Revision 06/2020

Siemens S7-1200/1500

TIA Portal (V13, V14, V15, V16)

MS SQL Functionality

Implementation into user project

PDSql Library v1.3.8

User guide

www.plcdirectsql.com

Contents

1 Introduction

PDSql Library allows you to connect your PLC Siemens S7-1200 or S7-1500 system directly to **Microsoft SQL Database**. With this library you are able to read and write data from/to SQL Database. As the communication between the PLC and the SQL Server takes place directly you don't need any PC as a broker in the communication between the PLC and SQL server.

1.1 Hardware requirements

PDSql library was built for **PLC Siemens S7-1200** (**fw 4.2**+) and **S7-1500 (fw 1.8**+).

1.2 Software requirements

Main requirements for this library is **TIA Portal V13+** and **Microsoft SQL Server (2000 or higher)**.

TIA Portal V13+ is available only with paid licence. Microsoft SQL Server 2017 is available as free Express Edition version with some limitations.

You can download it here: https://www.microsoft.com/en-us/sql-server/sql-server-editions-express

1.3 Supported SQL functions

This library allows you to execute all the basic SQL commands like

- SELECT
- INSERT
- DELETE
- UPDATE
- Execute Stored PROCEDURE

However, in addition to this basic SQL commands, **other commands can be also executed** depending on string content in query input. However, with some commands of this type, the query may end up with an error.

1.4 Supported SQL data types

Supported are only datatypes listed in the Table 1-1. Any other datatype that is not listed in Table 1.1 will either convert its value to a string or cause the query to quit with an error.

Table 1-1: SQL data types

Datatype		S7-1200	S7-1500
SQL	PLC		
bit	Bool	•	•
nchar(n)	String([1])	•	•
char(n)	String([1])	•	•
nvarchar(n)	String([1])	•	•
varchar(n)	String([1])	•	•
tinyint	DInt/LInt	• [2]	• [3]
smallint	DInt/LInt	• [2]	• [3]
int	DInt/LInt	• [2]	• [3]
bigint	LInt		•
float	LReal	•	•
real	LReal	•	•
datetime	DTL	•	•
smalldatetime	DTL	•	•
date	DTL	•	•
numeric	LReal	• [4]	• [5]
decimal	LReal	• [4]	• [5]
money	LReal		•
smallmoney	LReal	•	•

[1] The maximum length of string is defined in ***tSqlTable.MaxStringLength*** variable. By default the value is 50.
* If the string length of the SQL data type is greater than the string length in the PLC, so the rest of the text will be cut off.

[2] PLC data type DInt

[3] PLC data type LInt

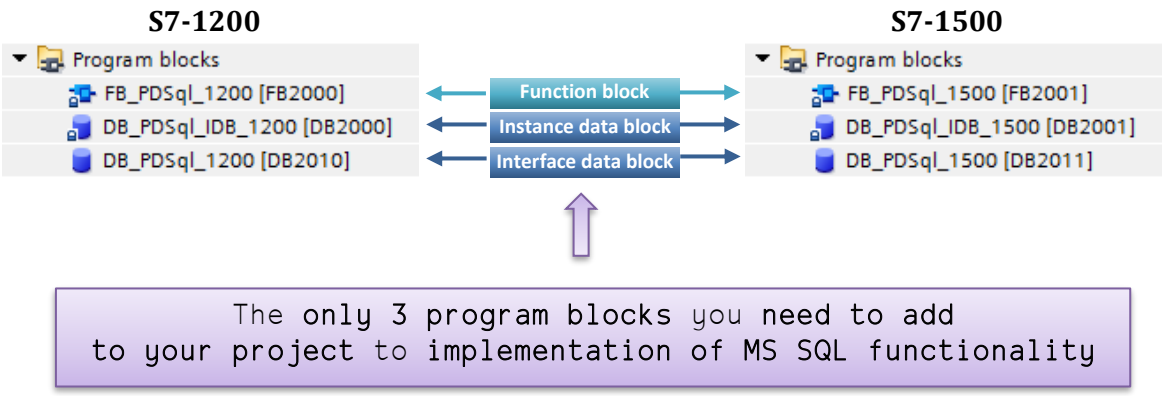
[4] Supported 32-bit value

[5] Supported 64-bit value

1.5 Program blocks

1.5.1 Function block FB_PDSql

The whole concept of this library was built on the ease of implementation into user projects. Therefore, all functions for communication with **MS SQL Server** are contained in only one function block **FB_PDSql**. This block is also associated with one instance of the data block **DB_PDSql_IDB**. The last one is the data block **DB_PDSql** which provides all parameters and the interface between the user application and the **PDSql library**. It contains the resulting SQL table obtained after SELECT command. These blocks are supplied in **2 versions** for **S7-1200** and **S7-1500**.



The only 3 program blocks you need to add to your project to implementation of MS SQL functionality

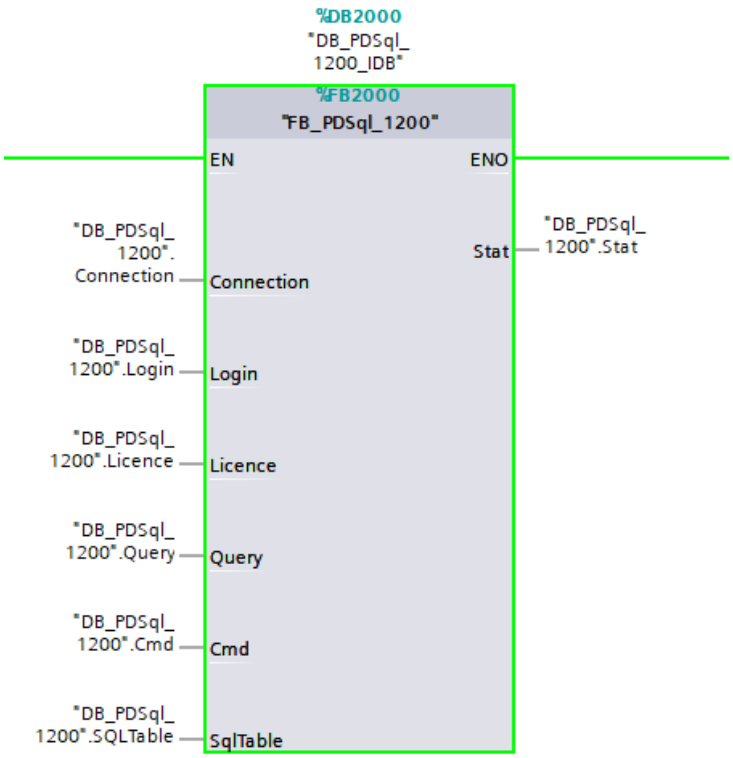


Table 1-2 : **FB_PDSql** interface

Type	Name	Data type	Comment
Input	Connection	tConnection	SQL server connection parameters
	Login	tLogin	Login parameters
	Licence	tLicence	Runtime license key
	Query	tQuery	SQL query string
Output	Stat	tStat	Status bits, messages and errors
InOut	Cmd	tCmd	Commands **Connect**, **Disconnect**, **ExecuteQuery**
	SqlTable	tSqlTable (1200/1500)	SQL parameters and table contents as a result of the SELECT statement

Recommended steps for executing SQL commands from a user application

1. Create a string of SQL statements and write it to ***Query***
2. Set the ***Cmd.ExecuteQuery*** signal to **TRUE**
3. Wait for signal ***Stat.ExecutedOK*** or ***Stat.Error***
4. If the ***Stat.Error*** signal is **TRUE**, repeat the action from step 1.
   If the ***Stat.ExecutedOK*** signal is **TRUE,** application can continue to prepare another SQL statement and repeat from step 1.

The user application can fully control the ***Cmd.Connect*** or ***Cmd.Disconnect*** signals, but it is sufficient to control only ***Cmd.ExecuteQuery*** signal which ensure the connection automatically if needed.

1.5.2 Data block DB_PDSql

Data block **DB_PDSql** is composed of all the data types described in chapter 1.6. All communication between the user application and the SQL server is done through this data block.

S7-1200

DB_PDSql_1200

	Name	Data type
1	Static	
2	Connection	"tConnection"
3	Licence	"tLicence"
4	Cmd	"tCmd"
5	Stat	"tStat"
6	Login	"tLogin"
7	Query	"tQuery"
8	SQLTable	"tSqlTable_1200"

S7-1500

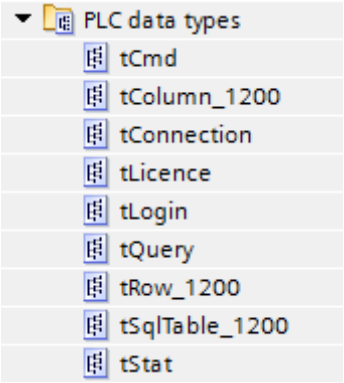
DB_PDSql_1500

	Name	Data type
1	Static	
2	Connection	"tConnection"
3	Licence	"tLicence"
4	Cmd	"tCmd"
5	Stat	"tStat"
6	Login	"tLogin"
7	Query	"tQuery"
8	SQLTable	"tSqlTable_1500"

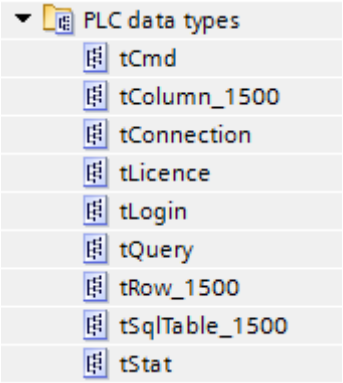
1.6 PLC data types

Function block **FB_PDSql** and data block **DB_PDSql** uses several custom PLC data types to communicate with user application. The following is a list and description of these data types used in the library for **S7-1200** and **S7-1500**.

S7-1200



S7-1500



1.6.1 tConnection

This data type provides path parameters to connect to SQL Server.

Name	Data type	Default value	Comment
IP1	USInt	192	Octet 1 of the SQL Server's IP address
IP2	USInt	168	Octet 2 of the SQL Server's IP address
IP3	USInt	1	Octet 3 of the SQL Server's IP address
IP4	USInt	1	Octet 4 of the SQL Server's IP address
Port	UDInt	1433	Remote port of the SQL Server. MS SQL default is 1433.
ConnID	CONN_OUC	1	Reference to this connection. The parameter uniquely identifies a connection within the PLC.
HW	HW_INTERFACE	64	Hardware ethernet interface of the PLC used to the communication. X1(64), X2(72)
Timeout	Time	T#2s	Maximum time allowed to wait for connect, disconnect and execute query command. After this time elapsed, the error status is triggered.

1.6.2 tCmd

This data type provides command triggers to the Function block.

Name	Data type	Default value	Comment
Connect [1]	Bool	False	Trigger to connect to SQL server
Disconnect [1]	Bool	False	Trigger to disconnect from SQL server
ExecuteQuery [1]	Bool	False	Trigger to execute SQL query. If the SQL server has not been connected before activating this signal, the connection process will start automatically and then the SQL command will be executed after a successful connection to the sql server. **So the user application control can use only this one trigger signal to connect and execute commands automatically**.

[1] The bit signal is active in TRUE and triggered on the rising edge. Function block always resets this bit.

1.6.3 tLogin

This data type provides login parameters to the SQL Server.

Name	Data type	Default value	Comment
HostName	String	''	Freely definable name (can be empty string)
UserName	String	''	The user name of SQL Server account
Password	String	''	The password of SQL Server account
Database	String	''	The name of database

1.6.4 tStat

This data type provides status signals from the Function block.

Name	Data type	Default value	Comment
Connected	Bool	False	TRUE when logged to SQL server successfully
Busy	Bool	False	TRUE during execution of SQL statement
ExecutedOK	Bool	False	TRUE when SQL statement was executed successfully
Error	Bool	False	TRUE when SQL statement ended with an error
Status	Word	W#16#0000	Info or error code
Message	String	''	Info or error message after the last SQL command executed

1.6.5 tLicense

This data type provides runtime license key.

Name	Data type	Default value	Comment
LicKey1	DWord	16#00	License Key part 1
LicKey2	DWord	16#00	License Key part 2
LicKey3	DWord	16#00	License Key part 3
LicKey4	DWord	16#00	License Key part 4
LicKey5	DWord	16#00	License Key part 5

1.6.6 tQuery

This data type provides SQL query string.

Name	Data type	Default value	Comment
Query	Array[1..2] of String	''	Array of SQL Query

There is also PDSql Library **version** with an **extended** number of characters sent as a **Query** up to **10,160** characters (**40 Strings**). We will provide it for you to **download** for free on request.

1.6.7 tRow_1200

This data type provides row data for sql table result (S7-1200 version).

Name	Data type	Default value	Comment
Bool	Bool	False	Bool value
Null	Bool	False	NULL value
DInt	DInt	0	DInt value
LReal	LReal	0.0	LReal value
String	String[50]	''	String value
Datetime	DTL	DTL#1970-01-01-00:00:00	Datetime value

1.6.8 tRow_1500

This data type provides row data for sql table result (S7-1500 version).

Name	Data type	Default value	Comment
Bool	Bool	False	Bool value
Null	Bool	False	NULL value
LInt	LInt	0	LInt value
LReal	LReal	0.0	LReal value
String	String[50]	''	String value
Datetime	DTL	DTL#1970-01-01-00:00:00	Datetime value

1.6.9 tColumn_1200

This data type provides column data for sql table result (S7-1200 version).

Name	Data type	Default value	Comment
Name	String	''	Column's name
UserType	Byte	16#0	Column's user type
Type	Byte	16#0	Column's data type
LenSize	Byte	16#0	Column's length
TypeNumericPrecision	Byte	16#0	Precision factor in case of numeric or decimal data type
TypeNumericScale	Byte	16#0	Scale factor in case of numeric or decimal data type
Nullable	Bool	FALSE	Is Nullable
CaseSens	Bool	FALSE	Is Case Sensitive
Identity	Bool	FALSE	Is Identity type
Computed	Bool	FALSE	
FixedLenCLRType	Bool	FALSE	Internal use
UsUpdateable	Byte	16#0	0 – ReadOnly, 1 – Read/Write
Rows	Array[1..10] of "tRow_1200"		Array of rows

1.6.10 tColumn_1500

This data type provides column data for sql table result (S7-1500 version).

Name	Data type	Default value	Comment
Name	String	''	Column's name
UserType	Byte	16#0	Column's user type
Type	Byte	16#0	Column's data type
LenSize	Byte	16#0	Column's length
TypeNumericPrecision	Byte	16#0	Precision factor in case of numeric or decimal data type
TypeNumericScale	Byte	16#0	Scale factor in case of numeric or decimal data type
Nullable	Bool	FALSE	Is Nullable
CaseSens	Bool	FALSE	Is Case Sensitive
Identity	Bool	FALSE	Is Identity type
Computed	Bool	FALSE	
FixedLenCLRType	Bool	FALSE	Internal use
UsUpdateable	Byte	16#0	0 – ReadOnly, 1 – Read/Write
Rows	Array[1..10] of "tRow_1500"		Array of rows

1.6.11 tSqlTable_1200

This data type provides main parameters and sql table result (S7-1200 version).

Name	Data type	Default value	Comment
ColumnCount	UInt	0	The number of readed columns
RowCount	UInt	0	The number of readed rows
MaxColumns ⚠	UInt	10	Maximum allowed number of columns. The value must be less than or equal to ***tSqlTable.Columns*** array size
MaxRows ⚠	UInt	10	Maximum allowed number of rows. The value must be less than or equal to ***tColumn.Rows*** array size
MaxStringLength	USInt	50	Maximum number of characters in the string variable. Must be consistent with string length setting in ***tRow.String***
Columns	Array[1..10] of "tColumn_1200"		Array of columns

1.6.12 tSqlTable_1500

This data type provides main parameters and sql table result (S7-1500 version).

Name	Data type	Default value	Comment
ColumnCount	UInt	0	The number of readed columns
RowCount	UInt	0	The number of readed rows
MaxColumns ⚠	UInt	10	Maximum allowed number of columns. The value must be less than or equal to ***tSqlTable.Columns*** array size
MaxRows ⚠	UInt	10	Maximum allowed number of rows. The value must be less than or equal to ***tColumn.Rows*** array size
MaxStringLength	USInt	50	Maximum number of characters in the string variable. Must be consistent with string length setting in ***tRow.String***
Columns	Array[1..10] of "tColumn_1500"		Array of columns

⚠ The ***MaxColumns*** parameter must be less than equal to the array size of ***SqlTable.Columns*** and the ***MaxRows*** parameter must be less than equal to the array size of ***tColumn.Rows*** otherwise the PLC may go to STOP

1.7 Errors

Here is the list of errors written to the **tStat.Status** variable as output from the function block **FB_PDSql**. During the error code, a more detailed description of the error is written to the **tStat.Message.**

Table 1.3

Code	Error message	Tips
W#16#0000	No error	
W#16#8001	Cannot connect to the remote server	Check IP address and port
W#16#8002	Conection timeout	Check IP address and port
W#16#8003	Login failed	See details in tStat.Message
W#16#8004	Disconection timeout	Something is wrong with ethernet connection
W#16#8005	SQL Command execution timeout	SQL Server does not respond
W#16#8006	SQL Packet error	See details in tStat.Message
W#16#8007	SQL Command execution error	See details in tStat.Message
W#16#8080	Invalid license key	Check the license key

1.8 License / Demo

The license key is binded to the PLC serial number. After creating a license key with an online license tool on the webite **www.plcdirectsql.com**, these 5 license numbers need to be written to **LicKey1-5**. It is recommended to write them as the start value so that they remain written after the PLC restarts.

For the first **2 hours** after switching on the power supply of the PLC device, the library is **fully functional** even without a valid license key and works in **demo** mode.

Name	Data type	Start value
Static		
Connection	"tConnection"	
Licence	"tLicence"	
LicKey1	DWord	16#F292_D938
LicKey2	DWord	16#9A3F_E367
LicKey3	DWord	16#12DA_4E93
LicKey4	DWord	16#E388_F3A7
LicKey5	DWord	16#E07E_0D2D
Cmd	"tCmd"	
Stat	"tStat"	
Login	"tLogin"	
Query	"tQuery"	
SQLTable	"tSqlTable"	

2 Instalation

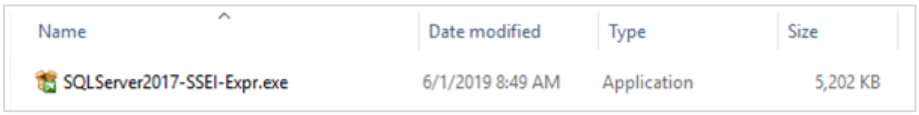
2.1 Microsoft SQL Server 2017

This chapter describes how to install Microsoft SQL Server 2017 Express Edition and set it up for communicating with the PDSql library. The link to download the instalation package is here -> https://www.microsoft.com/en-us/sql-server/sql-server-editions-express

1

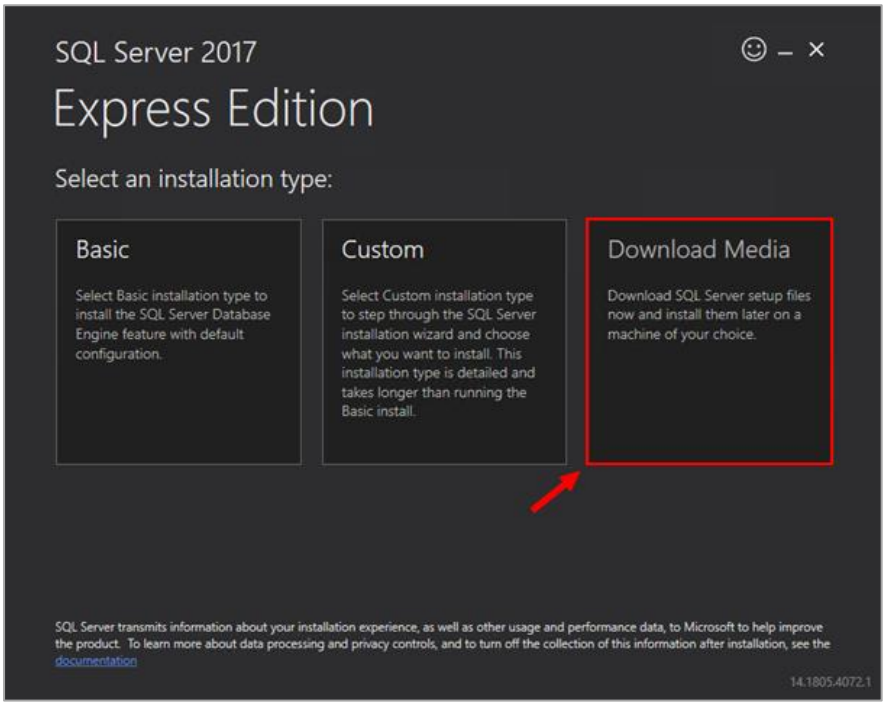
Click on **„Download now“**

2



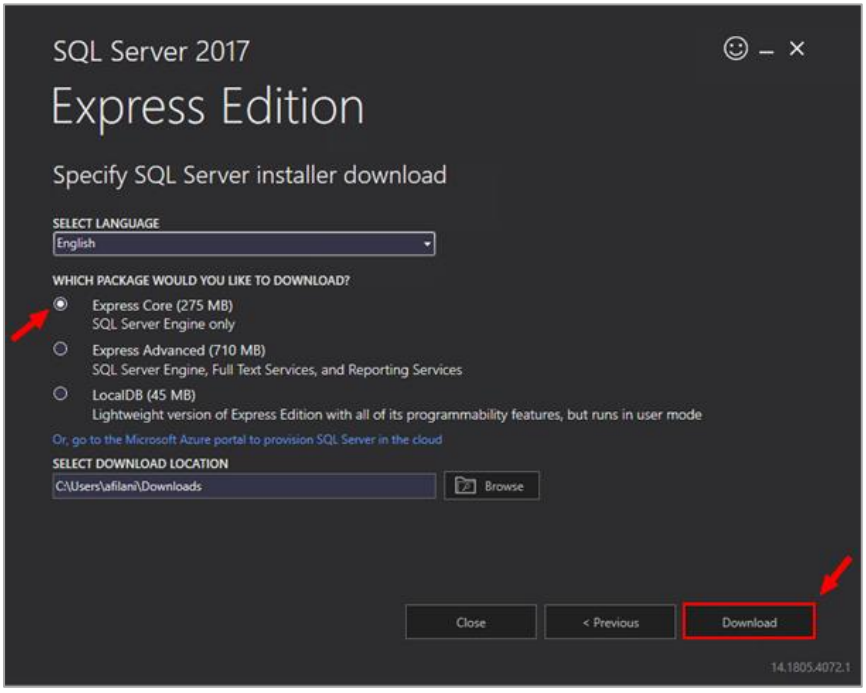
The **SQLServer2017-SSEI-Expr.exe** will be downloaded in your download directory. Run this file to continue.

3



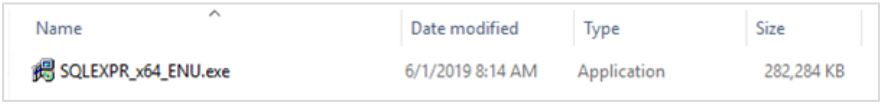
Click on **„Download Media“**

4



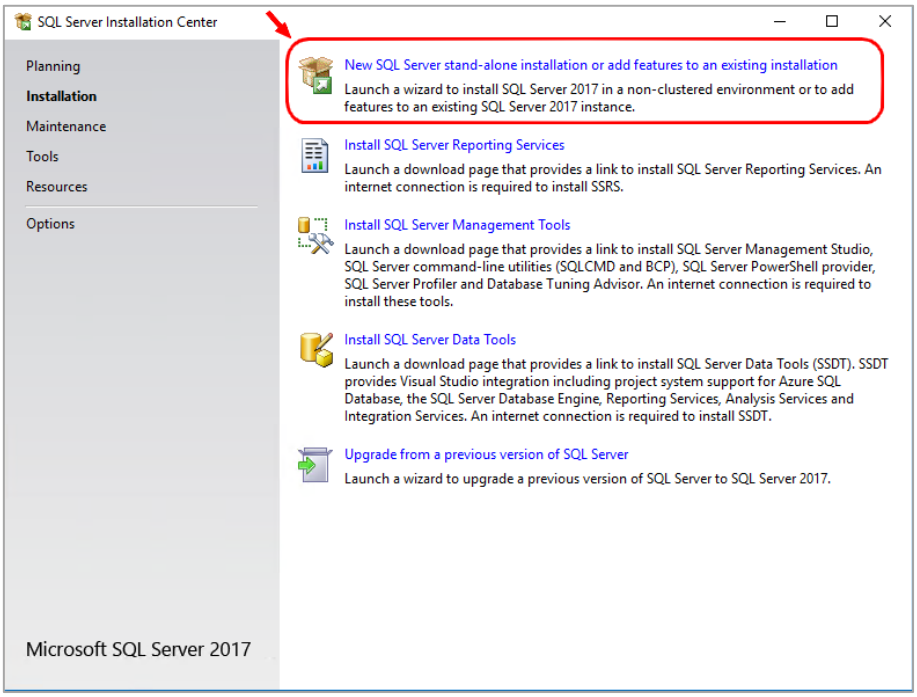
Select **„Express Core“** to install SQL Server Engine only and click on „**Download**“

5



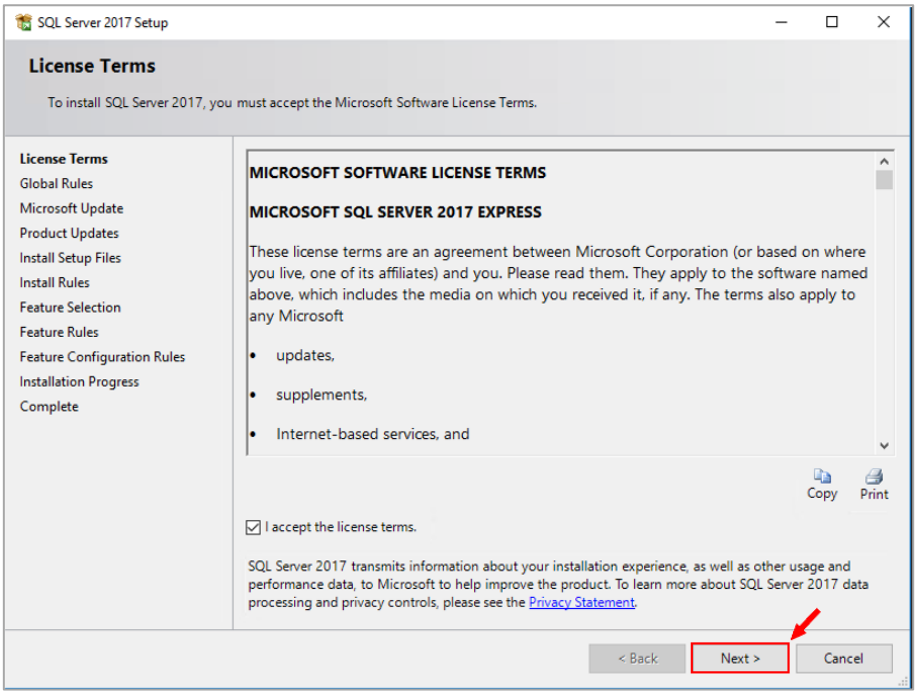
The **SQLEXPR_x64_ENU.exe** will be downloaded in your download directory. Run this file to continue.

6



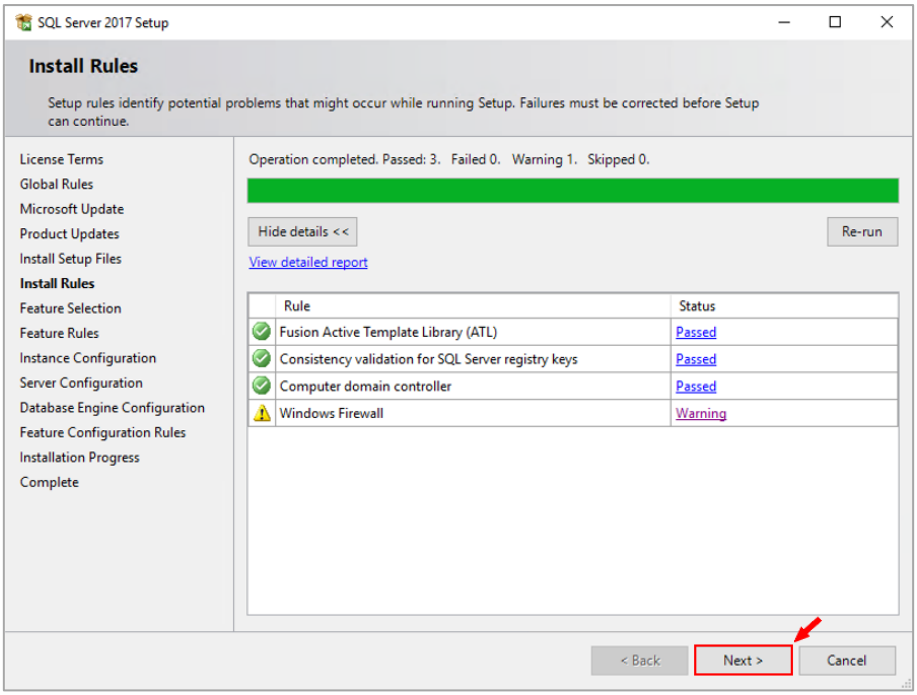
Click on the link **„New SQL Server stand-alone instalation ....“**

7



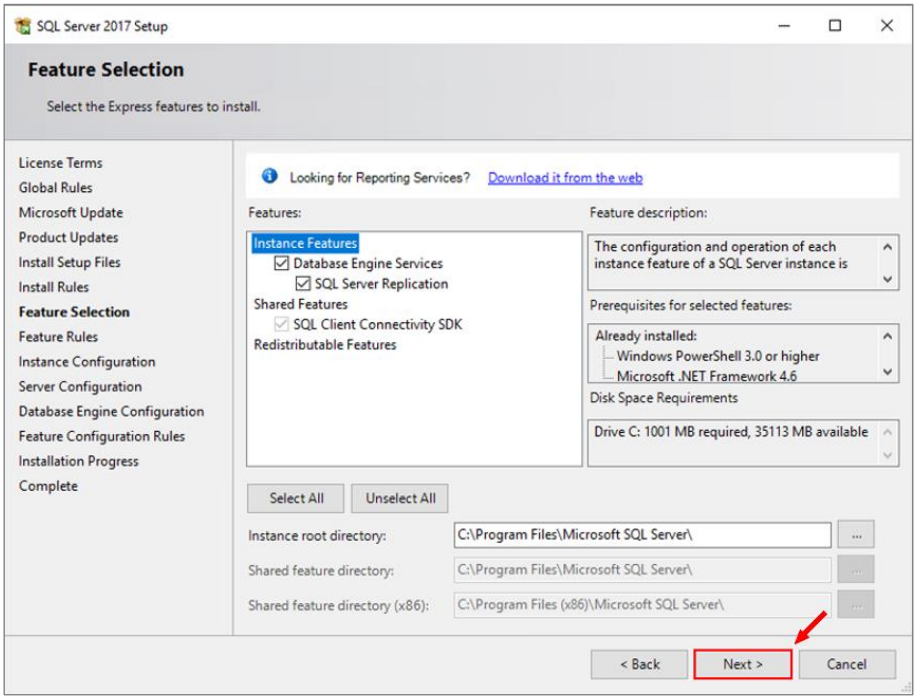
Accept the license terms and click on **„Next“**

8



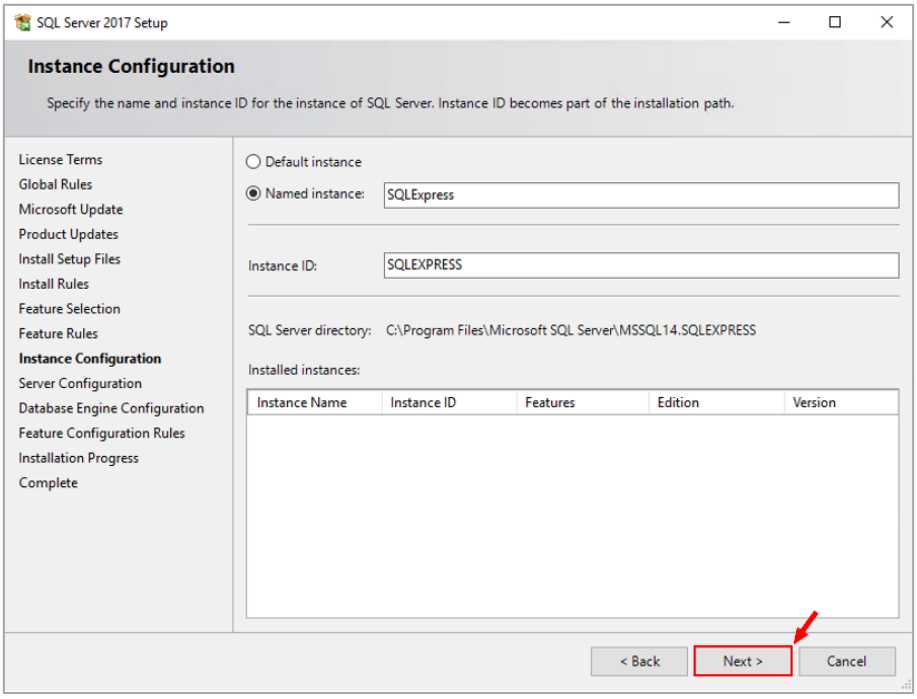
Click on **„Next“** (in this case, the Windows Firewall will be set later)

9



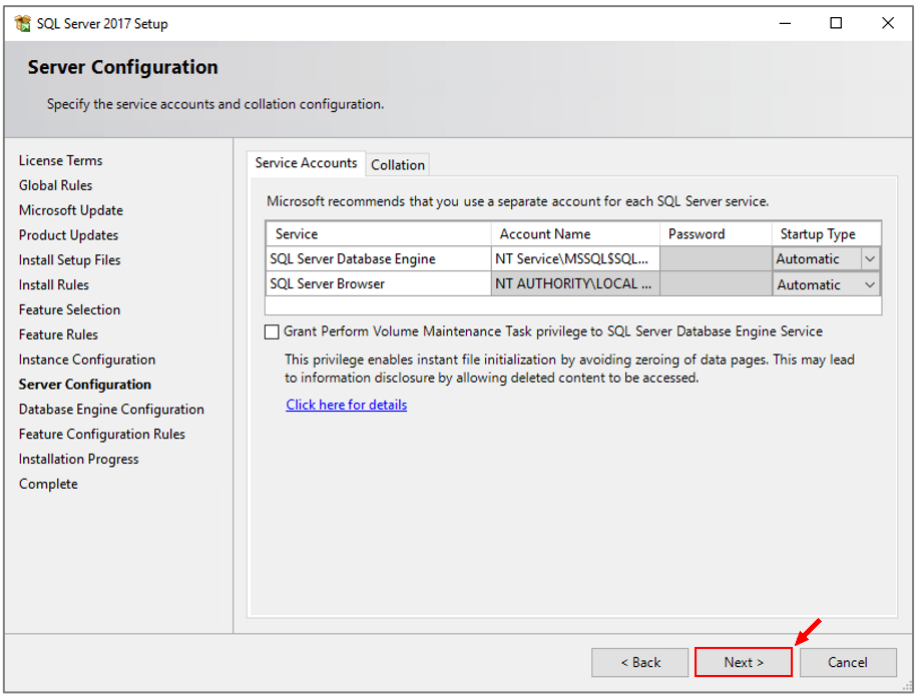
The minimum instance feature what we need to select is **„Database Engine Services“** but usually all of this selection is chosen. Click on **„Next“**

10



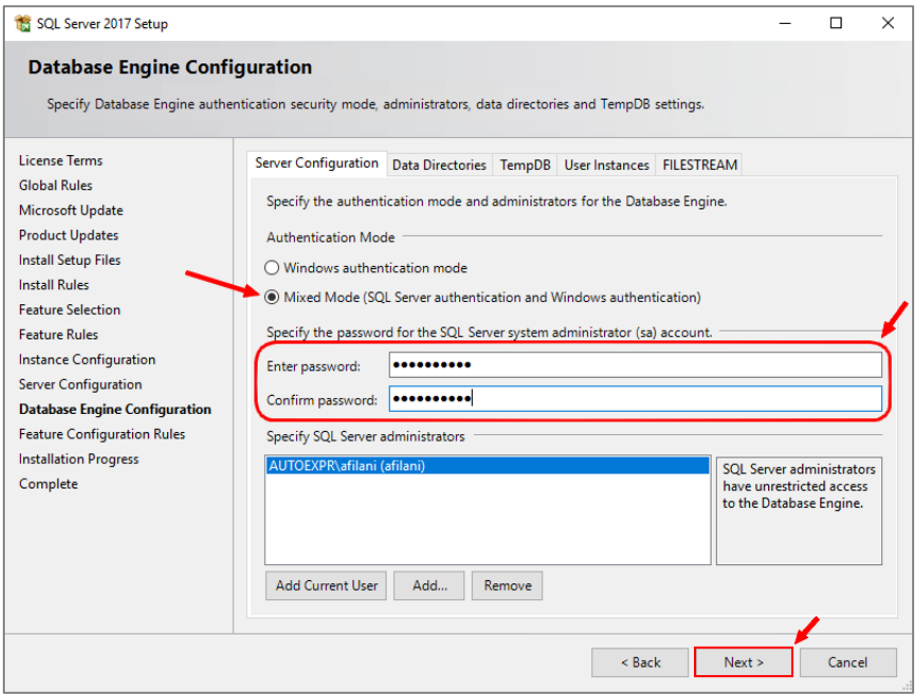
Keep the name, or make your own name, or just select „Default instance“. Click on **„Next“**

11



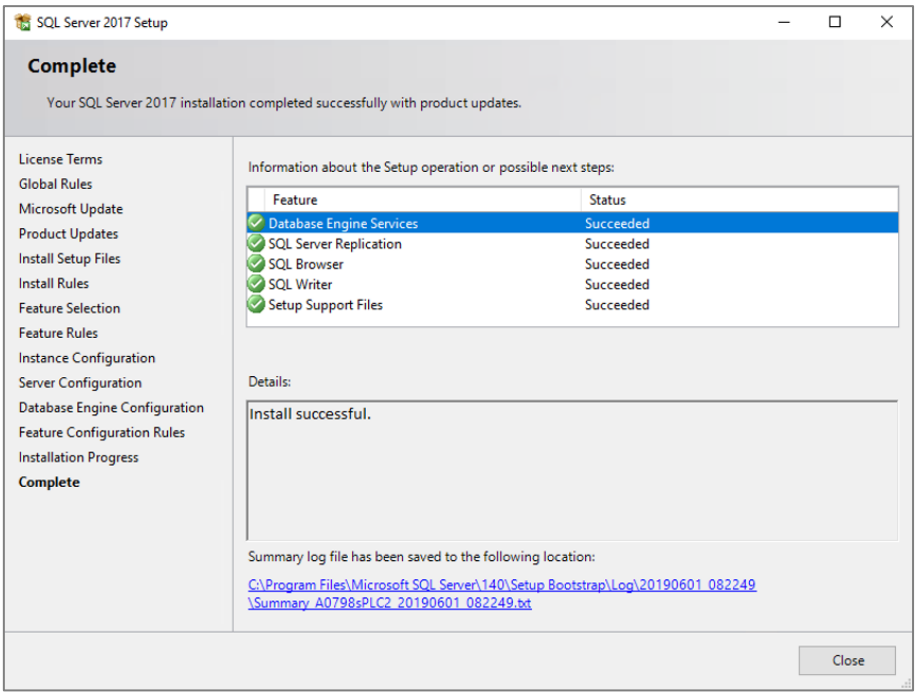
Keep default settings, click on **„Next“**

12



Select **„Mixed Mode“** and specify the password for administrator (**sa**) account, click on **„Next“**

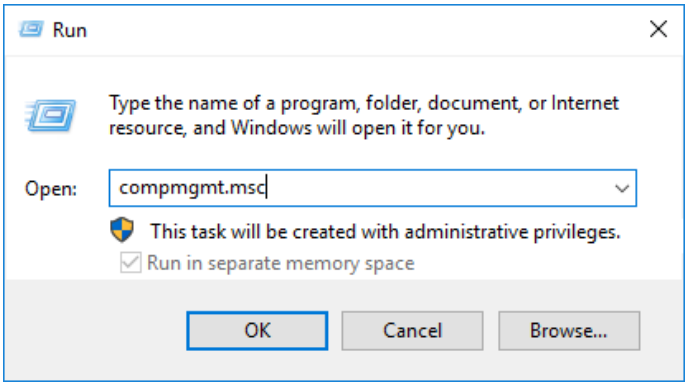
13



After some time you should see this screen telling you to successfully install the SQL Server.

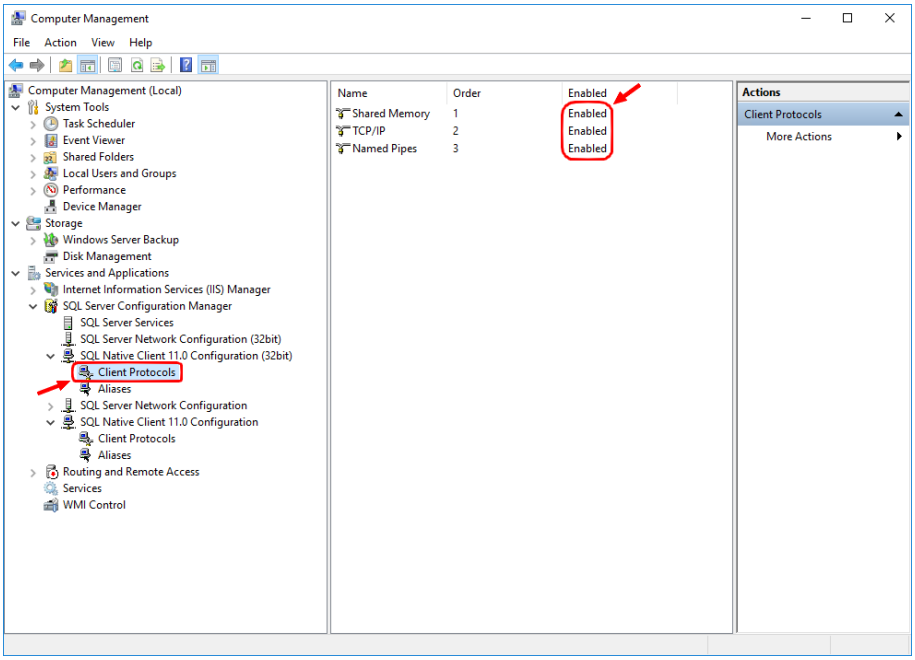
2.1.1 Configuration

1



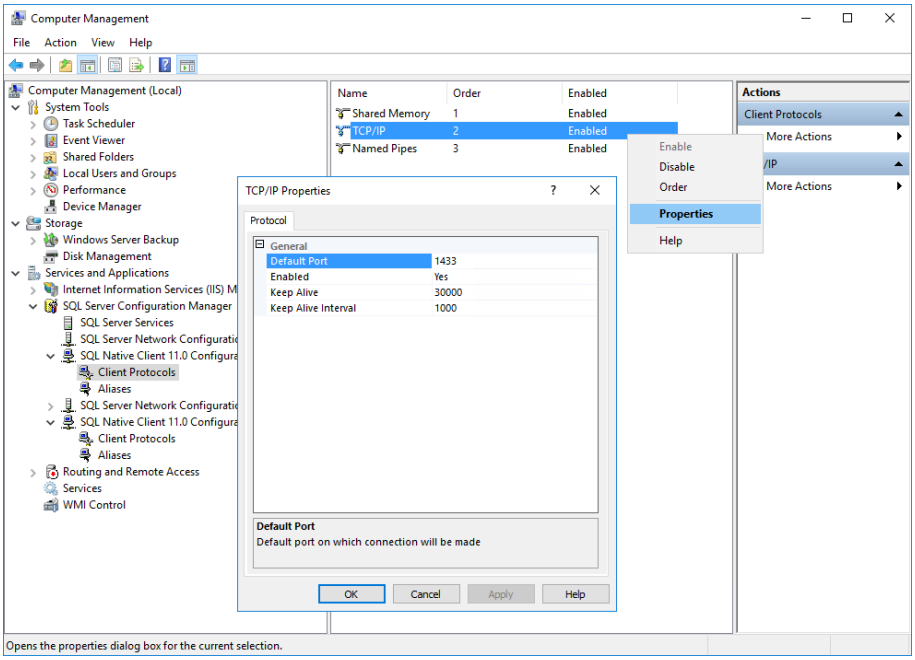
Open **Computer Management**. You can do it by executing the **„Run“** (by simply typing the word **„run“** in the windows start menu) and inserting the **„compmgmt.msc“** command.

2



On left side choose **Services and Applications -> SQL Server Configuration Manager -> SQL Native Client 11.0 Configuration -> Client Protocols**. All 3 protocols has to be Enabled.
Note: SQL Native Client 11.0 Configuration naming is different in every version of SQL Server and also version(32 or 64 bits)!!

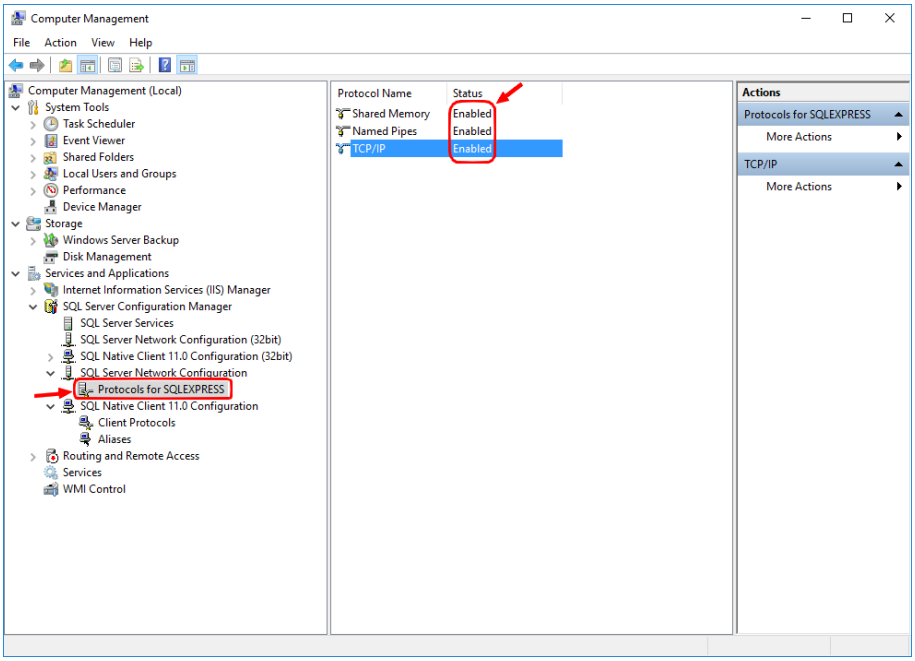
3



By right click on **TCP/IP** you can edit **Properties**:

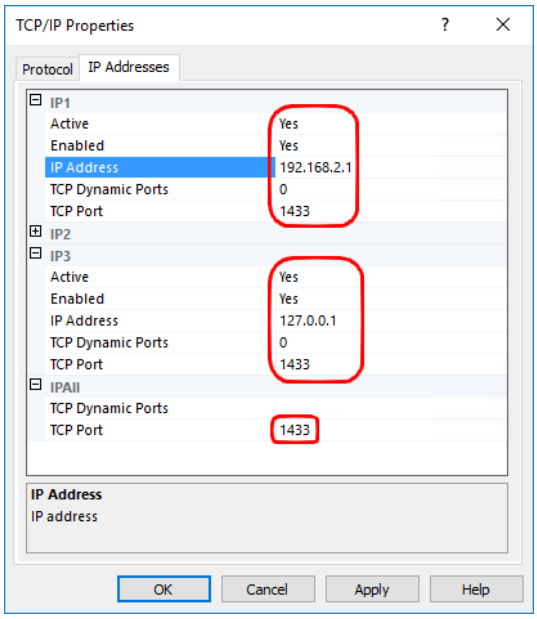
- **Default Port** - port on which will be your SQL server reachable across network
- **Enabled** - Enabling of TCP/IP protocol

4



Another settings will be made in **SQL Server Network Configuration**. Click on **Protocols for SQLEXPRESS**. Also all 3 protocols has to be **Enabled**. We need to check if port is set correctly right click on **TCP/IP** and choose tab **IP Addresses.**

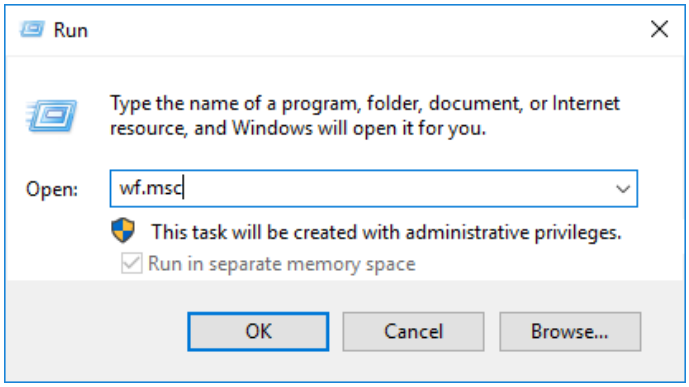
5



Here is set the default port for **SQL Server** and also all **IP addresses** for connection to the SQL Server. And click yes to parameter **Enabled** on each IP address which you want to have active for SQL. We activated **127.0.0.1**(localhost) and **192.168.2.1** in this example.

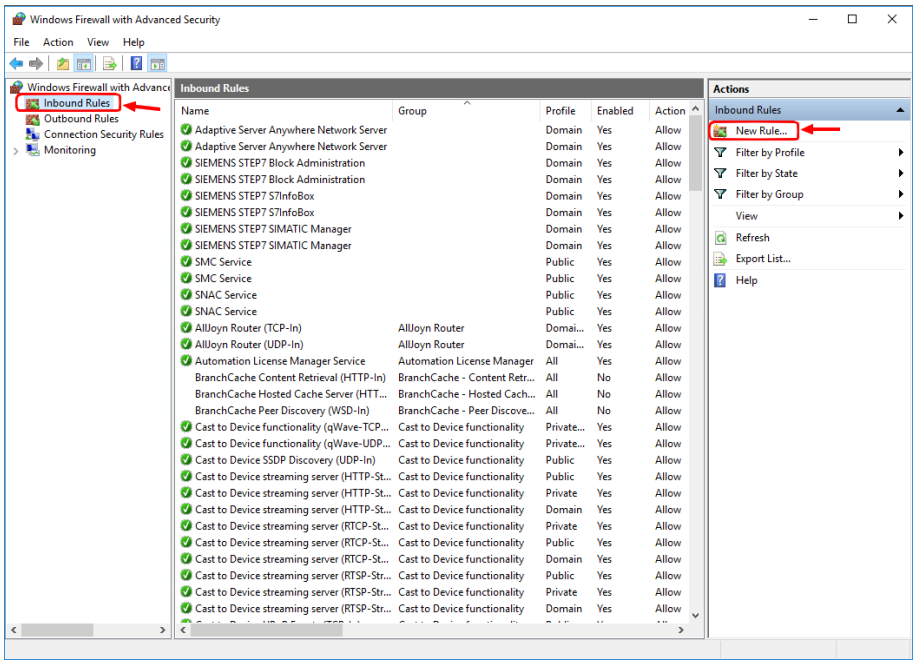
2.1.2 Firewall permission

1



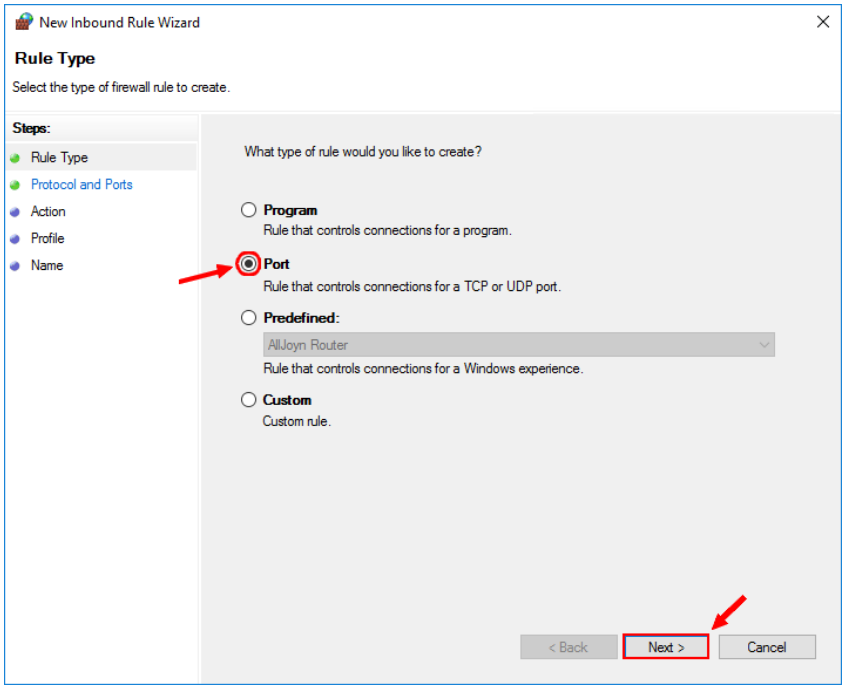
Open **Windows Firewall with Advanced security**. You can do it by executing the **„Run“** (by simply typing the word **„run“** in the windows start menu) and inserting the **„wf.msc“** command.

2



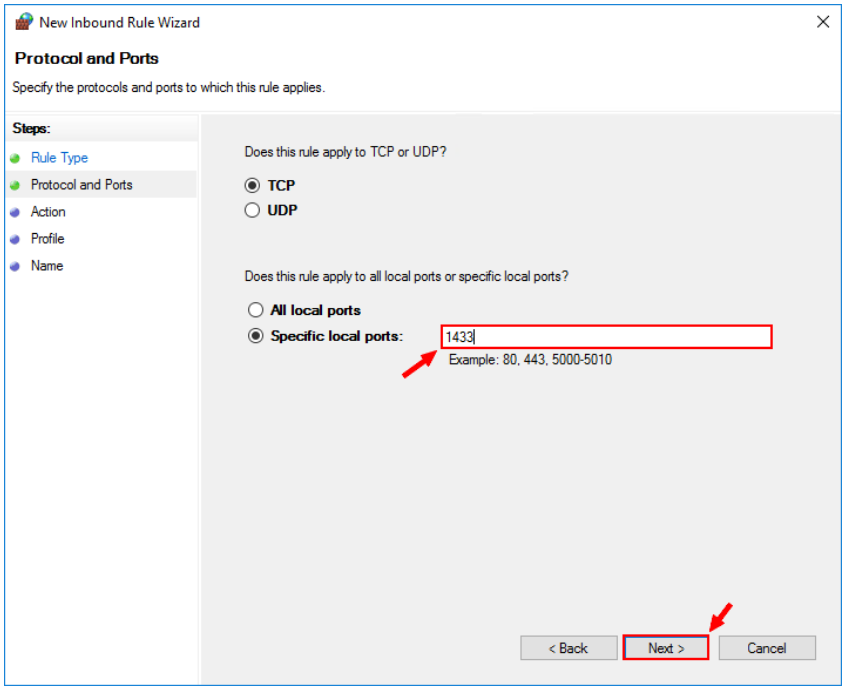
On left side choose **„Inbound Rules“** by left click. This will open list with all Inbound rules set for your PC. On right side choose **„New Rule“**.

3



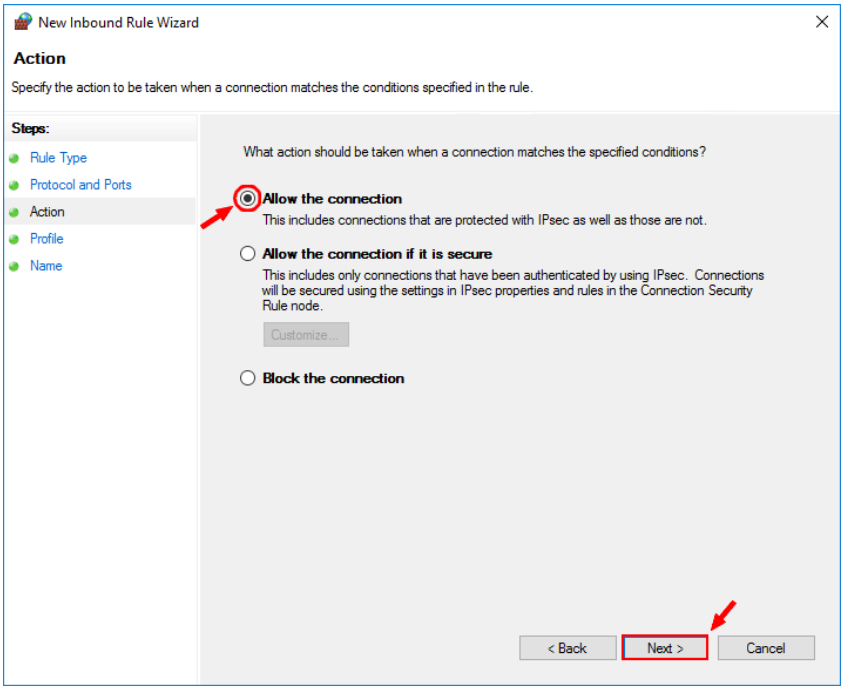
This wizard will help us to set correct firewall rule for our PC. First we have to choose **„Port“** and click on **„Next“**.

4



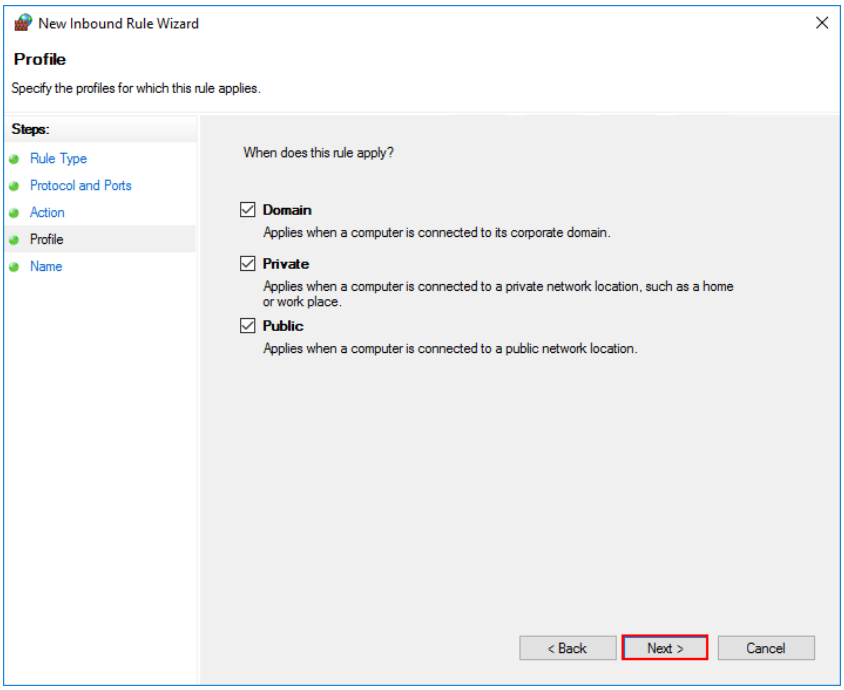
Here we will specify on which port is our SQL server running, **default port** is **1433**. It's adjustable in settings of SQL Server. Click on **„Next“**

5



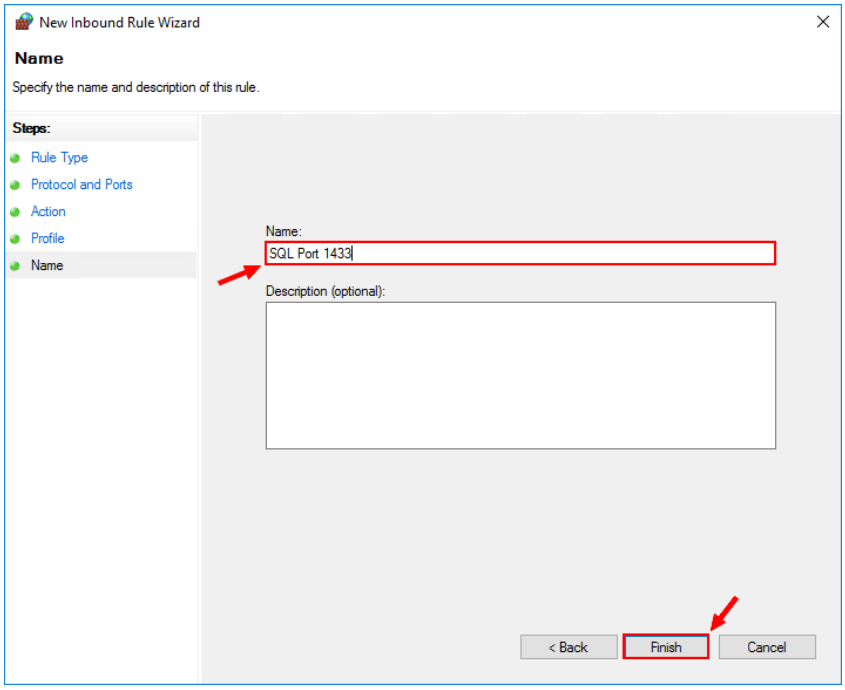
Just chose **„Allow the connection“** and click on **„Next“**

6



This window is selection of profiles when will be rule applied, click on **„Next“**.

7

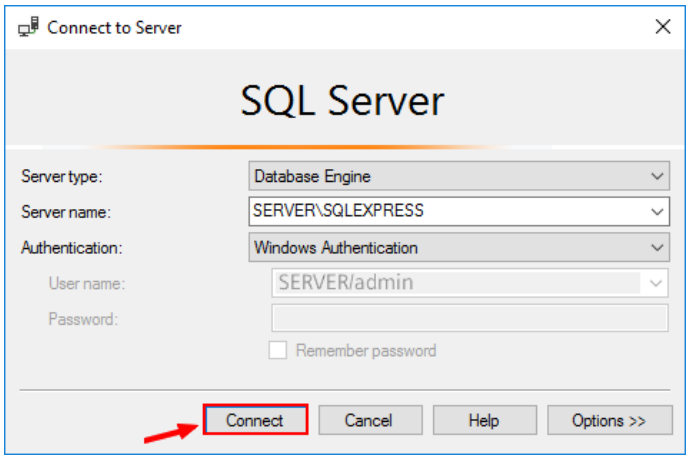


Insert name of the firewall rule and click on **„Finish“**.

2.1.3 Enable SQL Authentification

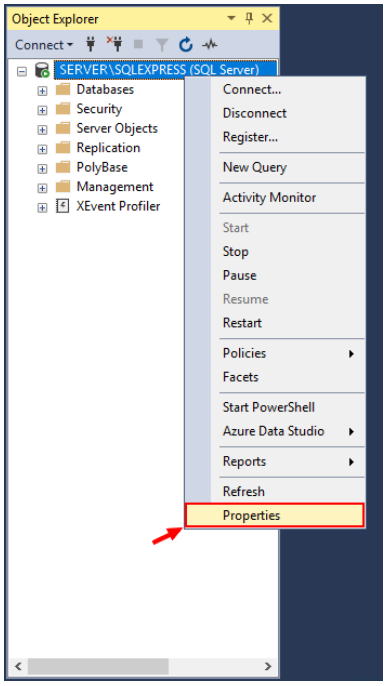
If you need to use SQL server that does not have SQL authentication enabled, here's how it is turned on.

1



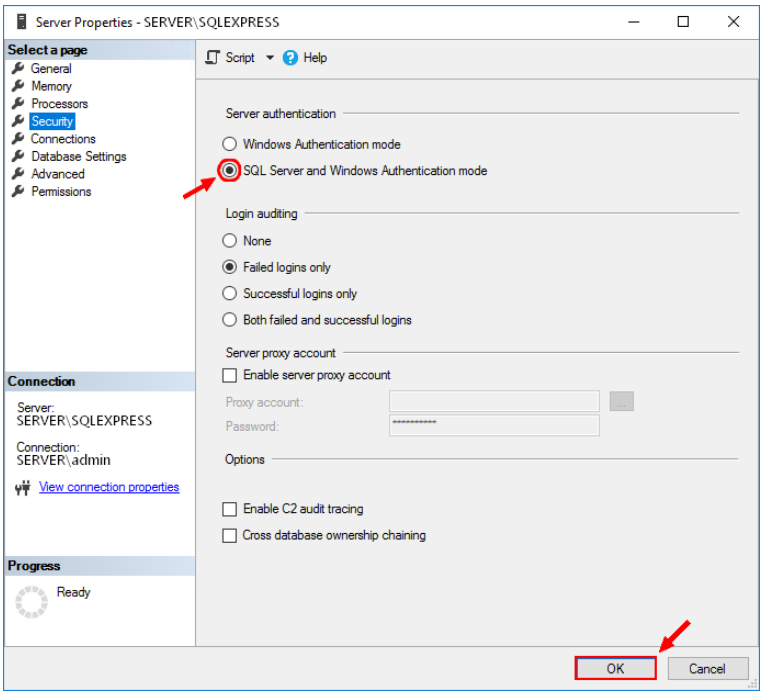
Connect to your database with Microsoft SQL Management Studio. You will login with **Windows authentication**.

2



After successful connection to the SQL Server. Right click on the SQL server icon and choose **Properties**.

3

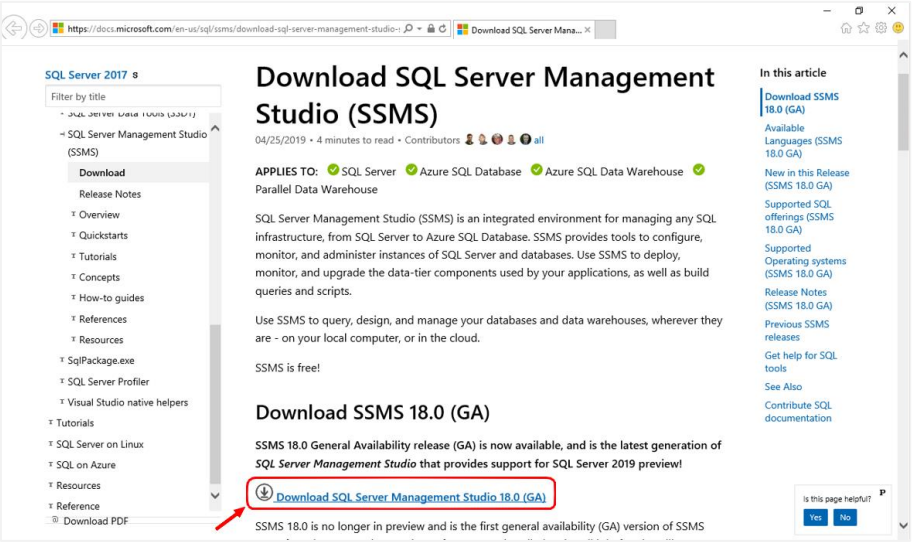


In Server Properties choose page **Security,** and choose **SQL Server and Windows Authentication mode**. Click on **„OK“**

2.2 Microsoft SQL Server Management Studio (SSMS) 18.0

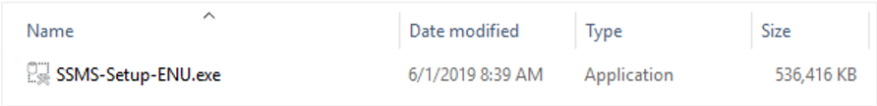
You will need **Microsoft SQL Management Studio (18.0)** to be able to connect and manage the **SQL Server**. The link to download the instalation package is here -> https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017

1



Click on the link **„Download SQL Server Management Studio 18.0 (GA)“**

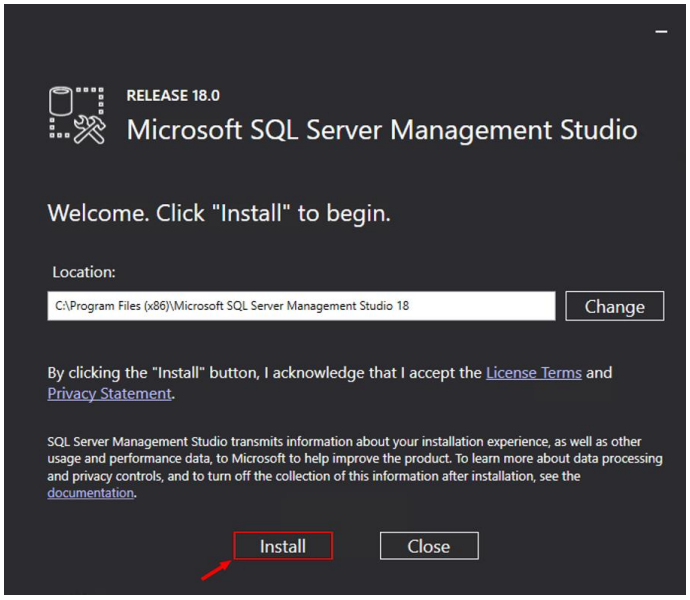
2



Name	Date modified	Type	Size
SSMS-Setup-ENU.exe	6/1/2019 8:39 AM	Application	536,416 KB

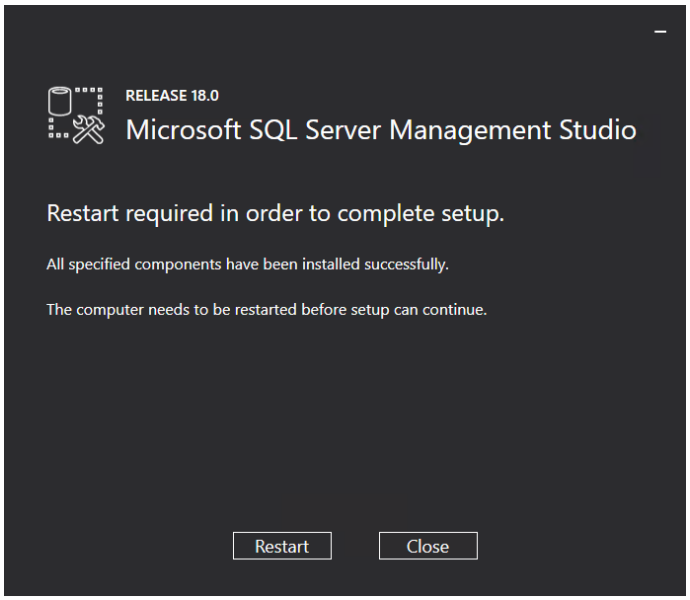
The **SSMS-Setup-ENU.exe** will be downloaded in your download directory. Run this file to continue.

3



Keep or change the location and click on **„Install“**

4

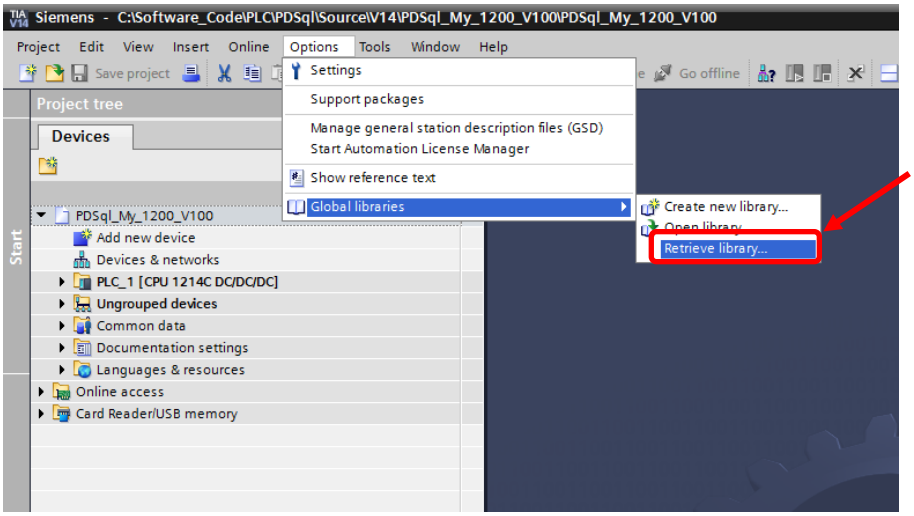


Click on **„Restart“** to complete setup

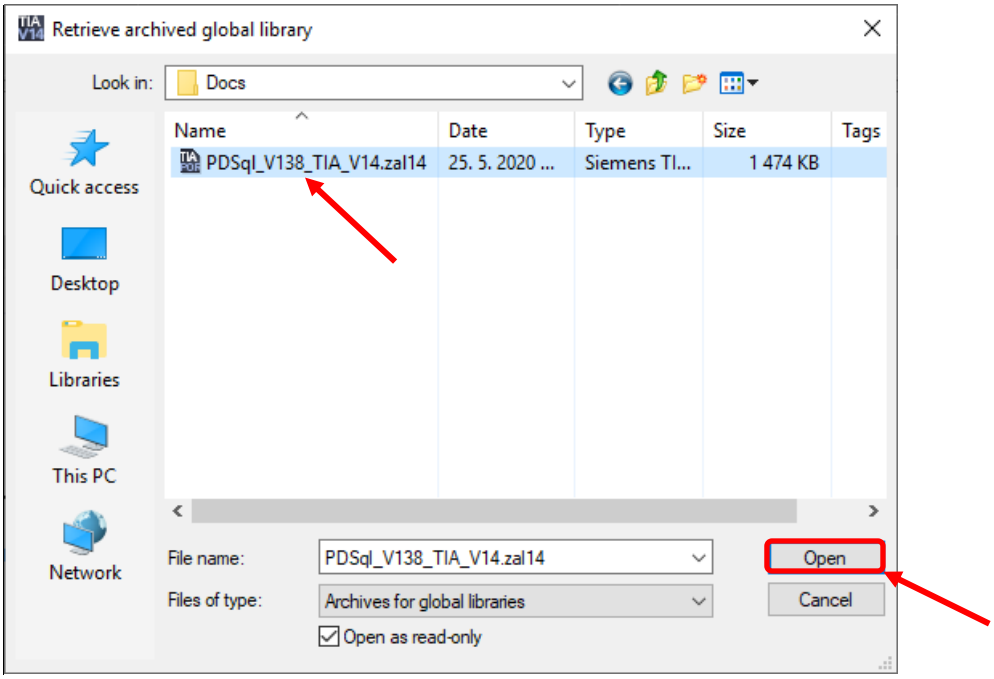
2.3 Instalation PDSql Library to TIA Portal

- Retrive **PDSql Library** from archive.

1

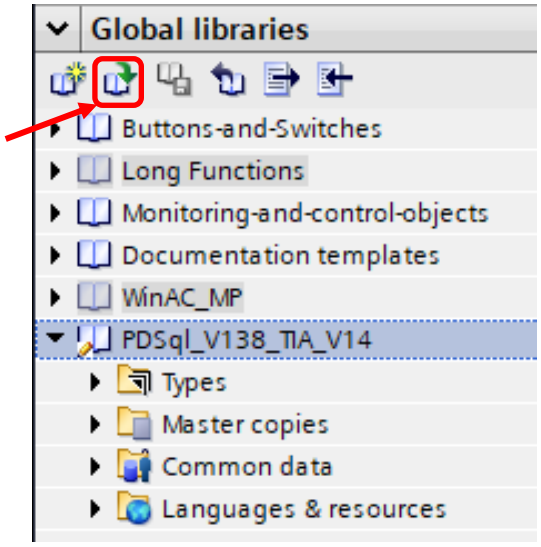


2



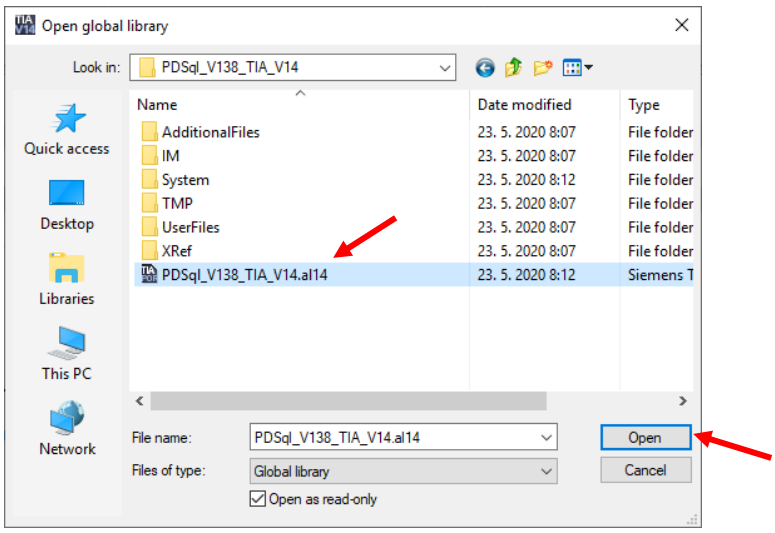
➢ Open **PDSql Library** in **TIA Portal**.

3



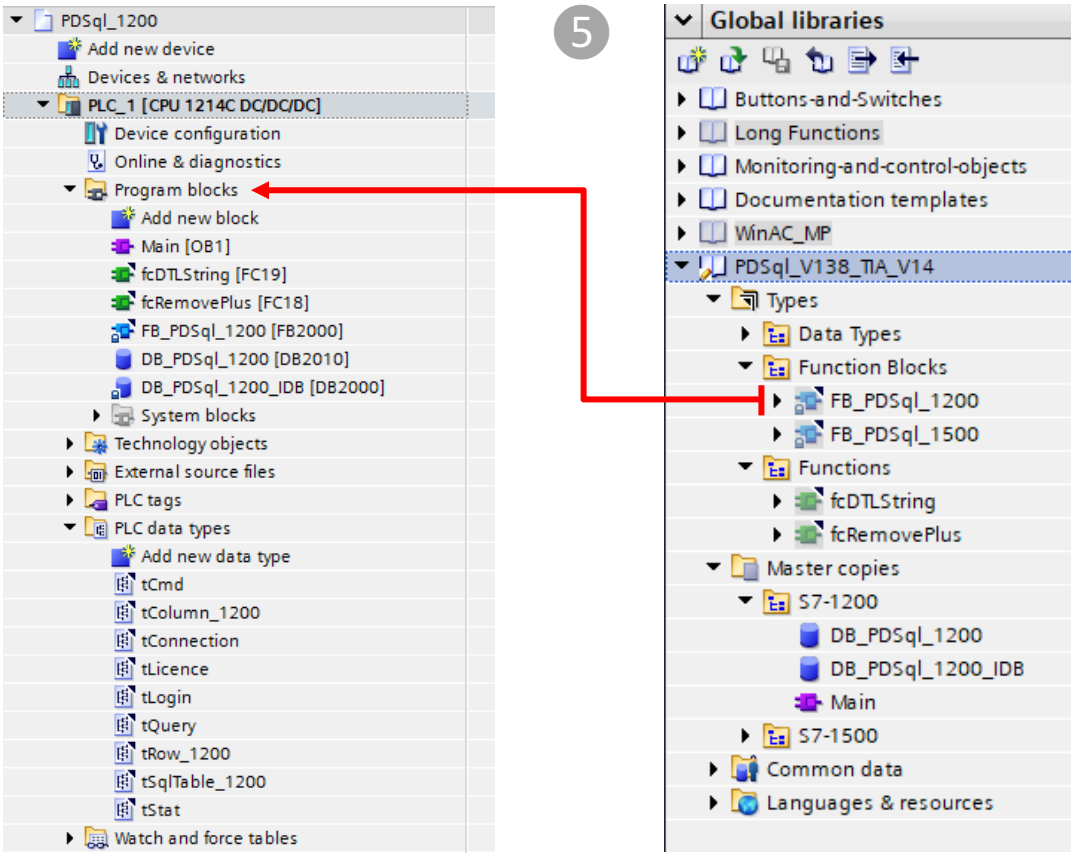
➢ Select **PDSql Library** file and open.

4

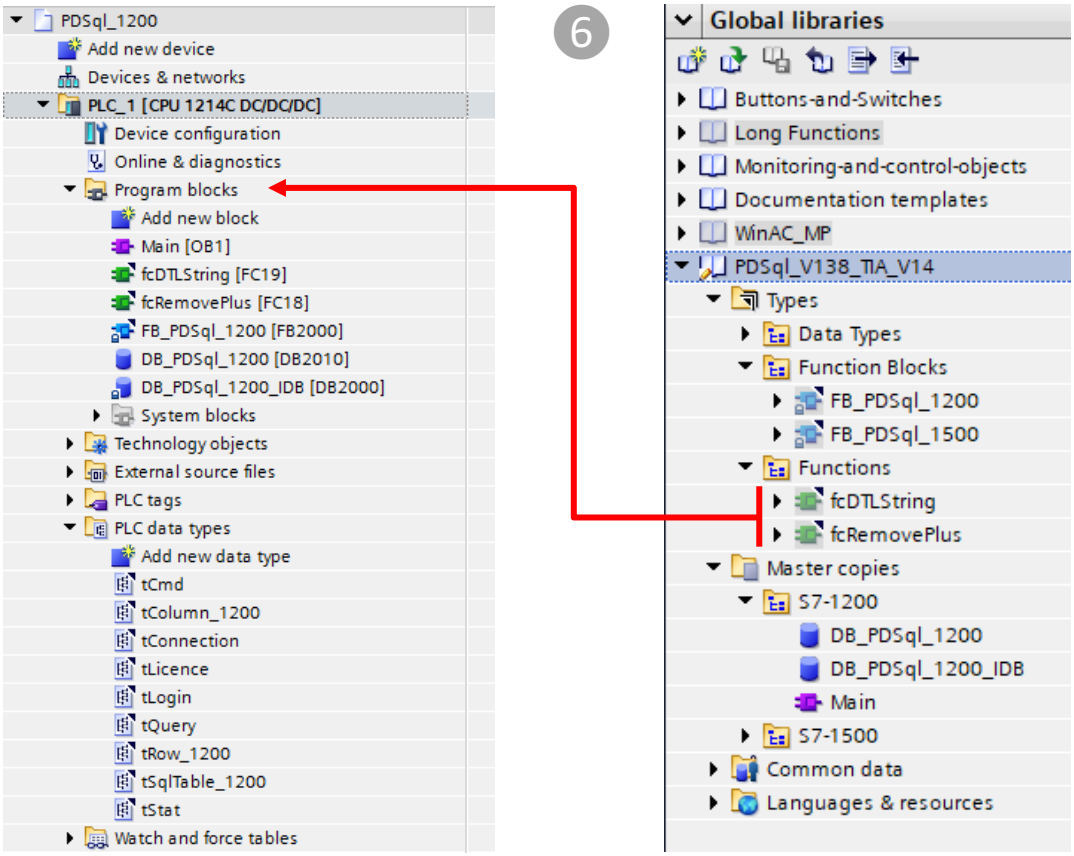


Example for S7-1200 project

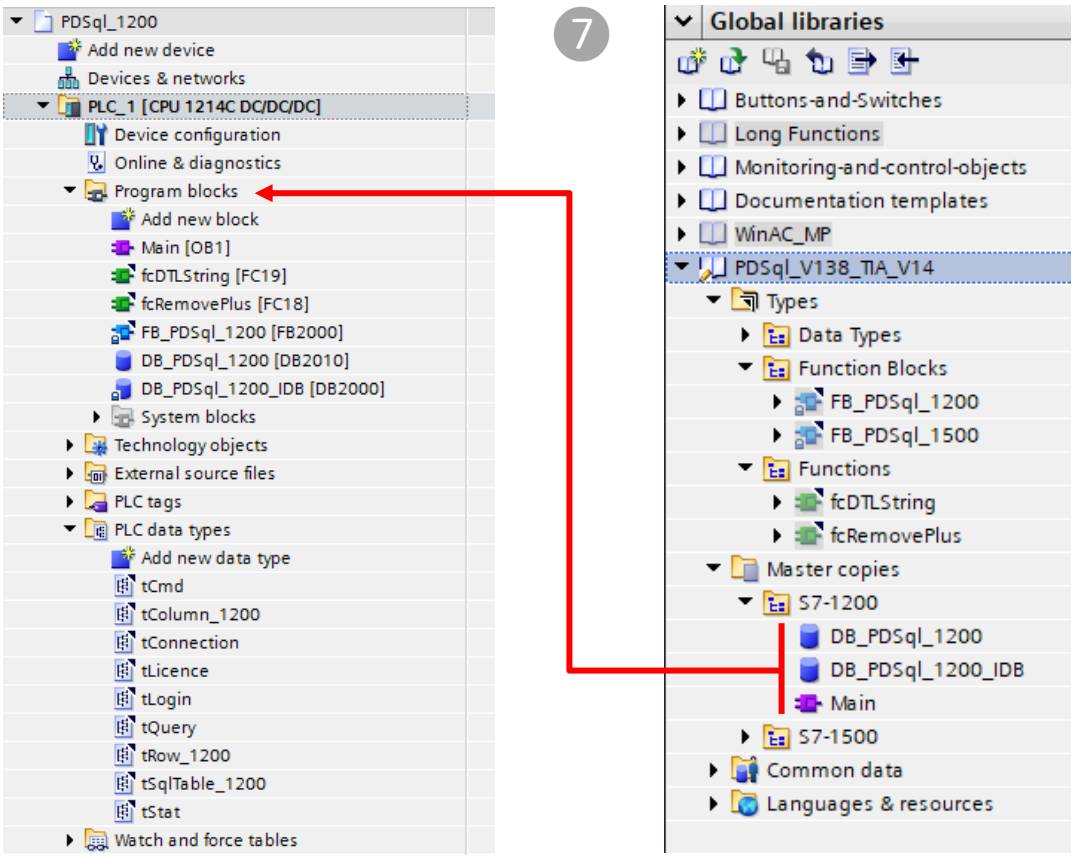
- Drag & Drop **Function Block** from **library** to your **Project / Program blocks**
  *This action will also **automatically** transfer all necessary **data types** from **library** to **Project / PLC data types***



➢ Drag & Drop **Functions** from **library** to your **Project / Program blocks**



- Drag & Drop **Master copies / S7-1200** data blocks from **library** to your **Project / Program blocks**



3 Examples

In the following examples we will use

- **Microsoft Management Studio**
- **Microsoft SQL Server**
- **PDSql Library**.

3.1 Create a table

Using **Microsoft Management Studio**, first create a table named "**TestTable**" that will be used in the following examples and will contain these columns ->

Column name	Data type	Description
[Id]	int	auto increment Id number
[Datetime]	Datetime	date and time
[FruitName]	varchar(20)	fruit name
[Quantity]	int	fruit quantity
[Weight]	real	fruit weight
[CitrusType]	bit	citrus type [1 – Yes, 0 – No]

➢ **SQL statement to create this table**

```
CREATE TABLE [dbo].[TestTable](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [Datetime] [datetime] NULL,
      [FruitName] [varchar](20) NULL,
      [Quantity] [int] NULL,
      [Weight] [real] NULL,
      [CitrusType] [bit] NULL
) ON [PRIMARY]
```

➢ **Insert test rows into the table**

```
INSERT INTO [dbo].[TestTable]
(Datetime,FruitName,Quantity,Weight,CitrusType)
VALUES ('2019-07-01 23:42:18','Apple',10,2.54,0),
       ('2019-07-01 23:48:54','Orange',16,3.28,1),
       ('2019-07-01 23:52:26','Lemon',8,1.64,1),
       ('2019-07-01 23:56:33','Mango',7,4.32,0)
```

Table: „TestTable“

	Id	Datetime	FruitName	Quantity	Weight	CitrusType
1	1	2019-07-01 23:42:18.000	Apple	10	2.54	0
2	2	2019-07-01 23:48:54.000	Orange	16	3.28	1
3	3	2019-07-01 23:52:26.000	Lemon	8	1.64	1
4	4	2019-07-01 23:56:33.000	Mango	7	4.32	0

3.2 **SELECT** row from table

➢ **Read** a **row** from the **table** where **FruitName = "Apple"**

Syntax of the SQL statement in SSMS

```
SELECT * FROM TestTable WHERE FruitName = 'Apple'
```

Syntax of the SQL statement in TIA environment

```
Query[1] := 'SELECT * FROM TestTable WHERE FruitName=$'Apple$'';
Query[2] := '';
```

⚠ In the **TIA environment** you must **add** a **dollar** (**$**) sign **before** each **single quote** in the **query string !**

Execute SQL query with PDSql Library

1. **Execute** SQL query
   `Cmd.ExecuteQuery := TRUE;`
   Set `Cmd.ExecuteQuery` on `TRUE` will also automatically connect to SQL server if not connected yet.

2. **Wait** for the execution of the SQL query
   `(Stat.ExecutedOK OR Stat.Error) = TRUE;`

3. If **ExecutedOK** then result is stored in the **SQLtable**

	Id	Datetime	FruitName	Quantity	Weight	CitrusType
1	1	2019-07-01 23:42:18.000	Apple	10	2.54	0

Connected	Bool	false	TRUE
Busy	Bool	false	FALSE
ExecutedOK	Bool	false	TRUE
Error	Bool	false	FALSE
Status	Word	16#0	16#0000
Message	String	''	'1 row(s) affected'

The status after SELECT command

SQLTable	"tSqlTable_12...		
ColumnCount	UInt	0	6
RowCount	UInt	0	1
MaxColumns	UInt	10	10
MaxRows	UInt	10	10
MaxStringLength	USInt	50	50

The number of rows and columns returned after SELECT command

Columns	Array[1..10] of "tCo...		
Columns[1]	"tColumn_1200"		
Name	String	''	'Id'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#38
LenSize	Byte	16#0	16#04
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	FALSE
CaseSens	Bool	false	FALSE
Identity	Bool	false	TRUE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#00
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	FALSE
Null	Bool	false	FALSE
DInt	DInt	0	1
LReal	LReal	0.0	0.0
String	String[50]	''	''
Date...	DTL	DTL#1970-01-01-	DTL#1970-01-01-00:00:00

Id = 1

Columns[2]	"tColumn_1200"		
Name	String	''	'Datetime'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#6F
LenSize	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	TRUE
CaseSens	Bool	false	FALSE
Identity	Bool	false	FALSE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#02
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	FALSE
Null	Bool	false	FALSE
DInt	DInt	0	0
LReal	LReal	0.0	0.0
String	String[50]	''	''
Date...	DTL	DTL#1970-01-01-	DTL#2019-07-01-23:42:18

Datetime = "2019-07-01-23:42:18"

Columns[3]	"tColumn_1200"		
Name	String	''	'FruitName'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#A7
LenSize	Byte	16#0	16#14
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	TRUE
CaseSens	Bool	false	FALSE
Identity	Bool	false	FALSE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#02
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	FALSE
Null	Bool	false	FALSE
DInt	DInt	0	0
LReal	LReal	0.0	0.0
String	String[50]	''	'Apple'
Date...	DTL	DTL#1970-01-01	DTL#1970-01-01-00:00:00

FruitName = "Apple"

Columns[4]	"tColumn_1200"		
Name	String	''	'Quantity'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#26
LenSize	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	TRUE
CaseSens	Bool	false	FALSE
Identity	Bool	false	FALSE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#02
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	TRUE
Null	Bool	false	FALSE
DInt	DInt	0	10
LReal	LReal	0.0	0.0
String	String[50]	''	''
Date...	DTL	DTL#1970-01-01	DTL#1970-01-01-00:00:00

Quantity = 10

Columns[5]	"tColumn_1200"		
Name	String	''	'Weight'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#6D
LenSize	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	TRUE
CaseSens	Bool	false	FALSE
Identity	Bool	false	FALSE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#02
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	FALSE
Null	Bool	false	FALSE
DInt	DInt	0	0
LReal	LReal	0.0	2.53999996185303
String	String[50]	''	''
Date...	DTL	DTL#1970-01-01	DTL#1970-01-01-00:00:00

Weight = 2.54

Columns[6]	"tColumn_1200"		
Name	String	''	'CitrusType'
UserType	Byte	16#0	16#00
Type	Byte	16#0	16#68
LenSize	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
TypeNumer...	Byte	16#0	16#00
Nullable	Bool	false	TRUE
CaseSens	Bool	false	FALSE
Identity	Bool	false	FALSE
Computed	Bool	false	FALSE
FixedLenCL...	Bool	false	FALSE
UsUpdatable	Byte	16#0	16#02
Rows	Array[1..10] of "tRo...		
Rows[1]	"tRow_1200"		
Bool	Bool	false	FALSE
Null	Bool	false	FALSE
DInt	DInt	0	0
LReal	LReal	0.0	0.0
String	String[50]	''	''
Date...	DTL	DTL#1970-01-01-	DTL#1970-01-01-00:00:00

CitrusType = FALSE

4. If **Error** then `Stat.Status` contains an **error code** and `Stat.Message` contains an **error message**

3.3 **INSERT** row to the table

➢ **Write a new row into the table**

Syntax of the SQL statement in <u>SSMS</u>

```
INSERT INTO TestTable (Datetime,FruitName,Quantity,Weight,CitrusType)
VALUES ('2019-07-02 10:42:56','Banana',24,8.48,0)
```

Syntax of the SQL statement in <u>TIA environment</u>

```
Query[1] := 'INSERT INTO TestTable
(Datetime,FruitName,Quantity,Weight,CitrusType) VALUES ($'2019-07-02
10:42:56$',$'Banana$',24,8.48,0)';
Query[2] := '';
```

⚠ In the **TIA environment** you must **add** a **dollar ($)** sign **before** each **single quote** in the **query string !**

Compose INSERT string from input variables (SCL language)

```
#dtlDateTime := '2019-07-02 10:42:56';
#sFruitName := 'Banana';
#iQuantity := 24;
#rWeight := 8.48;
#bCitrusType := FALSE;

#str := 'INSERT INTO TestTable(Datetime,FruitName,Quantity,Weight,CitrusType) VALUES($'';
#str := CONCAT(IN1 := #str, IN2 := "fcDTLString"(#dtlDateTime));
#str := CONCAT(IN1 := #str, IN2 := '$',$'');
#str := CONCAT(IN1 := #str, IN2 := #sFruitName);
#str := CONCAT(IN1 := #str, IN2 := '$',');
#str := CONCAT(IN1 := #str, IN2 := INT_TO_STRING(#iQuantity));
#str := CONCAT(IN1 := #str, IN2 := ',');
#str := CONCAT(IN1 := #str, IN2 := REAL_TO_STRING(#rWeight));
#str := CONCAT(IN1 := #str, IN2 := ',');
#str := CONCAT(IN1 := #str, IN2 := INT_TO_STRING(BOOL_TO_BYTE(#bCitrusType)));
#str := CONCAT(IN1 := #str, IN2 := ')');

"DB_PDSql_1200".Query.Query[1] := #str;
"DB_PDSql_1200".Query.Query[2] := '';
```

Execute SQL query with PDSql Library

1. **Execute** SQL query

   `Cmd.ExecuteQuery := TRUE;`

   Set Cmd.ExecuteQuery on TRUE will also automatically connect to SQL server if not connected yet.

2. **Wait** for the execution of the SQL query

   `(Stat.ExecutedOK OR Stat.Error) = TRUE;`

3. If **ExecutedOK** then a row was inserted successfully

Connected	Bool	false	TRUE
Busy	Bool	false	FALSE
ExecutedOK	Bool	false	TRUE
Error	Bool	false	FALSE
Status	Word	16#0	16#0000
Message	String	''	'1 row(s) affected'

"**TestTable**" after executing the **INSERT** command

	Id	Datetime	FruitName	Quantity	Weight	CitrusType
1	1	2019-07-01 23:42:18.000	Apple	10	2.54	0
2	2	2019-07-01 23:48:54.000	Orange	16	3.28	1
3	3	2019-07-01 23:52:26.000	Lemon	8	1.64	1
4	4	2019-07-01 23:56:33.000	Mango	7	4.32	0
5	5	2019-07-02 10:42:56.000	Banana	24	8.48	0

4. If **Error** then `Stat.Status` contains an **error code** and `Stat.Message` contains an **error message**

3.4 **UPDATE** row in table

➢ **Update** an existing **row** in the **table** where **FruitName = "Mango"**

Syntax of the SQL statement in <u>SSMS</u>

```
UPDATE TestTable SET Quantity = 32, Weight = 12.68 WHERE FruitName =
'Mango'
```

Syntax of the SQL statement in <u>TIA environment</u>

```
Query[1] := UPDATE TestTable SET Quantity=32, Weight=12.68 WHERE
FruitName = $'Mango$'';
Query[2] := '';
```

⚠ In the **TIA environment** you must **add** a **dollar** (**$**) sign **before** each **single quote** in the **query string !**

Compose UPDATE string from input variables (SCL language)

```
#sFruitName := 'Mango';
#iQuantity := 32;
#rWeight := 12.68;

#str := 'UPDATE TestTable SET Quantity=';
#str := CONCAT(IN1 := #str, IN2 := INT_TO_STRING(#iQuantity));
#str := CONCAT(IN1 := #str, IN2 := ',Weight=');
#str := CONCAT(IN1 := #str, IN2 := REAL_TO_STRING(#rWeight));
#str := CONCAT(IN1 := #str, IN2 := ' WHERE FruitName=$'');
#str := CONCAT(IN1 := #str, IN2 := #sFruitName);
#str := CONCAT(IN1 := #str, IN2 := '$'');

"DB_PDSql_1200".Query.Query[1] := #str;
"DB_PDSql_1200".Query.Query[2] := '';
```

Execute SQL query with PDSql Library

1. **Execute** SQL query
   `Cmd.ExecuteQuery := TRUE;`
   Set `Cmd.ExecuteQuery` on `TRUE` will also automatically connect to SQL server if not connected yet.

2. **Wait** for the execution of the SQL query
   `(Stat.ExecutedOK OR Stat.Error) = TRUE;`

3. If **ExecutedOK** then a row was updated successfully

Connected	Bool	false	TRUE
Busy	Bool	false	FALSE
ExecutedOK	Bool	false	TRUE
Error	Bool	false	FALSE
Status	Word	16#0	16#0000
Message	String	''	'1 row(s) affected'

"**TestTable**" after executing the **UPDATE** command

	Id	Datetime	FruitName	Quantity	Weight	CitrusType
1	1	2019-07-01 23:42:18.000	Apple	10	2.54	0
2	2	2019-07-01 23:48:54.000	Orange	16	3.28	1
3	3	2019-07-01 23:52:26.000	Lemon	8	1.64	1
4	4	2019-07-01 23:56:33.000	Mango	32	12.68	0
5	5	2019-07-02 10:42:56.000	Banana	24	8.48	0

4. If **Error** then `Stat.Status` contains an **error code** and
   `Stat.Message` contains an **error message**

3.5 **DELETE** row from table

➢ **Delete** an existing **row** from the **table** where **FruitName = "Lemon"**

Syntax of the SQL statement in SSMS

```
DELETE FROM TestTable WHERE FruitName = 'Lemon'
```

Syntax of the SQL statement in TIA environment

```
Query[1] := 'DELETE FROM TestTable WHERE FruitName=$'Lemon$'';
Query[2] := '';
```

⚠ In the **TIA environment** you must **add** a **dollar** ($) sign **before** each **single quote** in the **query string !**

Compose DELETE string from input variables (SCL language)

```
#sFruitName := 'Lemon';

#str := 'DELETE FROM TestTable WHERE FruitName=$'';
#str := CONCAT(IN1 := #str, IN2 := #sFruitName);
#str := CONCAT(IN1 := #str, IN2 := '$'');

"DB_PDSql_1200".Query.Query[1] := #str;
"DB_PDSql_1200".Query.Query[2] := '';
```

Execute SQL query with PDSql Library

1. **Execute** SQL query
   `Cmd.ExecuteQuery := TRUE;`
   Set `Cmd.ExecuteQuery` on `TRUE` will also automatically connect to SQL server if not connected yet.

2. **Wait** for the execution of the SQL query
   `(Stat.ExecutedOK OR Stat.Error) = TRUE;`

3. If **ExecutedOK** then the existing row was deleted successfully

Connected	Bool	false	TRUE
Busy	Bool	false	FALSE
ExecutedOK	Bool	false	TRUE
Error	Bool	false	FALSE
Status	Word	16#0	16#0000
Message	String	''	'1 row(s) affected'

"**TestTable**" after executing the **DELETE** command

	Id	Datetime	FruitName	Quantity	Weight	CitrusType
1	1	2019-07-01 23:42:18.000	Apple	10	2.54	0
2	2	2019-07-01 23:48:54.000	Orange	16	3.28	1
3	4	2019-07-01 23:56:33.000	Mango	32	12.68	0
4	5	2019-07-02 10:42:56.000	Banana	24	8.48	0

4. If **Error** then `Stat.Status` contains an **error code** and `Stat.Message` contains an **error message**