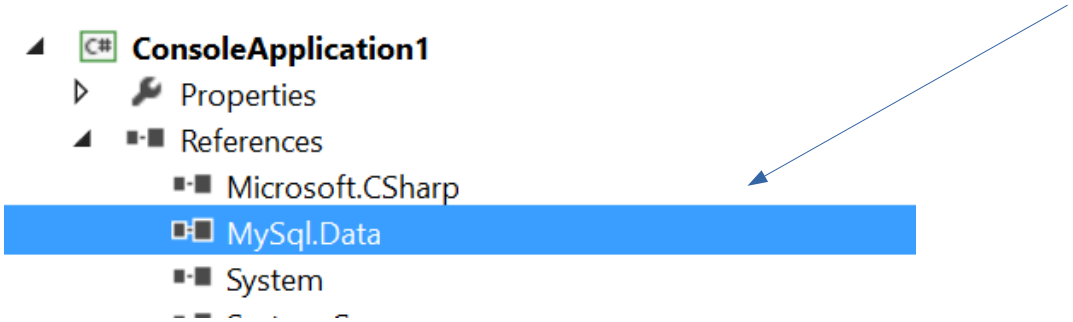
PERSISTENCIA EN MYSQL CON C#

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Referenciando Al Ensamblado Pata Conectar A Mysql

- El paso a seguir, es agregar una referencia al ensamblado o librería con la definición de clases que me permitirán usar MySQL a nuestro proyecto.
- Para lograrlo, agregaremos la librería MySql.Data.dll, con dichas definiciones.



- Una vez realizado, como de costumnbre, deberemos incluir el espacio de nombre con las definiciones para usar las clases realizando...

```
using MySql.Data.MySqlClient;
```

- Dispones de una referencia al API de este namespace en la siguiente URL https://dev.mysql.com/doc/dev/connector-net/6.10/html/N_MySql_Data_MySqlClient.htm

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Creando Un String De Conexión A MySQL

- Necesitaremos crear un string de conexión con los datos necesarios para conectarse a la BD, lógicamente esos datos deberán ser los mismos que pusimos en el momento de crear la BD.
- Será necesarios nombrarlos con un alias preestablecido. La manera correcta es la siguiente:

```
Server="";Port="";DataBase="";Uid="";Pwd="";
```

- Dentro de las comillas va la respectiva información, por ejemplo…

```
Server=127.0.0.1;Port=3306;DataBase=prueba;Uid=root;Pwd=1234;
```

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Crear Una Conexión - I

- Para abrir la conexión deberemos de crear un objeto de tipo MySqlConnection, asignar a su propiedad ConnectionString el string de conexión como hemos visto en el punto anterior y proceder a llamar al método Open.

```
string cConexion = "server=127.0.0.1;uid=root;pwd=;database=pruebas;"
MySqlConnection c = null;
try {
     c = new MySqlConnection(cConexion)
     c.Open();
     // Aquí realizaremos operaciones con la conexión.
}
catch (MySqlException e) {
     Console.WriteLine(e.Message);
}
finally {
     if (c != null && c.State == ConnectionState.Open)    {
          c.Close();
          c = null;
     }
}
```

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Crear Una Conexión - II

Puesto que MySqlConnection hereda de IDIsposable, podremos reescribir el código anterior de conexión de la siguiente forma simplificada...

```
string cConexion = "server=127.0.0.1;uid=root;pwd=;database=pruebas;"
using(MySqlConnection c = new MySqlConnection(cConexion)) {
     c.Open();
     // Aquí realizaremos operaciones con la conexión.
}
```

- No deberemos abrir más de una conexión a la vez, ya que es una operación costosa temporalmente y podremos saturar a MySQL y rechazarnos la misma si tenemos muchas abiertas.
- Si es necesario, nos guardaremos o pasaremos elobjeto con la conexión a aquellos métodos que lo necesiten.

- Si vas a realizar diferentes consultas de modificación seguidas como insert, update o delete. Es importante hacerlas usando la misma conexión.

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Ejecutar Comandos Sql - I

- Una de las cosas que podemos hacer es ejecutar sentencias SQL de cualquier tipo, tanto para crear tablas, como para eliminarlas o modificar datos y como no también para consultar o modificar el contenido de las tablas.
- Para poder hacer todo esto tendremos que crearnos un objeto del tipo MySqlCommand al que le asignaremos la conexión, indicaremos la sentencia sql en la propiedad commandText y la ejecutaremos.

```
…
    using(MySqlCommand cmd = new MySqlCommand()
    {
        Connection = c,
        CommandType = System.Data.CommandType.Text,
        CommandText = "INSERT INTO TABLA ... ;"
    })
    {
        cmd.ExecuteNonQuery();
    }
…
```

- Como se trata de una consulta de inserción llamaremos a ExecuteNonQuery.

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Ejecutar Comandos Sql - II

- También podremos obtener vistas mediante consultas select y recorrerlas con ExecuteReader.

```
MySqlCommand cmd = new MySqlCommand("SELECT * FROM tabla;", c);
MySqlDataReader r = cmd.ExecuteReader();
while (r.Read())
    Console.WriteLine($"{r["c1"]} {r["c2"]} ...");
```

- Read leerá tupla a tupla de la vista. Para cada tupla, podremos extraer los campos con un indizador con el nombre del campo.
- El indizador devolverá un tipo object que o le hacemos un downcasting o solo podremos llamar a ToString(). Pero si sabemos el tipo que le asignamos al campo al definir la tabla en MySQL. Es más conveniente la siguiente forma de acceder al valor del campo en la tupla, en lugar del indizador.

```
while (r.Read())
    Console.WriteLine($"{r.GetDouble("c1")} {r.GetString("c2")} ...");
```

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Ejecutar Comandos Sql - III

- También podremos obtener el resultado de agregaciones mediante consultas select como count, sum, max, etc.. con ExecuteScalar que devuelve object.

```
MySqlCommand cmd = new MySqlCommand("SELECT COUNT(*) FROM tabla;", c);
int r = int.Parse(cmd.ExecuteScalar().ToString());
```

Llamadas A Procedimientos Almacenados - I

- Deberemos tener creado y añadido el procedimiento a la BD antes de la invocación desde nuestro programa.
- Tendremos que saber el nombre del procedimiento, y dependiendo de la forma en que creemos nuestro código, necesitaremos saber también o el nombre de los parámetros que queremos pasar o la posición que ocupan, en los dos casos deberemos saber de que tipo son.

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Llamadas A Procedimientos Almacenados – II

- Por ejemplo si quisiéramos invocar al siguiente procedimiento...

```
CREATE PROCEDURE inserta(a int, b varchar(25), c int)
BEGIN
  INSERT INTO TABLA (a, b, c);
END;
```

- Usaremos la siguiente sintaxis.

```
using (MySqlCommand cmd = new MySqlCommand()
{
    Connection = c,
    CommandType = System.Data.CommandType.StoredProcedure,
    CommandText = "inserta"

})
{
     // Además del nombre podremos usar la posición o índice.
    cmd.Parameters["a"].value = valor1;
    cmd.Parameters["b"].value = valor2;
    cmd.Parameters["c"].value = valor3;
    cmd.ExecuteNonQuery();
}
```

ACCEDIENDO A LA BBDD MYSQL DESDE C#

Llamadas A Funciones Almacenadas

- No podrá tener el mismo nombre que un procedimiento, porque aunque MySQL lo permite, C# no hace distinción entre los dos tipos y se produce una excepción.

```
CREATE FUNCTION finserta(a int, b varchar(25), c int) RETURN VARCHAR(25)
BEGIN
  INSERT INTO TABLA (a, b, c);
  RETURN 1;
END;
```

- Tendremos el mismo código que antes, pero después de invocar al procedimiento podremos consultar el resultado.

```
{
     cmd.Parameters[1].value = valor1; //El 0 es el valor de retorno.
     cmd.Parameters[2].value = valor2;
     cmd.Parameters[3].value = valor3;
     cmd.ExecuteNonQuery();
     Console.WriteLine($"{cmd.Parameters[0].value}");
     // o alternativamente
     Console.WriteLine($"{cmd.Parameters[@RETURN_VALUE].value}");
}
```