CRUD Operation on WordPress Database Using C# SQL Client

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Professor, Institute of Management & Commerce, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area of the Paper: Computer Science.
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed In: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.10162719
Google Scholar Citation: IJCSBE

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S. (2023). CRUD Operation on WordPress Database Using C# SQL Client. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, *7*(4), 138-149. DOI: https://doi.org/10.5281/zenodo.10162719

International Journal of Case Studies in Business, IT and Education (IJCSBE)
A Refereed International Journal of Srinivas University, India.

Crossref DOI: https://doi.org/10.47992/IJCSBE.2581.6942.0313

Paper Submission: 15/10/2023
Paper Publication: 23/11/2023

International Journal of Case Studies in Business, IT, and Education (IJCSBE), ISSN: 2581-6942, Vol. 7, No. 4, November 2023

SRINIVAS PUBLICATION

CRUD Operation on WordPress Database Using C# SQL Client

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Professor, Institute of Management & Commerce, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *Sometimes, the researcher stores sensor data inside the cloud to access the data globally. The widespread cloud infrastructures are AWS, Azure, Google, IBM, etc. This type of Cloud server is not free. We need to pay through subscription. Here, we demonstrate how to manage our sensor data using our existing WordPress Website database. The primary example of CRUD operation on the WordPress Database is provided here. The database class is portable. The researcher can easily integrate more functionality. Using the C# language, we created the "WordPressDB.cs" library with the help of an inbuilt SQL client module. On top of that, we developed a graphical user interface for easy operation. We use Flywheel to install WordPress websites locally for safe operation. Once the development is over, we can deploy online without any issues. The completed project code is available to download for quick integration.*

Design/Methodology/Approach: *To keep our research work free, we installed Flywheel Local software suitable for local WordPress Website installation. We installed a WordPress Website using the Flywheel. Once the Website is fully installed, we run the site from the Flywheel local interface. Now, we open our C# projects. We need to provide the Database credentials to connect with the WordPress Database. Once the connection is successful, we can create a table read, update, and delete the record.*

Findings/Result: *We demonstrated how to keep our sensor data using our existing WordPress database. We tested the provided approach, a good and efficient way to control and manage the sensor data. Our communication stack is as tiny as possible, so it is fast. We can keep our real-time data inside this Database with minimum delay.*

Originality/Value: *Several procedures are available to keep the sensor's data. Using this procedure, we can maintain the data at the least cost possible. We are paying for our website to run. We are getting online cloud storage to keep our sensor data without extra cost. The Website runs around the clock, so our research data is available anytime and anywhere. This approach differs slightly from the traditional approach, like purchasing subscription cloud hosting infrastructure. We can also run the IoT activities using this approach. To exchange the dynamic data between two nodes is also effective without buying a static IP.*

Paper Type: *Experimental-based Research.*

Keywords: CRUD Operation, WordPress Database, C# SQL Client, Basic Database operation Using C#, CRUD demonstration using Dot Net framework.

1. INTRODUCTION :

We research in various fields. When we work with sensor data, we need to store it. Nowadays. We do this task using IoT. Most of the tech giant has their IoT fracture. Like Amazon, Azure is the popular name in this domain. Using these types of infrastructure, we can manage sensor data easily or with minimal effects. However, the problem is the resources needed to pay to store the data on those types of cloud servers.

Now, we have found a way to lower the cost of web servers to store the data. WordPress Website is around 40%. To keep the data inside the WordPress database is one of the choices. There are several advantages to storing data inside the WordPress Website database. WordPress has several built-in functions to make the task easy for us. From post to JSON or vice versa is entirely handled by WordPress. There is no need to worry about receiving or transmitting content to the client or receiving the data from the client and saving it to the server.

Here, we will experiment using a C# SQL client. We developed an application in C# inside the community edition of Visual Studio 2022 from Microsoft Corporation. In the GUI, we have several buttons for easy operation. Through the SQL client module, we communicate with the WordPress website database. For our experiment, we are using a local WordPress website. Because if we work on the actual website server, any error code may stop the Website. Once our development is completed, we can deploy it inside the existing website.

The paper is organized as follows: Section 2 provides an overview of related work already done on IoT. Section 3 discusses the objective of the research work. Section 4 highlights the methodology we used for the research work. In section 5, we do the actual experiment. This section describes the procedure with a couple of required screenshots. Section 6 provides recommendations for reading or watching videos to understand the research topic better. Finally, Section 7 concludes the paper and offers future research directions.

2. RELATED WORKS :

In their paper, Kornienko D. et al. demonstrate the Single Page Application (SPA) architecture in the context of secure web services. It explores the development of SPAs for automating user information accounting. The study provides insights into securing web services using SPAs [1]. Luo et al. investigate the low-code effect from a practitioner's perspective [2]. Walker, C. et. Al., in their paper, focuses on a personal data lake with data gravity pull [3]. Resceanu et al. present a framework for websites targeting international audiences [4]. Al Mahruqi, R. S. et al. introduce a semi-automated framework for migrating web applications from SQL to document-oriented NoSQL databases in their paper [5]. Lemos et al. conducted a comprehensive survey of methods and tools related to web service composition [6]. Říčan presents an application for synchronizing events between VersionOne and ALM. This doctoral dissertation explores integrating software development and application lifecycle management [7]. Naményi's doctoral dissertation examines technology trends in web development and their impact on businesses [8]. These papers cover various topics in software development, web services, and data management, providing valuable insights into multiple aspects of the field. Researchers and practitioners can benefit from the knowledge presented in these papers to advance their understanding and practices in these areas.

3. OBJECTIVES :

This research aims to provide some reference information to the researcher to use the Website WordPress database to store the sensor or various research data. This approach may be as cheap as possible from other available procedures. We provide a step-by-step practical approach to execute the demonstrated solution to minimize the learning curve. The Communication module is portable to fit into our custom project. Our researcher must add the communication class and call the function according to the project's communication flow. As the first appearance, maybe some debug phase. However, no need to worry. The whole process is demonstrated in the local environment. Once fine, it can be deployable inside the online cloud server.

4. APPROACH AND METHODOLOGY :

Figure 1 depicts the block diagram of the project. Now, we describe the modules one by one.

1) **UI main Thread**: The central part of the architecture is C# Application's main thread. It coordinates among blocks or modules like user interaction, communication modules, etc. When the application starts running, it creates a "**WordPressDB.cs**" instance, and all GUI elements are initialized. Once the application is open, we must first connect with the Database. Once the Database connects successfully, then we can do any operation on the Database. In every function, we add a Database connection call for intelligent operation. If we forget to press the connection button first and try to

read data from the Database, the function first checks whether the connection is present. If not connected, it will connect with the Database first and then do the requested operation.

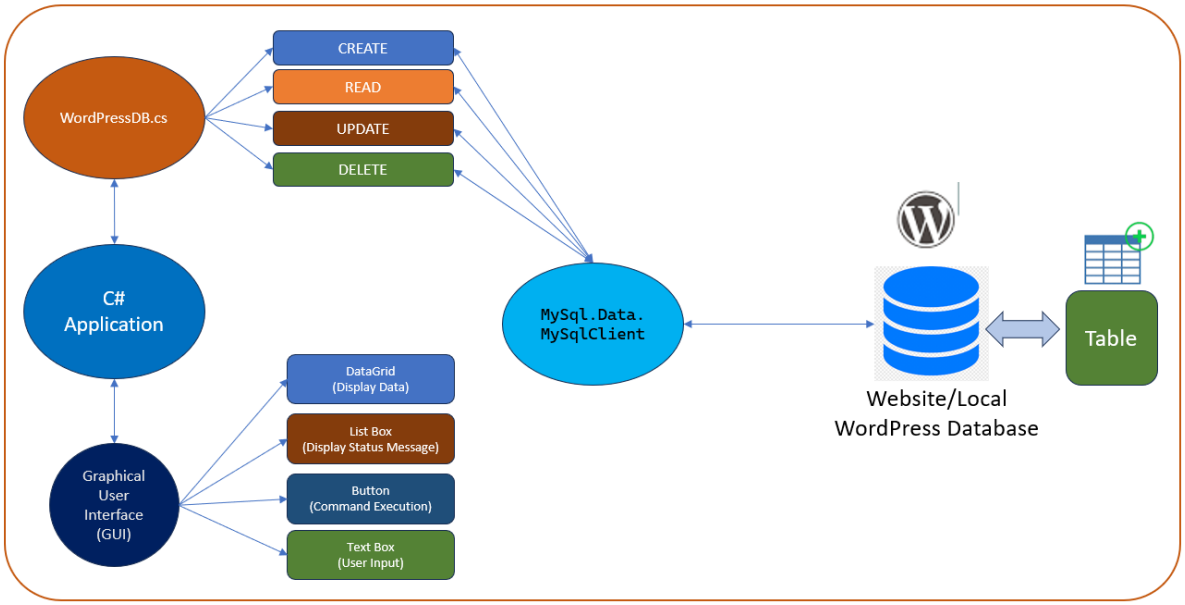

Fig. 1: Project block diagram

2) **Graphical User Interface (GUI)**: We added several GUI elements for easy operation. We added list boxes to display the status messages. We added a data grid to display the fetched data from the Database. Button control is used to send the command for execution. Text boxes were added to enter the user input data and server credentials.
3) **WordpressDB.cs**: This is the class that handles all communication with the Database. We created this class portable. It can be imported into any project. The Database credential will vary from project to project. This class does all communication with the help of the C# SQL client module. This module handles all protocols and formalities for us. We created separate functions for every task with SQL command strings as parameters. This can be helpful for custom queries.
4) **MySql.Data.MySqlClient**: This module is responsible for communicating with the Database. It hides all protocol details for communication. When "WordpressDB.cs" initiates, The SQL client instance is created. Moreover, one connection object is also created, which is needed for every command with the Database.
5) **WordPress Website Database**: The Database installs automatically when the Flywheel local application installs the WordPress. The available Database version can be found on the Local Dashboard. If some SQL command catches an error, that might be the issue of the SQL command not supporting that version.
6) **Database Table**: To Save the data inside the Database, we must create the table. The table can be made by clicking the "**Open Adminer**" button from the Local dashboard and also through the program.

5. EXPERIMENT :

Instead of working on a website, we will work on a local one. So, we need to install WordPress in our local system. The below steps are required to set up WordPress locally.

Install WordPress Locally:

1) Open ***https://localwp.com/.*** Click on the "DOWNLOAD FOR FREE" button.
2) Select platform/operating system. Fill up the rest of the form. Click on "GET IT NOW!". It will take a couple of minutes. Depends on internet speed.
3) The downloaded file looks like "**local-7.2.1-windows**" after the download. The version may vary at the download time. Double-click on the downloaded file.

4) The "**Choose Installation Options**" window will appear. Select "**Anyone who uses this computer (all users)**'. Click next.
5) Under the "**User Account Control**," click on the "**Yes**" button.
6) "**Choose Installation Location**" windows, select the destination folder, or keep the default path.
7) Click on the "**Install**" button.
8) Once installation is complete, click the "**Finish**" button. The program shortcut icon is available on the desktop. Click on it. The startup windows look like Figure 2.

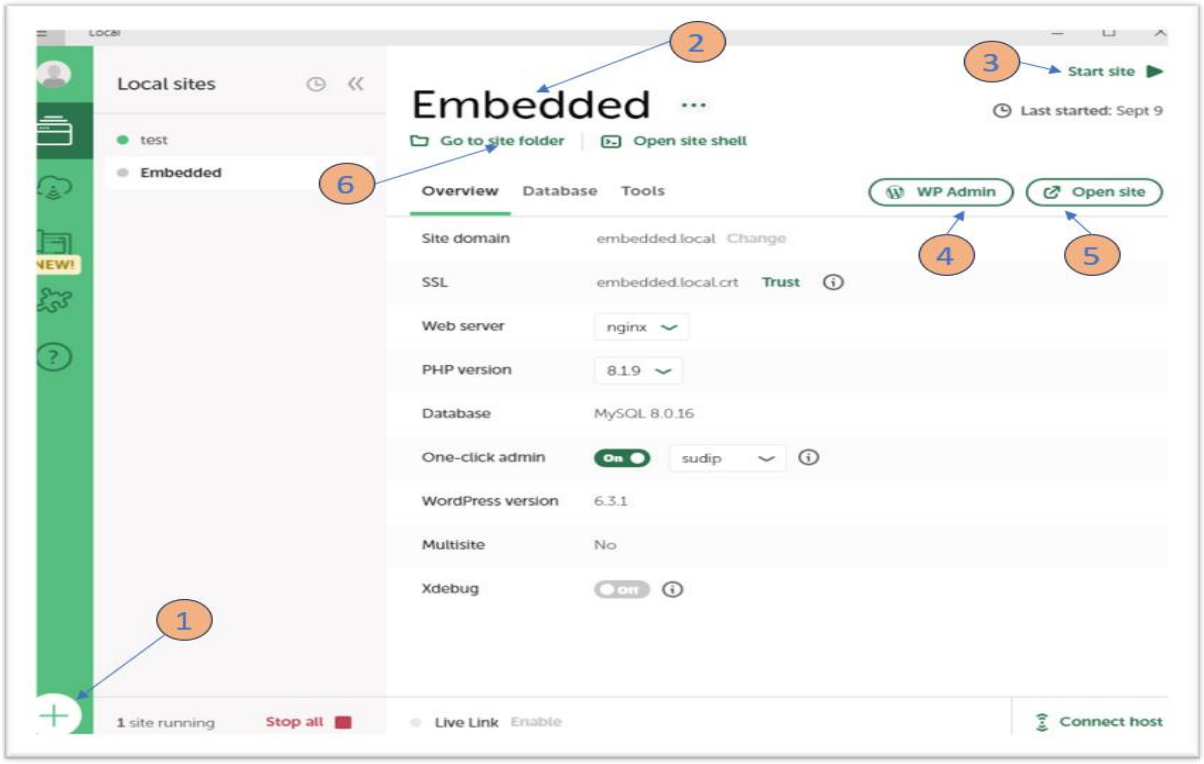

Fig. 2: The Flywheel Local interface

9) Now, we will see what is inside the window. Describe the marker point below:
    1) The "+" sign button is used to create a new website.
    2) It shows the name of the Website.
    3) This button is used to start the Website. When this button Shows "**start site,**" means clicking on it, the selected website will open. When the Website starts, it shows "**Stop site**" in red, which means clicking on it will stop the selected website.
    4) It is the most important button. It is used to open the WordPress admin console. Several configuration-related tasks can be done using this button.
    5) This button opens the Website in a new browser window.
    6) This button is used to open the website installation folder. These are the essential buttons we used for WordPress.

Create New Website:

1) Click on the "**Add Local site**" button.
2) Keep selecting "**Create a new site**" and press the "**Continue**" button.
3) Add the Site name and click the "**Continue**" button.
4) Select the "**Preferred**" button and click the "**Continue**" button.
5) Type "**WordPress username**" and "**WordPress password**" and click "**Add a site**."
6) The User Account Control window will come. Click "**Yes**". Then, WordPress will be installed within a few minutes, and "**Open site**" will be visible.

7) Click on the "**Open Site**" button. The brand-new Website will be visible in a new web browser window.
8) Our website is created now. Next, we will create Our Database

CRUD Operation from Adminer Interface:

1) Open the Local application using the Desktop shortcut icon.
2) Select the Website by clicking on the website name. If the site is stopped, click "**Start site**" to start the Website.
3) On the top-middle, click on the "**Database**" Tab. Click on "**Open Adminer**" to open the Adminer interface.
4) From the left side, click on "**Create table**". As Figure 3, configure the table.

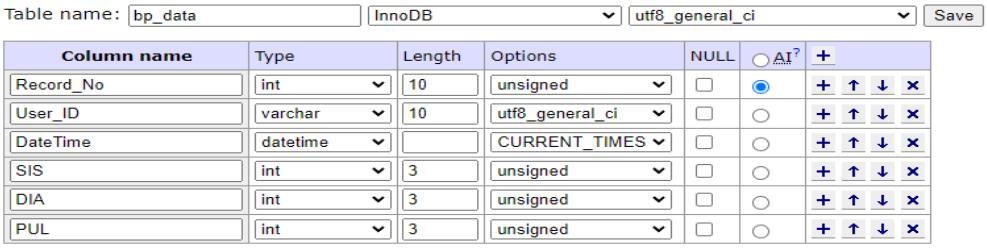

Fig. 3: Database Table Elements properties.

5) After entering the above details, click on the save button.
6) Click on the table name "**bp_data**" from the left side. From the top right, click on the "**New item**" field. We can add a sample to the table. Record_No. is auto increment. There is no need to add value. **User_ID** is the data for that specific user. The records are identified by user ID. When we enter the BP data, we must take care of the user ID, or it can mesh with another user. We can solve this problem from the application front end. When we log in to the application, it can be filled up automatically. So, no need to worry about user ID. As of now, we need to take care of ourselves. To enter the **DateTime**, we can manually add or select "**now**" from the dropdown. For manual date time, the format is **YYYY-MM-DD hh:mm: ss**. Then, measure the BP using a BP monitor and enter the SIS, DIA, and PUL values. Now click the "Save" button.
7) To view our inserted data, click "**select**" on the left side of the table name "**bp_data.**"
8) We observe our inserted data displayed in the table on the right side.
9) We created the table, added data, and displayed means to read the data. We can edit any record using the graphical button.
10) To Edit the record, click "**edit**" on particular data from the left side to edit, change, and save the changes.
11) To Delete any records, check those to delete and click the "**Delete**" button.
12) Sometimes, we need to clone records for our testing purposes. Select one or multiple forms and click " Clone ".

CRUD using SQL command from admin panel:

1) From the left side, click on "**SQL command**." One blank text box will appear.
2) Figure 4 is the example to read for a particular user ID='W001'

```
SELECT * FROM bp_data where USER_ID='W001'
```

Record_No	User_ID	DateTime	SIS	DIA	PUL
35	W001	2023-10-02 17:39:00	131	84	80
36	W001	2023-10-02 17:42:40	120	79	80

2 rows (0.000 s) Edit, Explain, Export

```
SELECT * FROM bp_data where USER_ID='W001'
```

Fig. 4: Read data on user ID

3) Figure 5 depicts the update of the records.

```
UPDATE bp_data SET SIS= '122' WHERE Record_No=35
```

Query executed OK, 1 row affected. (0.004 s) Edit

```
SELECT * FROM bp_data where USER_ID='W001'
```

Record_No	User_ID	DateTime	SIS	DIA	PUL
35	W001	2023-10-02 19:15:36	122	84	80
36	W001	2023-10-02 17:42:40	120	79	80

2 rows (0.000 s) Edit, Explain, Export

```
UPDATE bp_data SET SIS= '122' WHERE Record_No=35;
SELECT * FROM bp_data where USER_ID='W001'
```

Fig. 5: Update/modify record upon condition.

```
DELETE FROM bp_data WHERE Record_No=35
```

Query executed OK, 1 row affected. (0.004 s) Edit

```
SELECT * FROM bp_data where USER_ID='W001'
```

Record_No	User_ID	DateTime	SIS	DIA	PUL
36	W001	2023-10-02 17:42:40	120	79	80

1 row (0.000 s) Edit, Explain, Export

```
DELETE FROM bp_data WHERE Record_No=35;
SELECT * FROM bp_data where USER_ID='W001'
```

Fig. 6: Delete record upon condition.

4) Figure 6 is an example of deleting the records.

C# SQL Client Implemented function details:

```
/// @brief This function is used to connect with database
/// @param server=server address
/// @param port= server port
/// @param Database= Database name
/// @param Uid= User ID
/// @param Pwd= Password
/// @return void
/// </summary>
references
public void Connect(string server, string port,string Database, String Uid,String Pwd)
{
    string connectionString = "Server="+server+";Port="+port+";Database="+Database+";Uid="+Uid+";Pwd="+Pwd+";";
    connection = new MySqlConnection(connectionString);
    connection.Open();
    if (connection.State==ConnectionState.Open)
    {
        Connected= true;
    }
    else
    {
        Connected= false;
    }
```

Fig. 7: The function to connect with database.

```
/// <summary>
/// @brief This function is used to Disconnect from the database
/// @param void
/// @return void
/// </summary>
1 reference
public void Disconnect()
{
    if(Connected)
    {
        connection.Close();
        Connected = false;
    }
}
```

Fig. 8: The function to Disconnect from database.

Figure 7 depicts the function to connect with the Database. Five parameters are required to execute the function: server address, server port, Database name, user ID, and password. If the connection is successful, the connected status is set to true.

```
private void btn_create_table_click(object sender, EventArgs e)
{
    if(!db.Connected)
    {
        db.Connect(txt_server.Text, txt_port.Text, txt_Database.Text, txt_Uid.Text, txt_password.Text);
        msg.push("Connected With Database OK");
    }
    string createTableQuery =
        "CREATE TABLE IF NOT EXISTS" +
        " tbl_BP_Data " +
        "(" +
            "Record_No INT AUTO_INCREMENT PRIMARY KEY, " +
            "User_ID VARCHAR(10) NOT NULL, " +
            "DateTime datetime NOT NULL, " +
            "SIS INT(3) unsigned NOT NULL, " +
            "DIA int(3) unsigned NOT NULL, " +
            "PUL int(3) unsigned NOT NULL" +
        ")";
     db.Create_Table(createTableQuery);
}
```

Fig. 9: The function to create table inside the database.

Figure 8 shows that the function is used to disconnect from the server. The parameter and return type are void.

```
public bool Create_Table(String createTableQuery)
{
    try
    {
        MySqlCommand command = new MySqlCommand(createTableQuery, connection);
        command.ExecuteNonQuery();
    }
    catch (Exception)
    {
        //throw;
        return false;
    }
    return true;
}
```

Fig. 10: Create table function inside the WordPressDB.cs class.

Figure 9 shows the function of the “Create Table”. We keep this function inside a button-press event. The create table query string can be customized for different table structures. This is an example of a Blood pressure monitor data table. When we press the button, it checks the database connection. If a connection is absent, it connects first and then processes next. Once the query string is prepared, it calls the function inside the WordPressDB.cs module depicted in Figure 10.

Figure 11 depicts the delete function. These lines can be placed inside the button “Delete.” This function calls another function, which is depicted in Figure 12.

```
private void btn_delete_table_Click(object sender, EventArgs e)
{
    if (!db.Connected)
    {
        db.Connect(txt_server.Text, txt_port.Text, txt_Database.Text, txt_Uid.Text, txt_password.Text);
        msg.push("Connected With Database OK");
    }

    db.Delete_Table("tbl_bp_data");
}
```

Fig. 11: Delete table function inside the button press event.

```
public bool Delete_Table(String Table_Name)
{
    string cmd= "DROP TABLE IF EXISTS "+Table_Name;
    try
    {
        MySqlCommand command = new MySqlCommand(cmd, connection);
        command.ExecuteNonQuery();
    }
    catch (Exception)
    {
        //throw;
        return false;
    }
    return true;
}
```

Fig. 12: Delete table function inside the WordPressDB.cs class.

```
private void txt_insert_Click(object sender, EventArgs e)
{
    if (!db.Connected)
    {
        db.Connect(txt_server.Text, txt_port.Text, txt_Database.Text, txt_Uid.Text, txt_password.Text);
        msg.push("Connected With Database OK");
    }

    var List<KeyValuePair<string, string>> list = new List<KeyValuePair<string, string>>();

    list.Add(new KeyValuePair<string, string>("User_ID", txt_user_id.Text.ToString()));
    list.Add(new KeyValuePair<string, string>("DateTime", "2023-10-13 12:02:00"));
    list.Add(new KeyValuePair<string, string>("SIS", txt_sys.Text.ToString()));
    list.Add(new KeyValuePair<string, string>("DIA", txt_dia.Text.ToString()));
    list.Add(new KeyValuePair<string, string>("PUL", txt_pul.Text.ToString()));

    db.Insert_Data("tbl_bp_data", list);
}
```

Fig. 13: Insert function.

```
/// <summary>
/// @brief This function is used to Insert Data into Table
/// @param Query String to insert data into table
/// @return true if inserted ok, else false
/// </summary>
1 reference
public bool Insert_Data(String table_name, List<KeyValuePair<string, string>> data)
{
    string s1 = "(";
    string s2 = "VALUES (";

    foreach (KeyValuePair<string, string> k in data)
    {
        s1+=k.Key+",";
        s2=s2+"@"+k.Key+",";
    }
    s1=s1.Substring(0, s1.Length-1); s1+=")";
    s2=s2.Substring(0, s2.Length-1); s2+=")";

    string insertQuery = "INSERT INTO " + table_name +" " + s1 +" " + s2;
    MySqlCommand command = new MySqlCommand(insertQuery, connection);

    foreach (KeyValuePair<string, string> k in data)
    {
        command.Parameters.AddWithValue("@"+k.Key, k.Value);
    }
    int rowsAffected = command.ExecuteNonQuery();

    if (rowsAffected > 0) return true;
    else                  return false;
}
```

Fig. 14: Insert function inside the WordPressDB.cs class.

Figure 13 depicts the insert data function into the database table. This function needs to reside inside the button name "Insert." It checks the connection. If a connection is not present, connect first. Then, it creates a key-value pair list and passes as a parameter calling the function "Insert_Data" depicted in Figure 14, which resides inside the WordPressDB.cs class. The nested function parses the list, creates a query string, and executes an "ExecuteNonQuery()" function. If any row is affected, return true; else, return false.

```
private void dg_display_PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        int c_row = dg_display.CurrentRow.Index-1;

        string Record_ID = dg_display[0, c_row].Value.ToString();

        string User_ID = dg_display[1, c_row].Value.ToString();

        string SIS = dg_display[3, c_row].Value.ToString();
        string DIA = dg_display[4, c_row].Value.ToString();
        string PUL = dg_display[5, c_row].Value.ToString();

        var List<KeyValuePair<string, string>> list = new List<KeyValuePair<string, string>>();

        list.Add(new KeyValuePair<string, string>("User_ID", User_ID));

        DateTime myDateTime = DateTime.Now;
        string sqlFormattedDate = myDateTime.ToString("yyyy-MM-dd HH:mm:ss");
        list.Add(new KeyValuePair<string, string>("DateTime", sqlFormattedDate));
        list.Add(new KeyValuePair<string, string>("SIS", SIS));
        list.Add(new KeyValuePair<string, string>("DIA", DIA));
        list.Add(new KeyValuePair<string, string>("PUL", PUL));

        db.Update_Data("tbl_bp_data", list, "Record_No="+Record_ID);
        Display_Data();
    }
}
```

Fig. 15: Record Update function

```
/// <summary>
/// @brief This function is used to Update Data inside the Table
/// @param table_name Table Name
/// @param data=command string
/// @return true if update ok, else false
/// </summary>
1 reference
public bool Update_Data(String table_name, List<KeyValuePair<string, string>> data, string condition)
{
    string s1="";
    foreach (KeyValuePair<string, string> k in data)
    {
        s1+=k.Key+" =@"+k.Key+",";
    }
    s1=s1.Substring(0, s1.Length-1);

    string insertQuery = "UPDATE " + table_name +" SET " + s1 + " WHERE "+condition;
    MySqlCommand command = new MySqlCommand(insertQuery, connection);

    foreach (KeyValuePair<string, string> k in data)
    {
        command.Parameters.AddWithValue("@"+k.Key, k.Value);
    }
    int rowsAffected = command.ExecuteNonQuery();

    if (rowsAffected > 0) return true;
    else return false;
}
```

Fig. 16: Record Update function inside the WordPressDB.cs class

```
1 reference
private void button5_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show("Do you want to Delete Record?", "Confirmation",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (result == DialogResult.Yes)
    {
        int c_row = dg_display.CurrentRow.Index;
        string Record_ID = dg_display[0, c_row].Value.ToString();

        if( db.Delete_Data("tbl_bp_data", "Record_No="+Record_ID))
        {
            MessageBox.Show("Record Sucessfully Deleted");
            Display_Data();
        }
        else
        {
            MessageBox.Show("Error!!!!Not able to Delete Records");
        }
    }
}
```

Fig. 17: Record Delete function.

Figure 15 depicts the update record function. This function calls another nested function, which is depicted in Figure 16. to update the record, we need the parameters and conditions. In the general scenario, the record update depends on the record ID. It is unique, and auto incremented at record insert or addition time.

```
/// <summary>
/// @brief This function is used to Delete Row from Table
/// @param table_name Table Name
/// @param condition=condition to delete row
/// @return true if delete ok, else false
/// </summary>
1 reference
public bool Delete_Data(String table_name, string condition)
{
    string insertQuery = "DELETE FROM " + table_name +" WHERE "+condition;
    MySqlCommand command = new MySqlCommand(insertQuery, connection);
    int rowsAffected = command.ExecuteNonQuery();
    if (rowsAffected > 0) return true;
    else return false;
}
```

Fig. 18: Record Delete function inside the WordPressDB.cs class.

Figure 17 and Figure 18 are responsible for deleting the record function. The record is deleted on the record ID.

6. RECOMMENDATIONS :

- The project is available: https://github.com/sudipchakraborty/CRUD-Operation-On-WordPress-Server-Using-C-Sharp.git.
- We highly recommend adding an error-handling code if we want to move the project into production.

7. CONCLUSION :

We observe how to create our data table inside the WordPress Website Database using a step-by-step procedure. We demonstrate the CRUD operation using a simple example. The researcher trying to integrate CRUD operation to store their Data inside the online Database can get valuable references from this work. Using Flywheel Local, the development can be done free of charge. once the development is finalized, we can deploy it inside the online Database. We kept the example simple so new researchers or integrators could deploy it quickly.

REFERENCES :

[1] Kornienko, D. V., Mishina, S. V., & Melnikov, M. O. (2021, November). The Single Page Application architecture when developing secure Web services. In Journal of Physics: Conference Series (Vol. 2091, No. 1, p. 012065). IOP Publishing. Google Scholar

[2] Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021, October). Characteristics and challenges of low-code development: the practitioners' perspective. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1-11). Google Scholar↗

[3] Walker, C., & Alrehamy, H. (2015, August). Personal data lake with data gravity pull. In 2015 IEEE Fifth International Conference on Big Data and Cloud Computing (pp. 160-167). IEEE. Google Scholar↗

[4] Resceanu, I. C., Resceanu, C. F., & Simionescu, S. M. (2013, October). Developing a framework for websites with international audiences. In 2013 17th International Conference on System Theory, Control and Computing (ICSTCC) (pp. 453-458). IEEE. Google Scholar↗

[5] Al Mahruqi, R. S., Alalfi, M. H., & Dean, T. R. (2019, November). A semi-automated framework for migrating web applications from SQL to document-oriented NoSQL database. In CASCON (pp. 44-53). Google Scholar↗

[6] Lemos, A. L., Daniel, F., & Benatallah, B. (2015). Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR), 48(3), 1-41. Google Scholar↗

[7] Říčan, M. (2015). Application for synchronization of events between VersionOne and ALM (Doctoral dissertation, Technická Univerzita v Liberci). Google Scholar↗

[8] Naményi, D. (2016). The technology trends in web development and their effects on businesses (Doctoral dissertation, BCE Gazdálkodástudományi Kar). Google Scholar↗

****