
And the end result should be this:



Now we should be ready to start SSMS for the first time.

## Start Management Studio (SSMS)

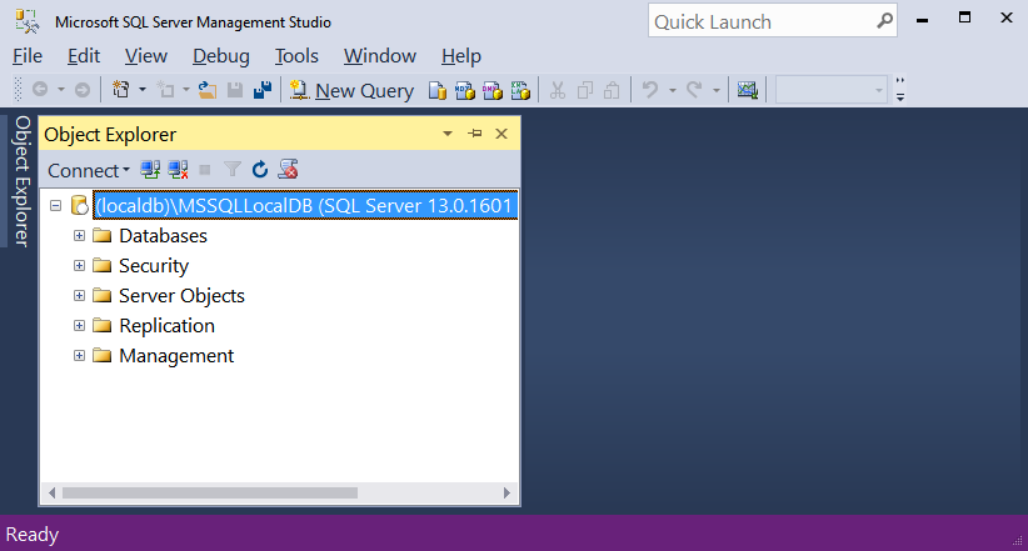
The first time you start Microsoft **SQL Server Management Studio**, you should see this screen:



Use the following server name to connect to **MSSQLLocalDB** instance in the **LocalDB**:

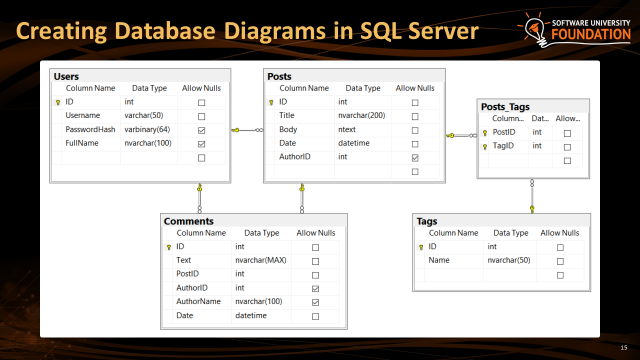
|  |
| --- |
| **(localdb)\MSSQLLocalDB** |

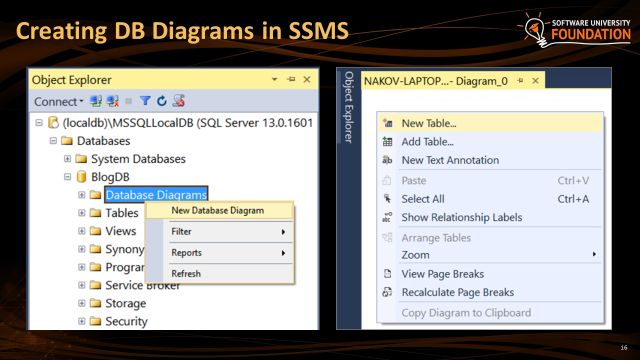
After successful connection to the LocalDB, you should see this screen:



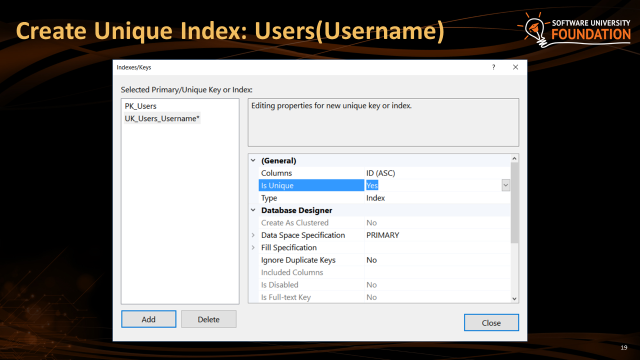
# Part II: Design a Database Schema with SSMS

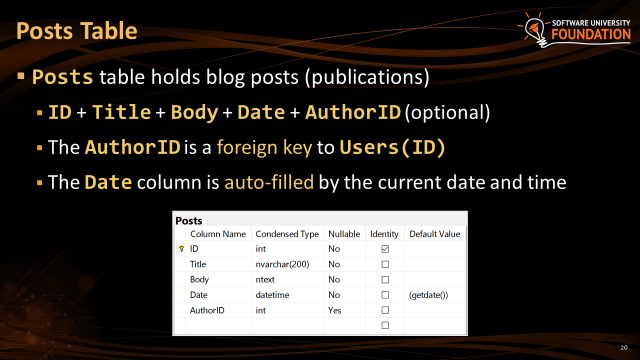
The goal of this part is to **design a database schema** with SQL Server Management Studio.

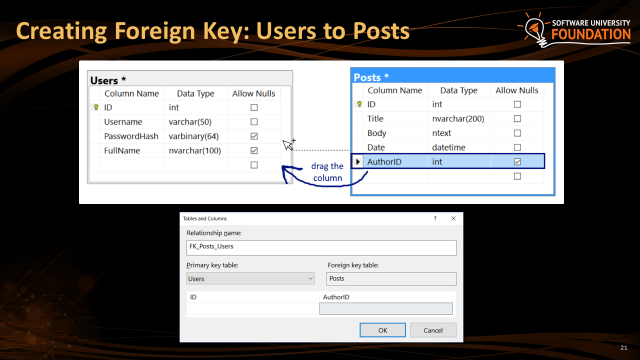


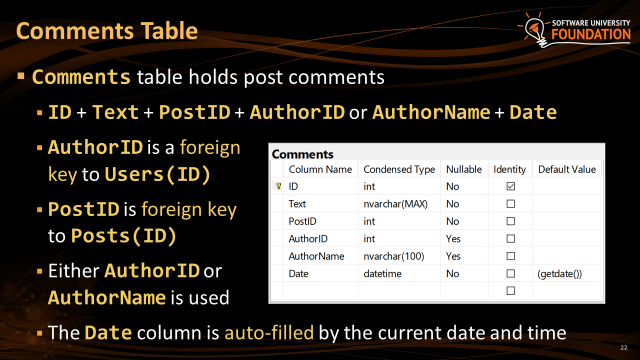


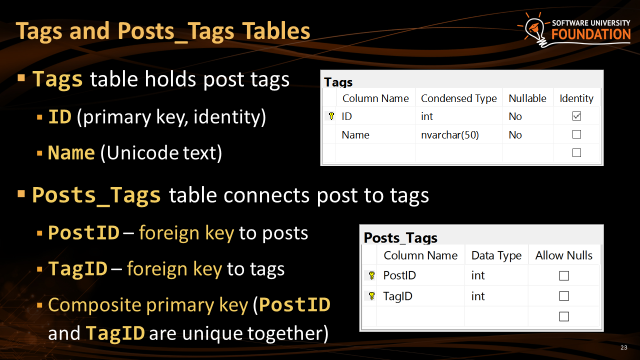


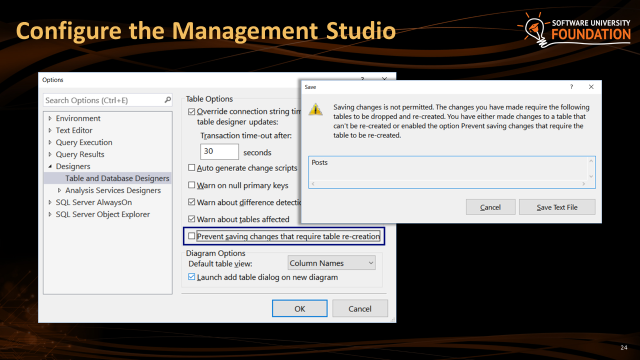


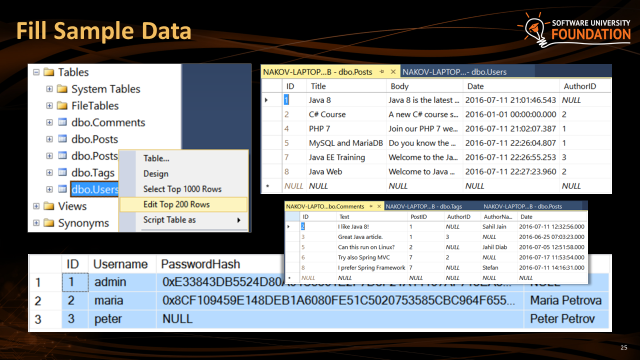


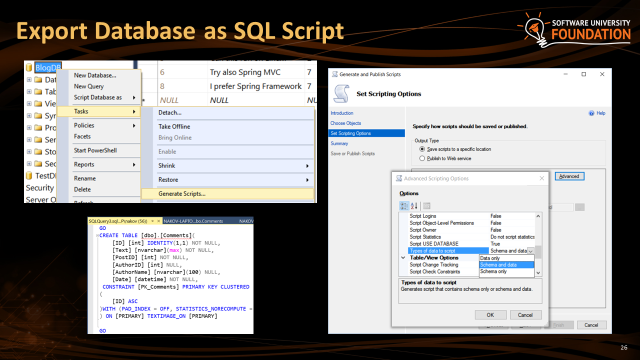












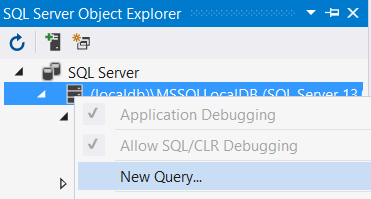
# Part II: Manipulating the SQL Database with Queries

In this part of the exercise lab you shall **write SQL commands** on a database given you as SQL script.

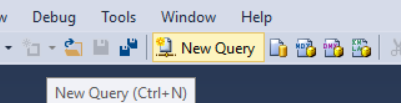
## Create Database by SQL Script

First, you should create (import) the working database called “BlogDB” and insert some sample data in it. All this is done by executing a SQL script: BlogDB-Create-Database-Script.sql.

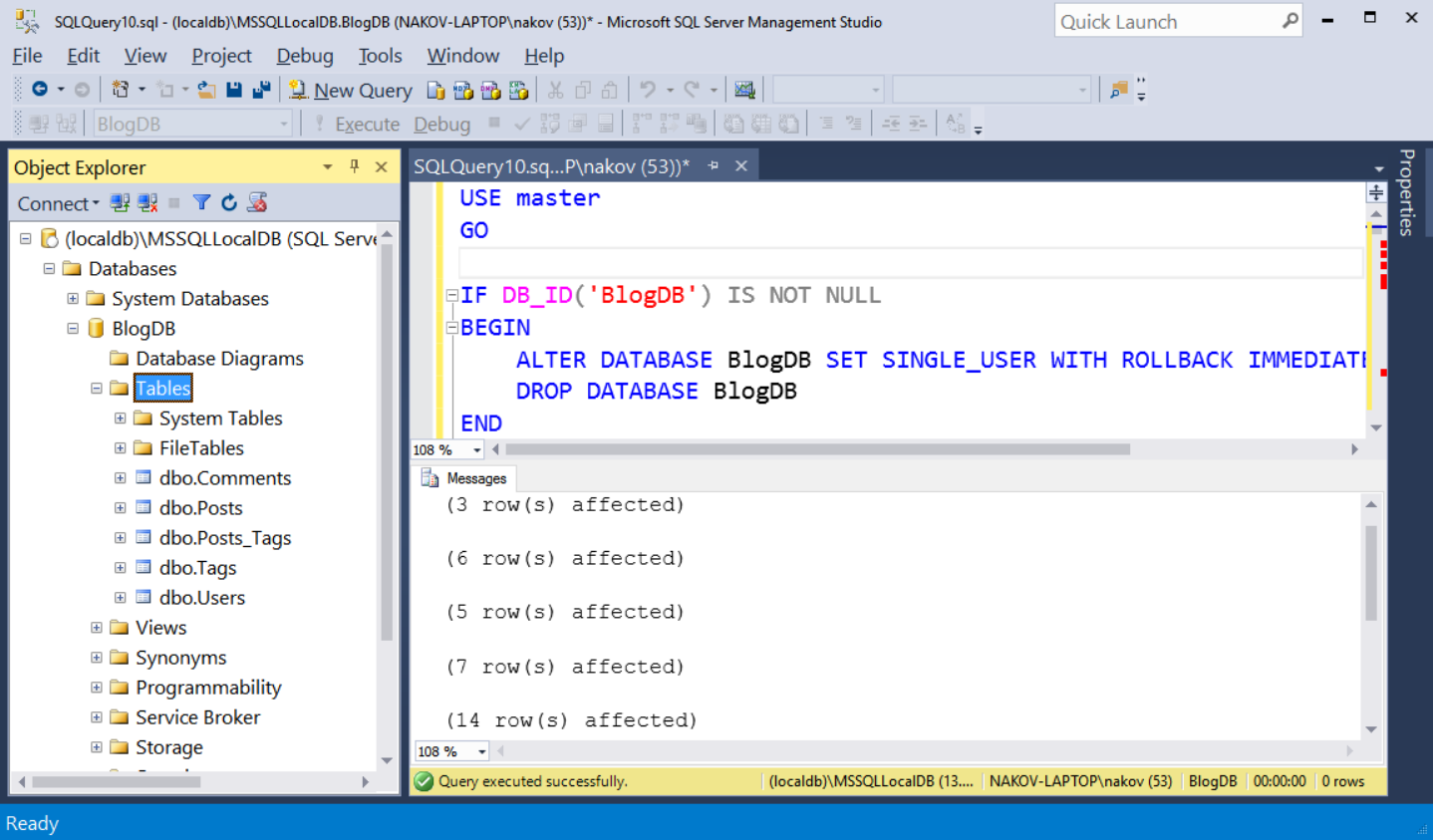
Open a new **query window** in respectively Visual Studio or SSMS. Here is the VS example:



And here is the SSMS location:



After the window opens, you simply **copy the SQL script code** from the SQL file and **paste it** to the SQL query window. Execute the query and you are ready to start. Ensure the query execution gives **no errors**!



Now that we have our SQL Database set up, we can start playing with the data in it. Working with that data requires the use of various SQL Queries. Here are some of them you must implement.

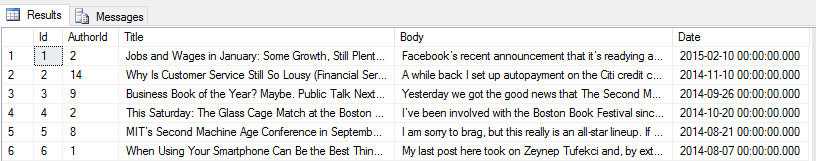
## SELECT Commands

You should already be familiar with the SELECT Command. The SELECT Command **selects** specified **columns** from the **rows** of a specified **table**. In our case we just type “\*”, which means – Select everything (all columns).

### List Posts

Let us list all data rows from the database’s Posts table.   
Open a New Query, and enter the following: ‘SELECT \* FROM [Blog].[dbo].[Posts]’.  
And execute the Query.

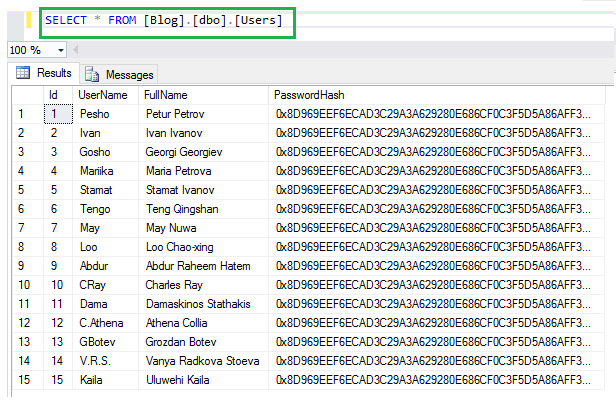
You should see this as a result:



### List Users

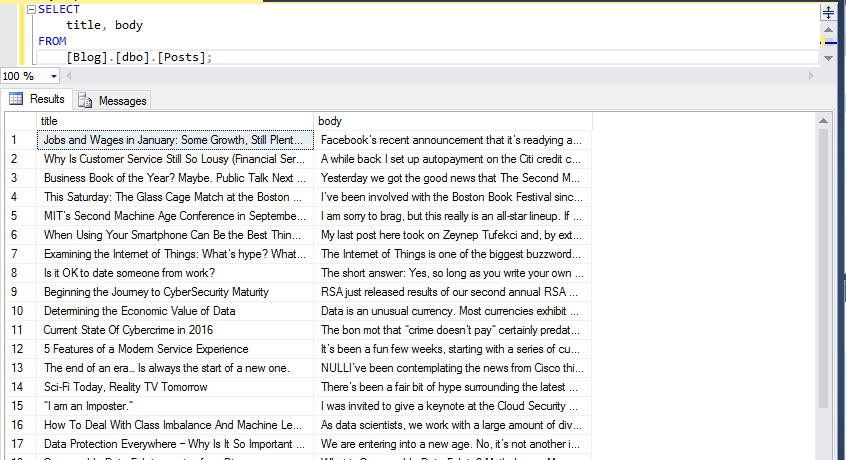
The “SELECT \* from …” will be our main means of testing whether our other Queries work, because it provides us with full data.

Let us get all users now:



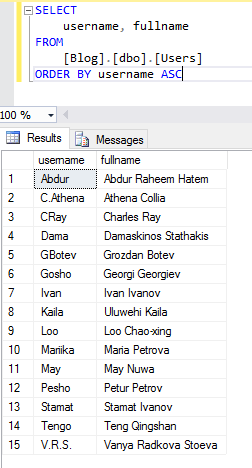
### List Columns from Posts

Let’s do something harder. We will get only the title and body from every post, ignoring the other fields:



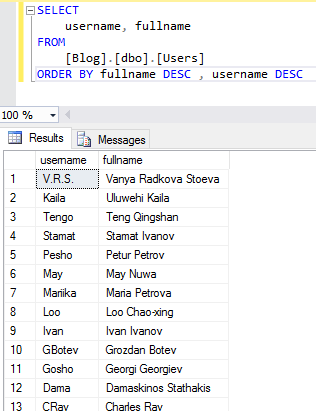
### Order Data

Now we will order the data in ascending order. Let’s take all usernames and full names of our users.



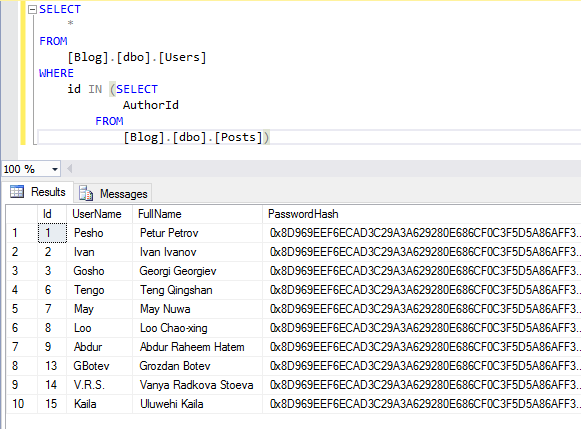
### Order by Two Columns

What if we have same username for 2 users? We should order by 2 parameters, but this time we will do it in descending order:



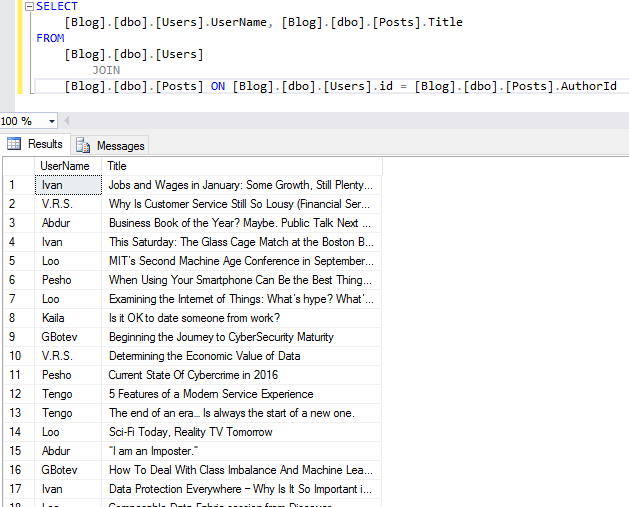
### Nested Select

Let’s imagine that we’ve had many users, but not all of them have created blog posts. How can we get only the users that have created a blog post? Just like that:



### Joins Users with Posts

Something more interesting now – let’s get the usernames with the post names for each user. The query should look like this:



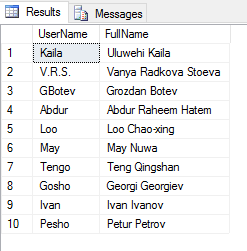
### Joins Users with Posts with Filtering

Let’s get the creator of post with id=4 now:



### Posts Authors

Let’s do one last SELECT command. It should get all of the post authors and order them by id in descending order. We will need only the username and full name. But this time, you will be given only the expected result.



How did we do that? To explain that I will split the query in 3 parts. The first part will be the nested SELECT. This query will return collection (array) with every ‘user id’ that has at least one post in the posts table. Let’s examine the first SELECT now. It takes every username and full name of the users, which ‘id’ is contained in the collection that our nested query returned. In other words, if we have post creators with ‘id’ – 3, 8 and 53 the query will do contains for every user id that we have (example -> collection.contains(currentId)). After we do that, the final part is the sorting by ‘id’ in descending order.

## INSERT Command

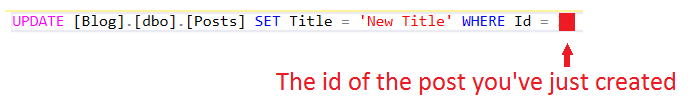
**INSERT** commands are used to enter data to our table. Let’s try them!



This query should create new post in our table. Create **SELECT** command to see if it worked.

## UPDATE Command

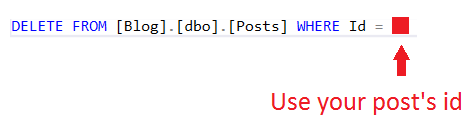
**UPDATE** commands are used to change the data in our tables.



This query should update our newly created post. Create **SELECT** command to see if it worked.

## DELETE Command

**DELETE** commands are used to delete data from our tables.



This query should delete our newly created post. Create **SELECT** command to see if it worked.

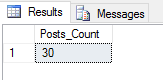
## COUNT Command

The COUNT() function in SQL is used to find the number of rows that match a given criteria. For example:  
COUNT(\*), would return the count of all rows on the table, or, in other words – all the stored records. Let us perform several simple actions using this command.

First, let us count all the records in the Posts table and print their count as “Posts\_Count”.

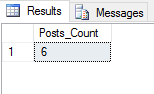


The criteria is “\*”, which means everything. So this will count everything from the Posts table and show it in the output as Posts\_Count. The result should be the following:



…assuming you have not changed the Posts table from its initial state.

We’ve seen how the COUNT command works, now let us count all the posts which have author id = 2. Write a query that will do that for you. The result should be the following:



There are 6 posts, which have author id – 2.

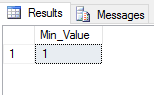
## MIN Command

The MIN() function returns the smallest value of a selected column in a given table.

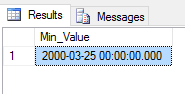
Example:



This query will select the minimal value of the AuthorId column in the Posts table, and show it in the output as Min\_Value. And the result should be the following:



Now, make a query that extracts the post which is the oldest in the table – in other words, the one that has the earliest date. The result should be the following:



## MAX Command

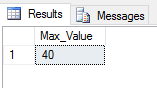
Opposite to the MIN() function, the MAX() function returns the maximum value of a selected column in a given table.

Example:



This will return the maximal value in the AuthorId column of the Posts table and show it in the output as Max\_Value.

Now, let us create a query that extracts from the Tags table the highest value of the id column. This one goes to you. If you do everything correctly the result should be the following:

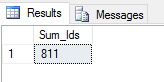


## SUM Command

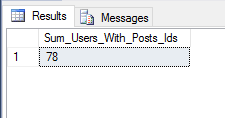
The SUM() function returns the sum of all the values in a given column with a **numeric** value type. We do not have any numerical value we could logically sum, but id’s should work just fine. Let’s sum all the id’s of the Tags table.



The result should be the following:



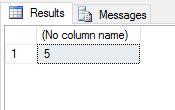
A bit more complicated task – Let us sum the id’s of all the users that have posted something. This one goes to you. If you perform the task correctly, the result should be the following:



# Part III: Playing with the SQL Database

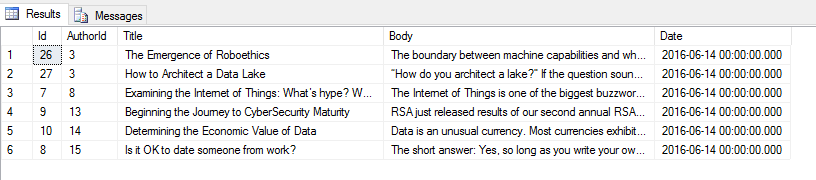
## Count of Users which Have a Post in a Specified Date

Write a query, which returns the count of all the users, which have posted a post on the 14th of June, 2016, and shows it in the output as Users\_Count.   
The result should be the following:



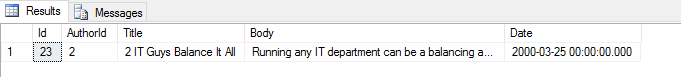
## Extract and Order All the Posts in a Specified Date

Write a query, which extracts all the posts from the 14th of June, 2016, and orders them, in ascending, by AuthorId.



## Find the Earliest Post from a Specified User

Write a query that finds the earliest post from user with id – 2.



## Play with Comments

Write a query, which extracts all comments on all posts and orders them by Author Name – Ascending Order and then by Id – Descending Order.

You should see the following results:

