Co-funded by the
Erasmus+ Programme
of the European Union

SMARTRAIN

Project 2020-1-ES01-KA202-082778

INTELLECTUAL OUTPUT O1

INNOVATIVE OPEN TRAINING MATERIAL ON CIRCULAR ECONOMY AND DIGITISATION FOR VET LEARNERS IN THE WOOD&FURNITURE SECTOR

UNIT 5. Practical cases of sensor handling, programming, insertion in furniture

Contents

1 INTRODUCTION

In this unit, some basic knowledge of networks, programming languages, and cloud platforms will be introduced. With that ground knowledge, we will be able to adapt these tools to our problem of developing and improving smart furniture.

2 UNDERSTANDING COMPUTER NETWORKS AND PROTOCOLS

Since the first moment that more than one computer existed, the problem of how to connect them in a way that they could share information arose. The development of solutions that solve this problem gave birth to what's known as computer networks nowadays.

In a computer network, we have 2 or more devices connected and it can fluctuate from a very simple network (like the one you can have in your home with many devices connected to the same router) to modern super complex networks which involve millions of devices (like the internet). However, if we want to implement a computer network, we need all the devices in the network to share the same language and follow the same set of rules.

This applies to human communications as well, imagine I write a letter in French to someone who only speaks Czech, and instead of throwing it in a mailbox, I throw it in a bottle to the sea. If the receiver in this scenario is expecting an email, it doesn't matter how hard they try, the message will never arrive.

To solve this problem, protocols exist. Protocols are nothing but a set of rules and procedures that each participant in a network must follow to ensure the communication and all the actions defined are executed properly. Many different protocols exist because many different types of communications should be approached in many different ways.

Here's a list of popular protocols that you may have heard of:

- **HTTP – HyperText Transfer Protocol.** Used to access websites and other types of communications with a server-client structure.
- **FTP – File Transfer Protocol**. Used to share files between 2 or more devices.
- **IP** – **Internet Protocol.** Used to identify users on the internet.
- **Etc**

Some of the most popular protocols for IoT systems are MQTT and OPC UA. To use them, both the sensor and the data receiver must be compatible with the chosen protocol.

2.1 THE OPC UA PROTOCOL

In this document, we will mostly talk about OPC UA because it has some very visual, easy-to-use tools that will allow us to develop examples easier. OPC UA follows a Server-Client architecture, which means that the sensors or the system attached to it (Arduino for example as developed in previous sections) will be known as a server and its job will be to provide data to the client whenever the second demands it.

Let's try some tools to start using this technology. First, let's download a simulation server, the brand Prosys has one completely free that you can download with this link:

https://downloads.prosysopc.com/opc-ua-simulation-server-downloads.php

Once downloaded we will install it and open it. We should be seeing something like this:

Figure 1. OPC UA Simulation Server initial screen

Now let's move to the "Simulation" tab.

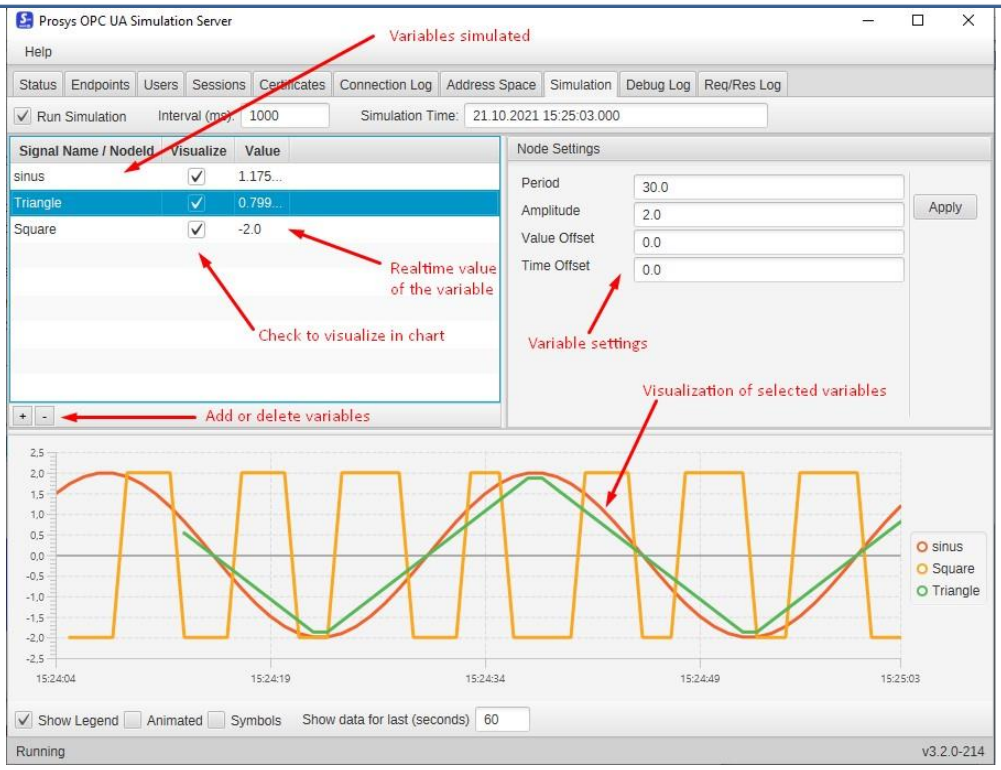


Figure 2. Simulation tab of OPC UA Simulation Server

Here we can manage the simulated variables. You can create new variables by pressing the "+" button and selecting the type of variable you want. If you click on the checkbox, the graph of the values will start appearing in real-time. Now create 3 variables and make them visible in the chart.

Now we have a server that simulates the values of at least 3 sensors. Let's now create and connect a client to check the variables from anywhere else. This second step can be done on the same computer as the one the server is running or in any other one which is connected to the same network. Let's download, install and open the client with this link:

https://downloads.prosysopc.com/opc-ua-client-downloads.php

Once opened you should be seeing something like this:

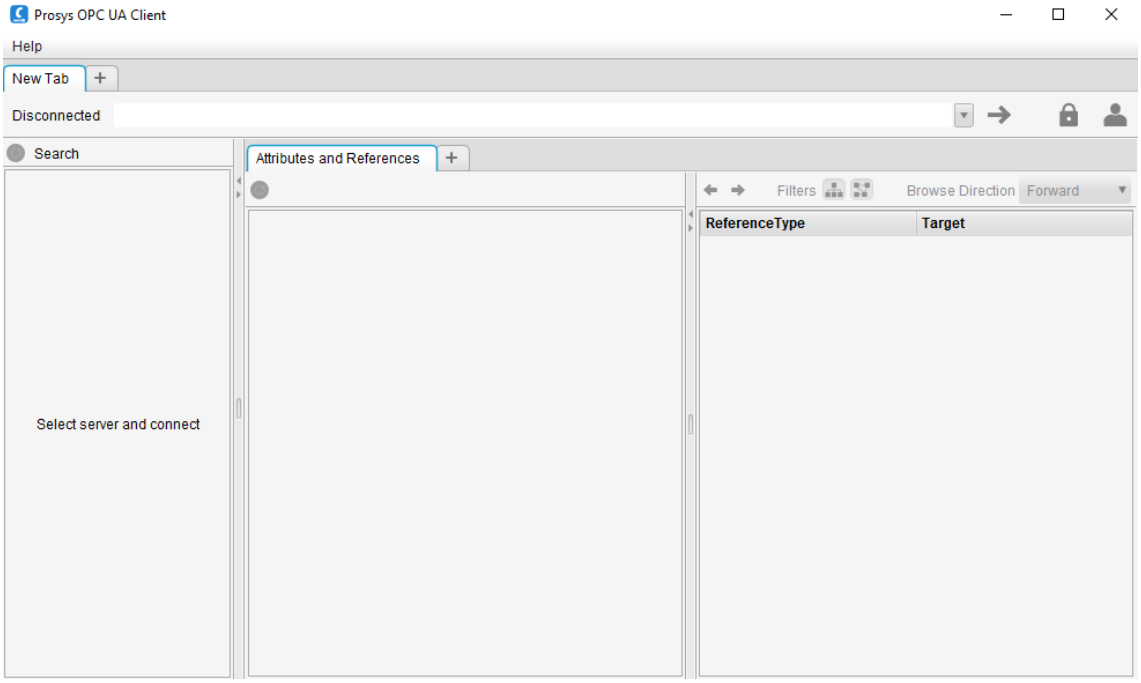


Figure 3. Aspect of Client page

Now we have to write the address of the server, you can find it in the STATUS tab of the simulation server referred to as “Connection Address UA TCP”. Once copied we can press the arrow button and see how the client connects to the server and now we can explore the values of all the variables simulated by the server. OPC UA implements an encryption mechanism, but in this case, we won’t use it, so press None-None, as shown in the image below:

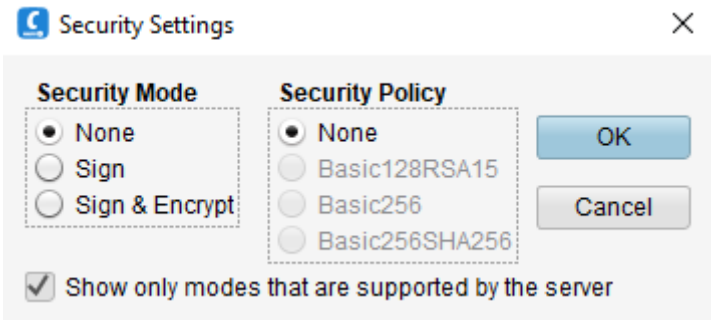
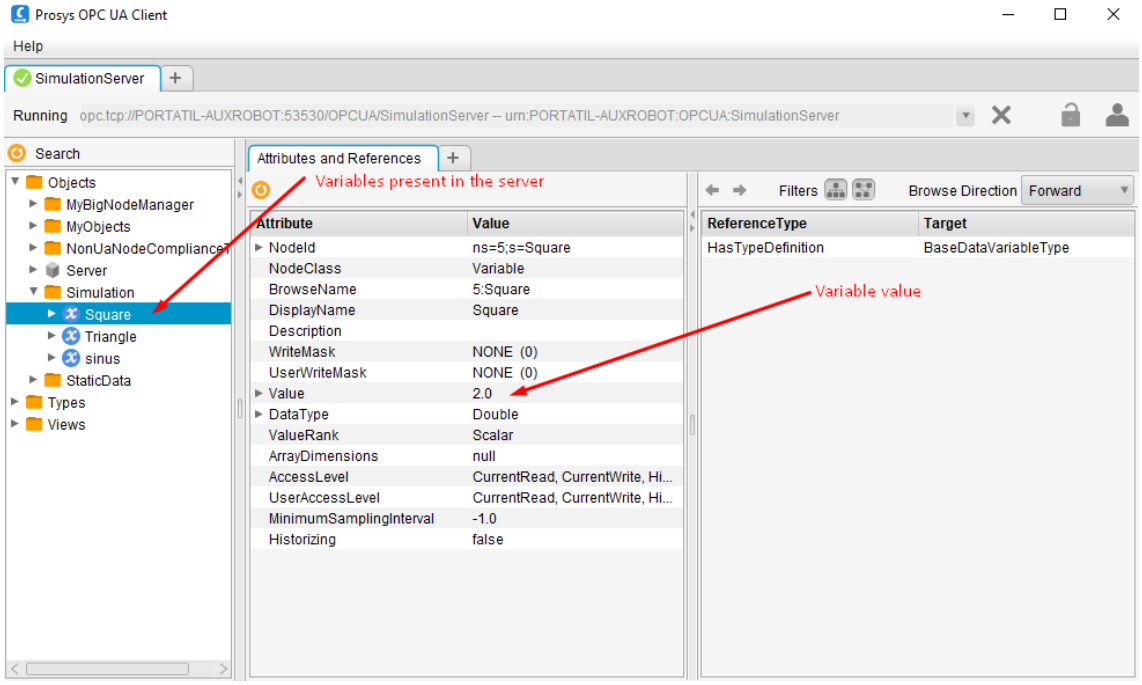


Figure 4. Configuration of Security settings

Now we can access the variables under Objects, Simulation:



Currently, we have already created both a server and a client. In the next section, we'll explore how we can automate the process and include data treatment using programming languages.

2.2 DATA TREATMENT WITH PROGRAMMING LANGUAGES

Once the problem of how to access the data collected by the sensors has been addressed, in this section, we will introduce how to manipulate all the measures read.

Programming languages are powerful tools that allow us to create programs i.e. we can use them to make computers do exactly what we want. Programming languages are the core base of every computer system since they are the ones that define the behaviour in each second and the response it should give to the user's actions. There are many programming languages as you can see in the image below:

Figure 5. Some common Programming languages

Each one is better for some occasions and worse for others, for example, Swift is mainly used to develop IOs applications (Apps for iPhones and Ipads), while Kotlin is mainly used in Android apps.

In this document, we will use python for many reasons including how easy it is to learn, how easy it is to install and expand, the fact that has great integration with OPC UA, and that there are great tools to use, between other reasons.

2.2.1 INSTALLING PYTHON, SPYDER, AND THE OPCUA LIBRARY

To install Python we can download Spyder, a program to develop python programs and it will also install Python. We can download and install Spyder with this link:

https://www.spyder-ide.org/

Once installed, we should see something like this when opened:

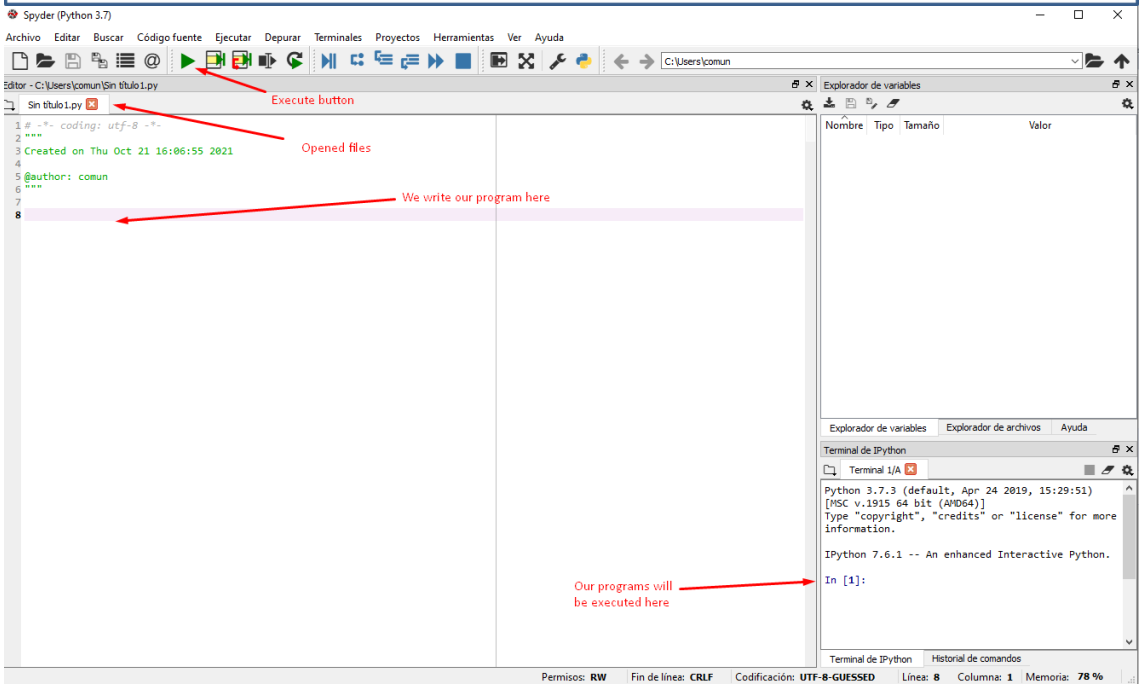


Figure 6. Screen of Spyder programme

Programming is an ability that takes time to learn and master, and of course, this document doesn't intend to be a programming guide or teach how to program. For this reason, only the example of how to get the data and send it to a database will be explained here. Don't worry if you don't fully understand the process being followed in the next lines if you don't have previous programming knowledge.

The first thing we have to do is to install the OPC-UA Python library which contains all the necessary tools to access or create an OPC-UA Server. But to do so, we first have to install PIP, the Packages Installer for Python, the easiest way is to download and execute this file in Python:

https://bootstrap.pypa.io/get-pip.py.

Save it to your computer, then in Spyder, you can open it by pressing File > Open… and selecting the downloaded file. Then execute it by pressing the execute button represented with a green triangle. Now this will install PIP and it will be ready to install the opcua library for Python. Now write this command in the console (the downright square with the name terminal):

```
pip install opcua
```

Once installed we can use it in our code. Let's try to connect to our simulation server. Now create a new file with File > New. First of all, we need to tell Python the tools we want to use in this particular code, this is we need to import the libraries. We can do it by writing the next lines in the file:

```
from opcua import Client
from time import sleep
```

With these lines now we can use the tool Client from the library opcua and the tool sleep from the library time. Now let's connect to the simulation server with the following lines:

```
cliente = Client(URL_OPC)
cliente.connect()
root = cliente.get_root_node()
valor = root.get_child(['0:Objects', '5:Simulation',
'5:sinus']).get_value()
```

Substitute URL_OPC with the direction of the server. Our client just connected to the Simulation server, retrieved the value of the variable sinus, and saved it to the variable valour. We now have to show it, we can do this using print:

```
print(valor)
```

We want our client to ask for the value every certain amount of time. We can do this with a while loop like this:

```
while True:
    valor = root.get_child(['0:Objects', '5:Simulation',
'5:sinus']).get_value()
    print(valor)
    sleep(10)
```

Our code is now asking the server for the value, prints the value, does nothing for 10 seconds, and repeats infinitely. With this, we have successfully created our first OPCUA Client. In the next section, we will see how to store these values to create a database of registers.

2.3 DATA STORAGE WITH DATABASES OR CLOUD SERVERS

Although we already have a full OPC UA client that connects to the server and retrieves the values of the sensors in real-time and will keep asking forever, the data is only displayed on the screen and then lost forever. Many times this might be good enough but normally we'd like to store this data somewhere more permanent this is where databases come into play.

2.3.1 DATABASES BASICS

Although there are several types of databases, the most widespread kind are relational databases and specifically, SQL databases. SQL stands for Structured Query Language and is a standard that many providers follow. Some of the main SQL databases options are the following ones shown in the picture:

Figure 7. Some common SQL databases

Once again, databases creation and manipulation are abilities that take time to learn and master, and teaching this kind of knowledge is far beyond the scope of this document. Although the following explanation was written with the idea of being accessible to anyone, don't worry if you don't fully understand everything if you don't have previous experience with databases.

Even though any one of the databases managers presented in the picture above would be a good fit for our project, we will be using MySQL. Let's start downloading with this link:

https://dev.mysql.com/downloads/windows/installer/8.0.html

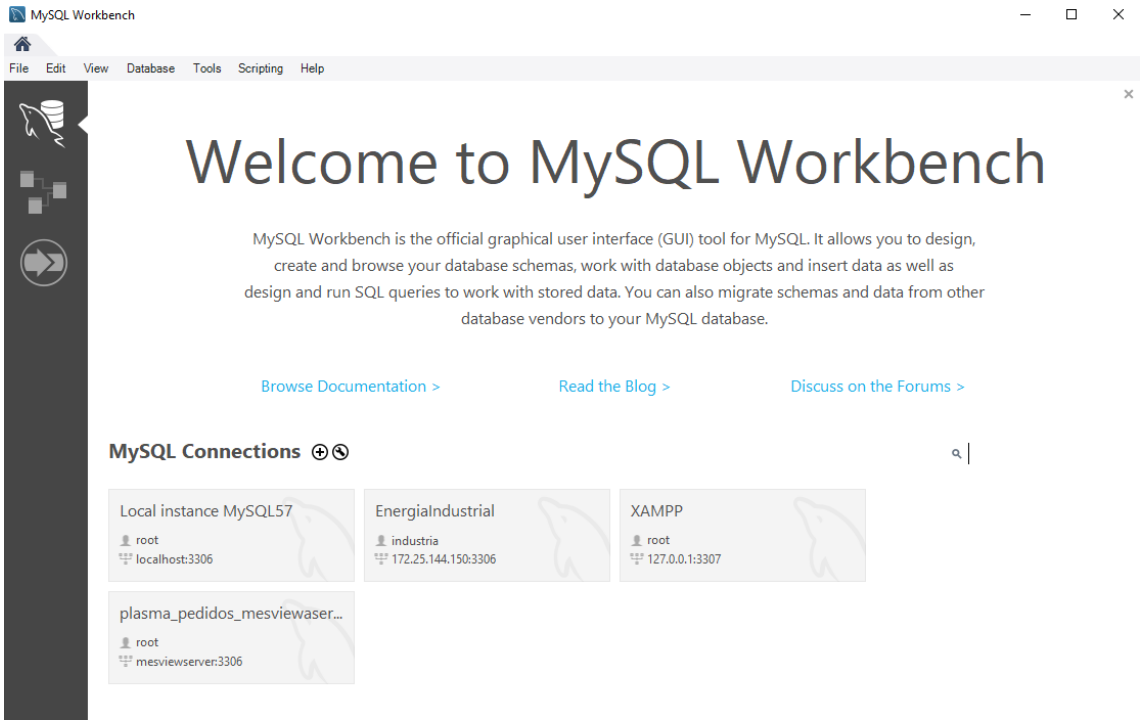
Once downloaded, let's install it. MySQL installer is actually a hub where you can install not only the database itself but also many other programs and complements that expand its functionality.

We only need to install two software:

- MySQL server – the database itself.
- MySQL workbench – A software that allows us to easily manipulate the database.

We can select only installing these by choosing custom installation. Then just follow all the instructions given by the installer.

After the database manager is installed, we can start creating our database. Let's start MySQL workbench and you should be seeing something like this:



But you will probably just have one connection let's click it and provide the credentials we just defined in the installation process. After that, we should be in the next window:

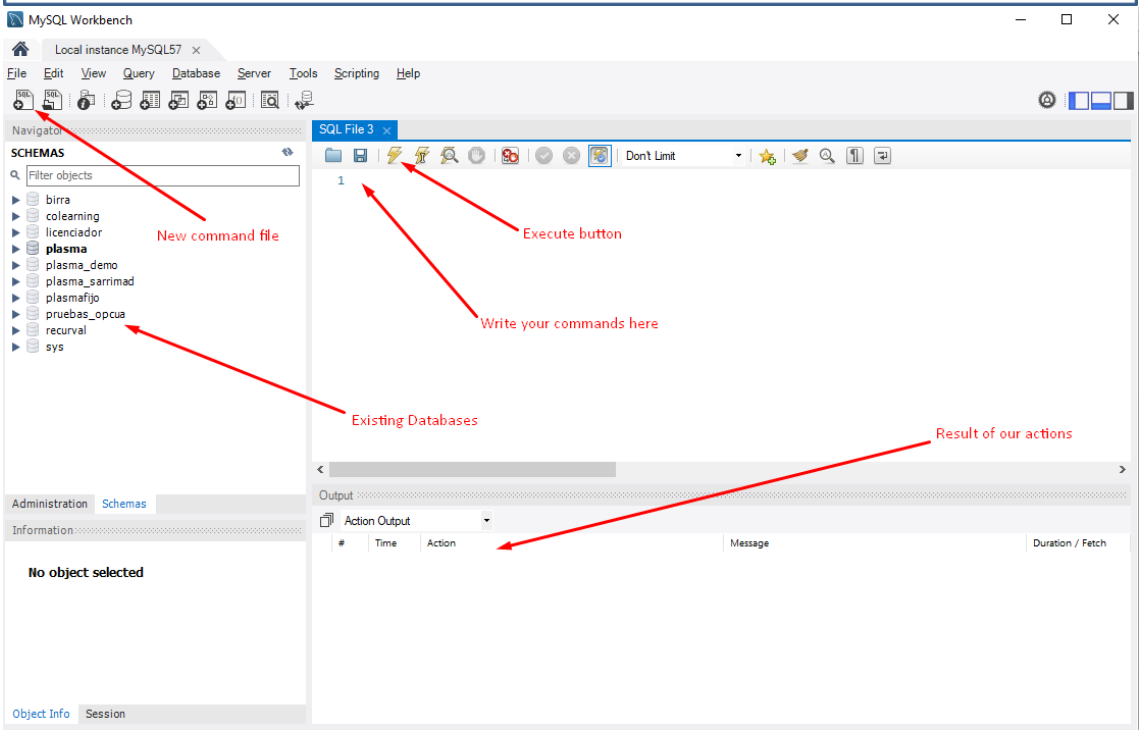


Figure 8. MySQL Workbench screen

Now let us create our first database, to do so, click the new command file button and write the following command:

```
create schema register;
```

Now if we right-click the existing databases field and then click update, we should be seeing the database we just created. If you double click it, its name should become bold pointing out that all the commands executed will affect this database and not others unless the opposite is explicitly stated.

SQL databases store the data in tables, and although with complex models we would have many tables and all the relations they have, for this example we will use a very simple model with only one table. In this table, we want to store all the data from the 3 simulated sensors we defined earlier. This means we have a table with at least 3 tables, we will also add the first column with a numeric identifier that uniquely identifies a row. The last column will be a timestamp of the exact moment the data was read. In short, we need to create the next table:

ID	Sensor1	Sensor2	Sensor3	Timestamp

To create it, we use the next SQL command:

```
CREATE TABLE sensor_data (
    id INT NOT NULL AUTO_INCREMENT,
    sensor1 FLOAT,
    sensor2 FLOAT,
    sensor3 FLOAT,
    ts DATETIME,
    PRIMARY KEY (id)
);
```

If you right-click the database and click update, now you should be seeing the new table created in the tables section.

2.3.2 SENDING FROM THE OPC UA SERVER TO THE DATABASE WITH PYTHON

We are almost done. We already have an OPC UA server simulating variable, a Python OPC UA client who reads the values of the sensors every 10 seconds, and a database with a table ready to store all the registers. The only remaining step is to tell Python to save the data to the database. We can do this by adding some lines to our python program.

First of all, we have to start by installing the Python-MySQL connection, we can do this from PIP, which we already installed before. We do this just like we did with the opcua library with the next command:

```
pip install mysql-connector
```

Once installed, we can import it to our program with:

```
import mysql.connector
```

Now we must define the credentials we need to connect to our database. Let us do this by adding the next code lines to our program:

```
mydb = mysql.connector.connect(
  host="127.0.0.1",
  user="root",
  password="YOUR PASSWORD HERE",
  database="register"
)
```

Now we can use this data to create a working connection with:

```
mycursor = mydb.cursor()
```

With this set, we can already start sending our data to our database. We now will be reading all the sensors from the server with these 3 lines:

```
sensor1 = root.get_child(['0:Objects', '5:Simulation', '5:sinus']).get_value()
sensor2 = root.get_child(['0:Objects', '5:Simulation', '5:Triangle']).get_value()
sensor3 = root.get_child(['0:Objects', '5:Simulation', '5:Square']).get_value()
```

And then we send them to the database with these lines:

```
sql="INSERT INTO sensor_data (sensor1, sensor2, sensor3, ts) VALUES (%s, %s, %s, %s)"
val = (sensor1, sensor2, sensor3, str(datetime.now()))
mycursor.execute(sql, val)
mydb.commit()
```

Note that `str(datetime.now())` is returning the current date and time. In order to being able to use it, we have to first import datetime from the datetime library like this:

```
from datetime import datetime
```

If we execute it, the data should start flowing to the table created in the database. We can check this by executing this command in the database:

```
SELECT * FROM register.sensor_data;
```

With this, we get a result like the next one:

id	sensor1	sensor2	sensor3	ts
1	1.17557	0.8	2	2021-10-22 12:18:43
2	0.415823	0.266667	-2	2021-10-22 12:18:45
3	-0.415823	-0.266667	-2	2021-10-22 12:18:47
4	-1.17557	-0.8	-2	2021-10-22 12:18:49
5	-1.73205	-1.33333	2	2021-10-22 12:18:51
6	-1.98904	-1.86667	2	2021-10-22 12:18:53
7	-1.90211	-1.6	-2	2021-10-22 12:18:55
8	-1.48629	-1.06667	2	2021-10-22 12:18:57
9	-0.813473	-0.533333	2	2021-10-22 12:18:59
10	-0.0000000459469	-0.0000000292507	2	2021-10-22 12:19:01

Fulfilling this way our target and the objectives of this chapter. For reference, this I the whole Python code that resulted from the development where some extra code was added for error handling:

```
from opcua import Client
from time import sleep
from datetime import datetime
import mysql.connector

FRECUENCIA = 10 #segundos
URL_OPC = 'opc.tcp://PORTATIL-AUXROBOT:53530/OPCUA/SimulationServer'

mydb = mysql.connector.connect(
  host="127.0.0.1",
  user="root",
  password="yourpassword",
  database="register"
)

mycursor = mydb.cursor()

try:

    cliente = Client(URL_OPC)
    cliente.connect()
    root = cliente.get_root_node()
    while True:

        #Read OPC UA variables
        sensor1 = root.get_child(['0:Objects', '5:Simulation',
'5:sinus']).get_value()
        sensor2 = root.get_child(['0:Objects', '5:Simulation',
'5:Triangle']).get_value()
        sensor3 = root.get_child(['0:Objects', '5:Simulation',
'5:Square']).get_value()


        #write in DB
        sql = "INSERT INTO sensor_data (sensor1, sensor2, sensor3, ts)
VALUES (%s, %s, %s, %s)"
        val = (sensor1, sensor2, sensor3, str(datetime.now()))
        mycursor.execute(sql, val)
        mydb.commit()

        sleep(FRECUENCIA)
except Exception as e:
    print(str(e))
finally:
    cliente.disconnect()
```