What is

UniDB (**Uni**ty **D**ata**b**ase) is a very simple-to-use system for performing Database **Queries** from your **Unity project** to your **Server** where your Databases are located. With UniDB you can work, on the same Unity project, with **one or more** Databases. Databases can be of **different kinds** as well.

Databases types

At the moment, UniDB supports the following Databases type:

- MySQL
- MariaDB
- PostgreSQL
- SQLite
- MSSQL

Features

Once everything has been correctly configured and your Unity project can communicate with your Server, your project will be able to:

- **read** your Database Tables looking for data (SELECT)
- **write** data in your Database Tables (INSERT, UPDATE)
- **delete** data (DELETE)
- check if data **exists** (EXIST)
- execute various **aggregation** functions (SUM, AVERAGE, etc.)
- and so on…

The UniDB asset

The **UniDB system** consists of two elements:

- the UniDB **Server App**, which is a lightweight PHP web application you have to place in your **web hosting**;
- the UniDB **Unity Asset**, which is a collection of C# scripts you have to place somewhere in your **Unity project**.

Once the two elements are correctly installed and configured in their environment, you can start to execute your Queries.

Updated on 25 January 2023

Requirements

The **UniDB Unity Asset** has been developed with **Unity 2021.2.14f1** and works from this version and above. It uses two frameworks:

- the **UniTask** framework (https://github.com/Cysharp/UniTask) for the various asynchronous operations (it's included inside the asset);
- the **Newtonsoft Json** framework (https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@3.0/manual/index.html) distributed by Unity and which can be installed and updated directly from the *Unity Package Manager* (➔ see the Newtonsoft Json (http://tigerforge.altervista.org/docs/unidb-1-0/installation/newtonsoft-json/) page for details).

The **UniDB Server App** is a simple collection of PHP scripts (*it's not a service or an application you have to launch in your server OS*). In order to work, it should be placed in a **dedicated folder** (*not mandatory, but recommended*), inside a **web hosting** with the following generic specifications:

- PHP version 7.4 or above;
- PDO Extension installed for each Database type you want to work with;
- the Server App folder with full reading/writing permissions;
- the Server App folder freely accessible by a standard HTTP(s) address.

You can also use a local PHP environment (*like Laragon (https://laragon.org/) for example*) for working locally on your machine. In this case, just keep attention to installing the right PHP version and the various PDO Extensions you may need (➔ see the Database Profiles (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/database-profiles/) page for details about this topic).

Database conventions

The UniDB system may rise some issues if the names of your Databases, Tables and Columns contain special characters, spaces or non-Latin characters.
If you are starting from scratch, choose simple names, using alphanumeric characters and avoiding the other characters. For naming just use the common variables' name conventions.

Useful knowledge

The UniDB has been developed in order to be very easy and practical to use. However, it involves various technical topics you should know:

- Databases, tables and queries: because you are using UniDB for this scope, you should have good knowledge about this topic, especially about how queries work and how they are written.

- Asynchronous C# logic: in a Unity project like a game, everything works asynchronously (and it has no sense working differently). The UniDB system relies on asynchronous tasks; this means that when you run a query, you have to wait for its response without blocking the Unity project execution.

Updated on 25 January 2023

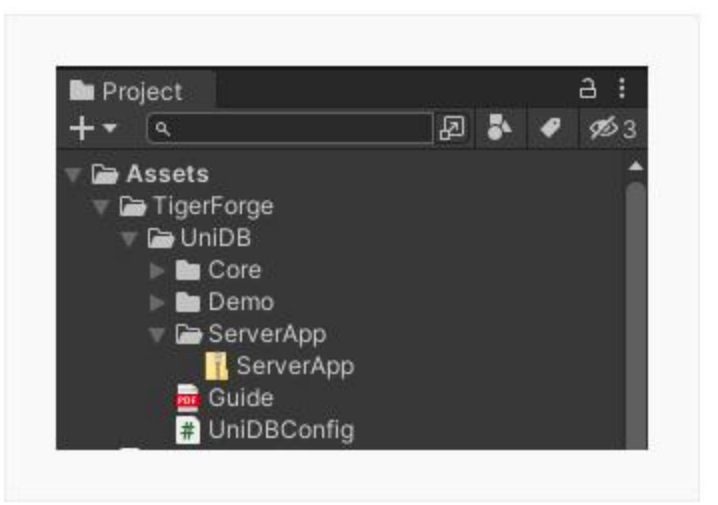
Setting up UniDB

The UniDB system installation consists of two macro steps:

- the installation of the **UniDB Server App** in a web hosting service (➔ see the Requirements (http://tigerforge.altervista.org/docs/unidb-1-0/introduction/#requirements) paragraph for details);
- the configuration of the **UniDB Unity Asset** placed into a Unity project.

UniDB Server App

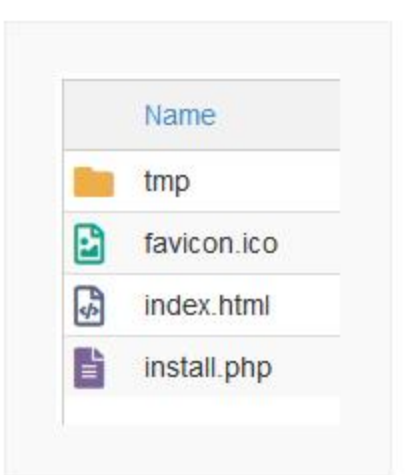
The **UniDB Server App** is included in the **UniDB Package** downloaded from the *Unity Asset Store*. It is compressed into a ZIP file named **ServerApp.zip**, under the TigerForge > UniDB > **ServerApp** *folder*.



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/serverappzipfile.png)

1. Upload the UniDB Server App on a web hosting

The first thing to do is to transfer the **UniDB Server App** to a **web hosting** service or, if you are developing locally (*for example, with Laragon*), to the **folder** controlled by your local environment tool. Unzip the compressed file in the location of your choice. You should have a set of files like this one:



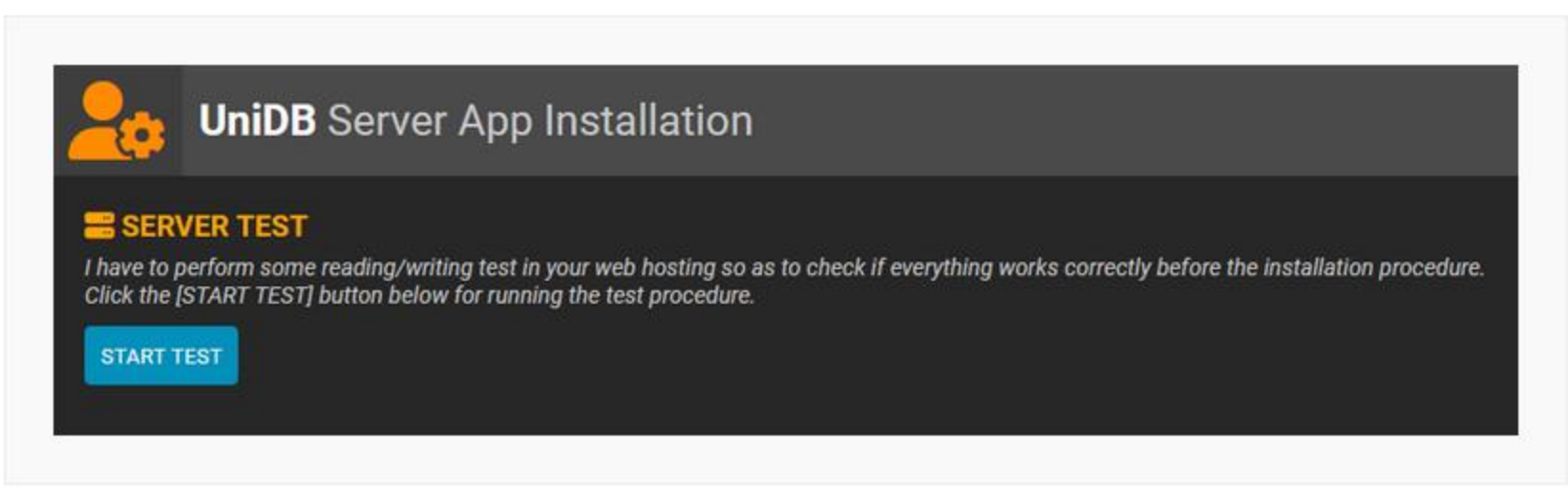
(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbfilelist.png)

2. Run the installation process

In a **browser**, call the **URL** (or the path) which points to the **install.php** file, in order to execute it. For example, if you uploaded the UniDB Server App to your **mysite.com** hosting, under the **db** folder, the URL would be something like this:

https://mysite.com/db/install.php

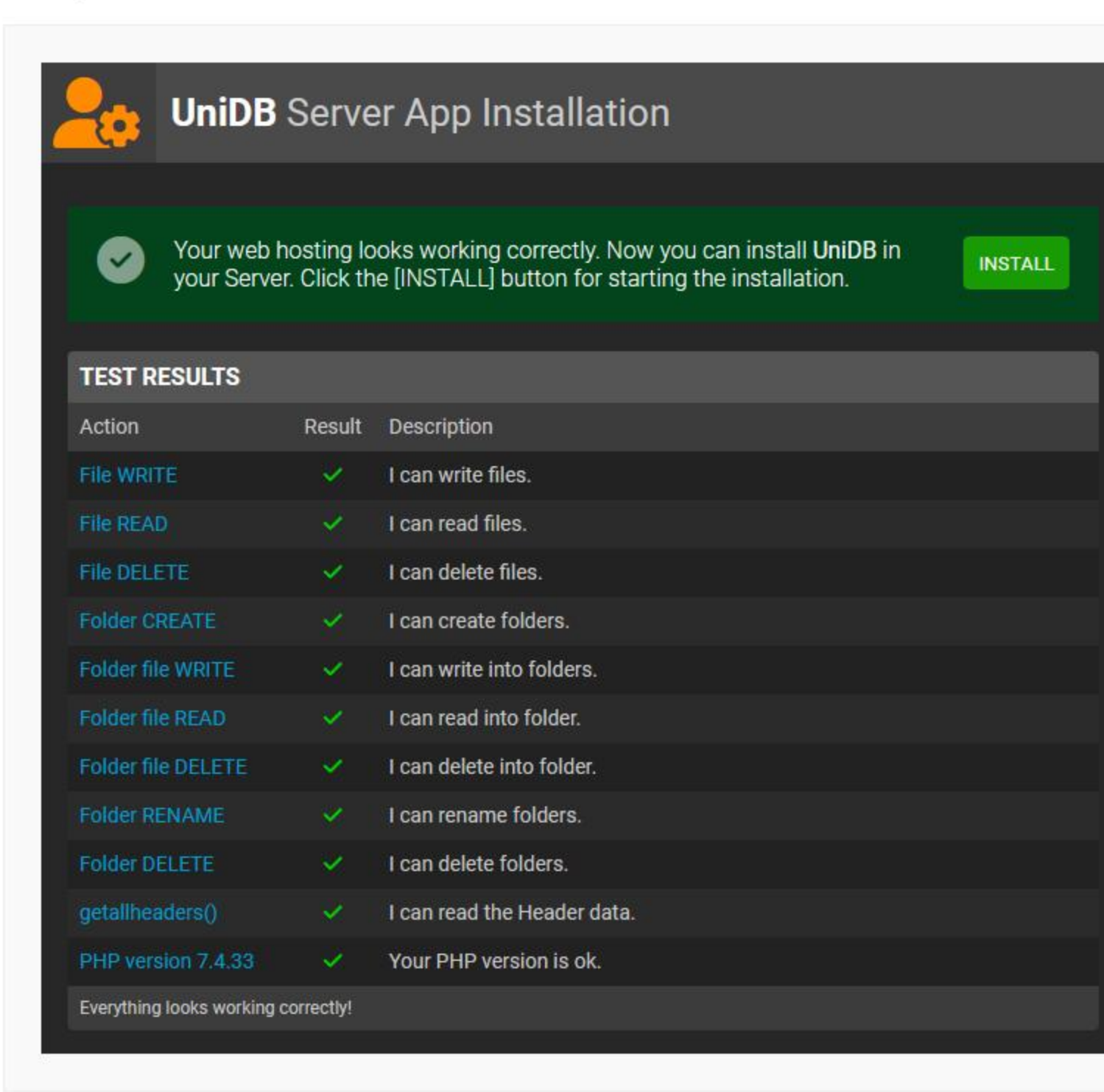
In your browser you should see a page like this one (*so, click the START TEST button*):



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbinstall1.png)

3. Start the Server Test

Because it's pretty important that your web hosting meets certain requirements, the installation process starts with a series of tests. The tests should require just a couple of seconds to be completed and will show a page like this one with the analysis results:

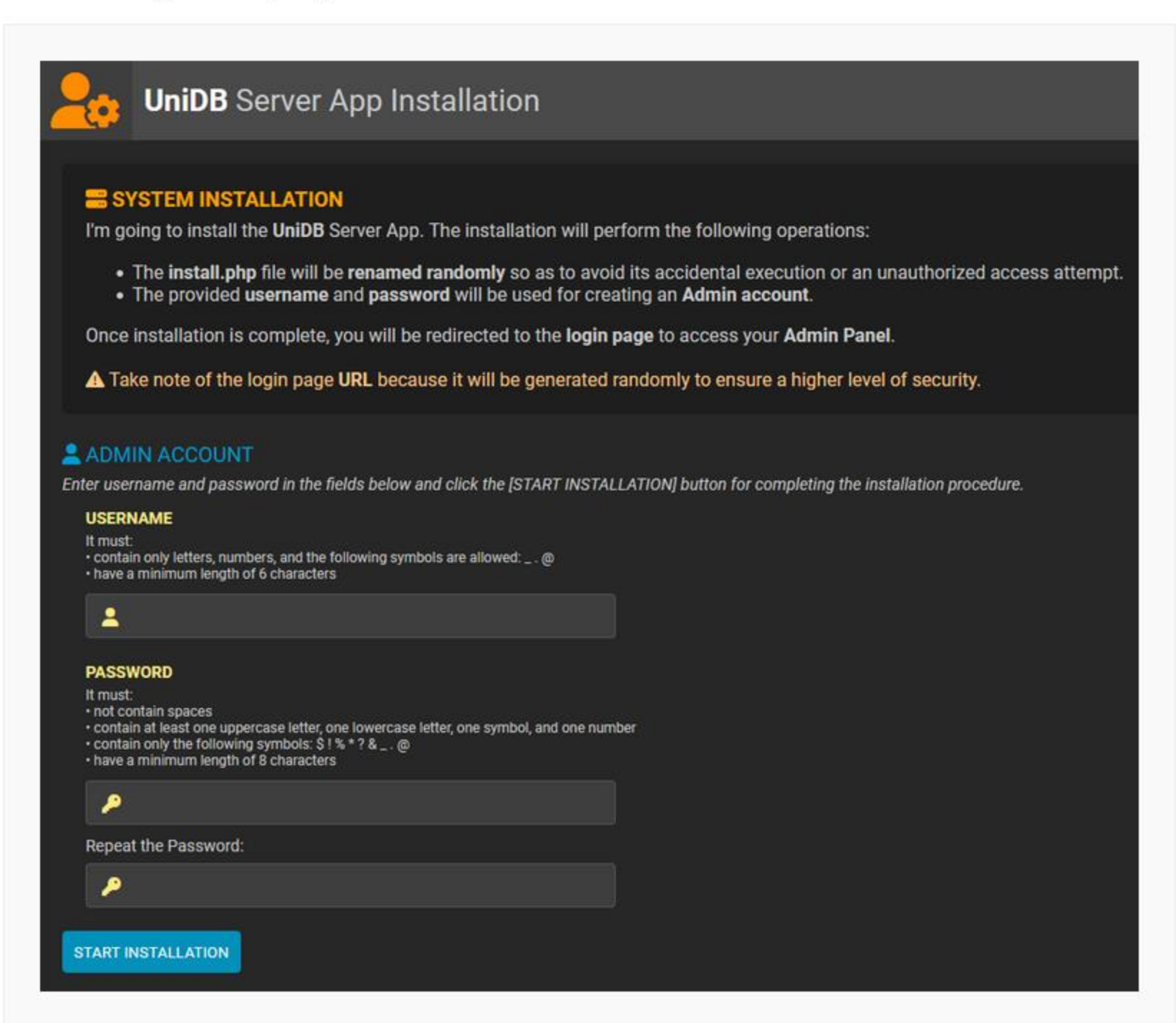


(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbinstallok.png)

In the best scenario, you will have all the test voices checked and the UniDB Server App ready to be installed (*so, click the INSTALL green button*). If instead, something goes wrong, the App will try to give you specific information and tips on how to resolve the found issues (➔ see the Issues (http://tigerforge.altervista.org/docs/unidb-1-0/installation/issues/) page for more details).

4. Create an Admin account

Access to the UniDB Server App is (obviously) protected by username and password. On this page, you must provide a valid and strong **username** and **password**. You will have to use these credentials every time you need to use the UniDB Server App for configuring your Databases.



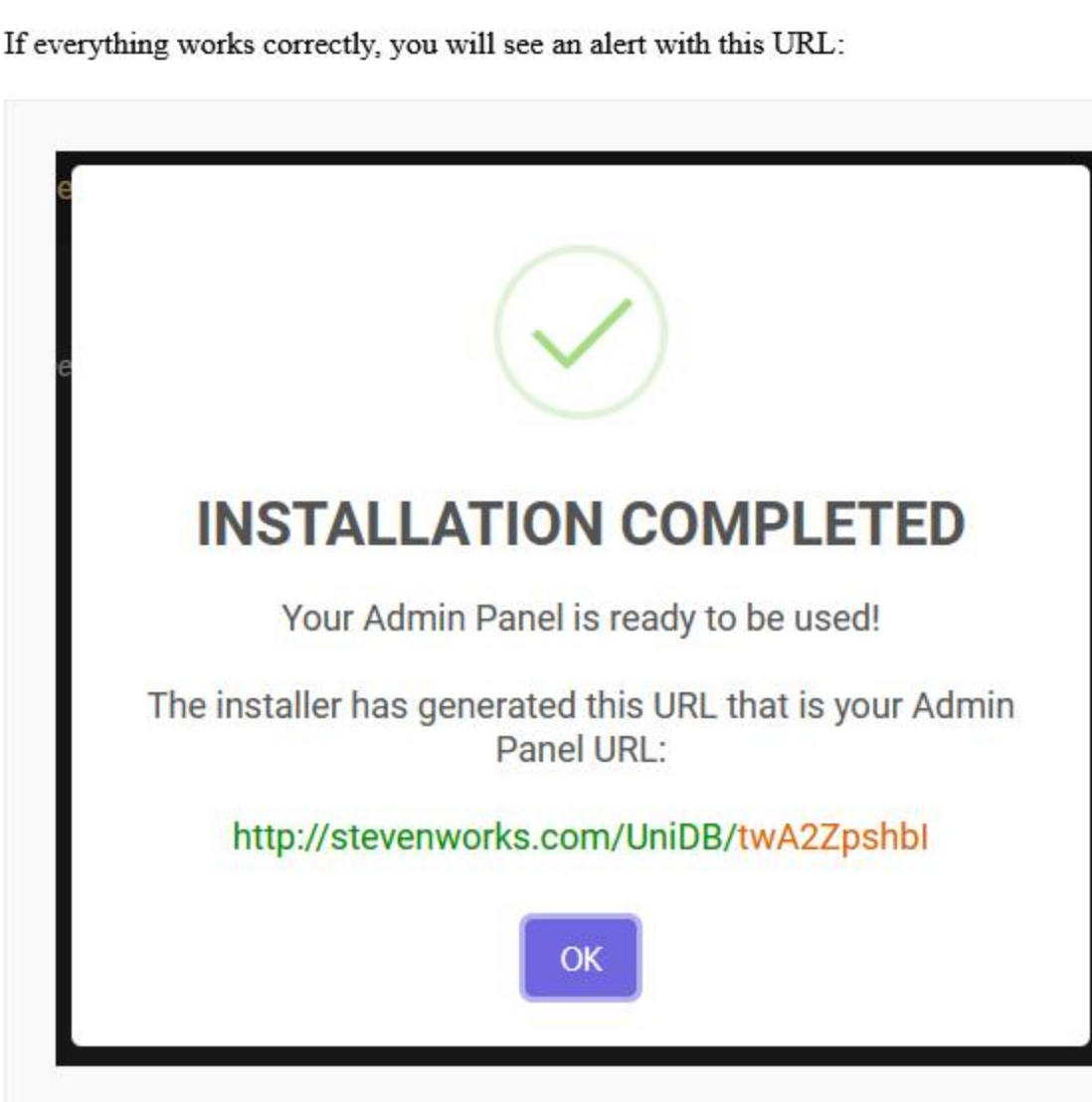
(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbaccount.png)

5. Start the installation

Once you have entered the required information for creating an **Admin account**, click the START INSTALLATION button. This final procedure will perform the following operation:

- the **install.php** file will be randomly **renamed**. It mustn't be called again (*you can even delete it if you prefer!*);
- the **tmp** folder, which contains the UniDB Server App, will be randomly **renamed**. The App will be there, so **take note** of that **URL**. You will have to call that URL when you need to access the App.

If everything works correctly, you will see an alert with this URL:

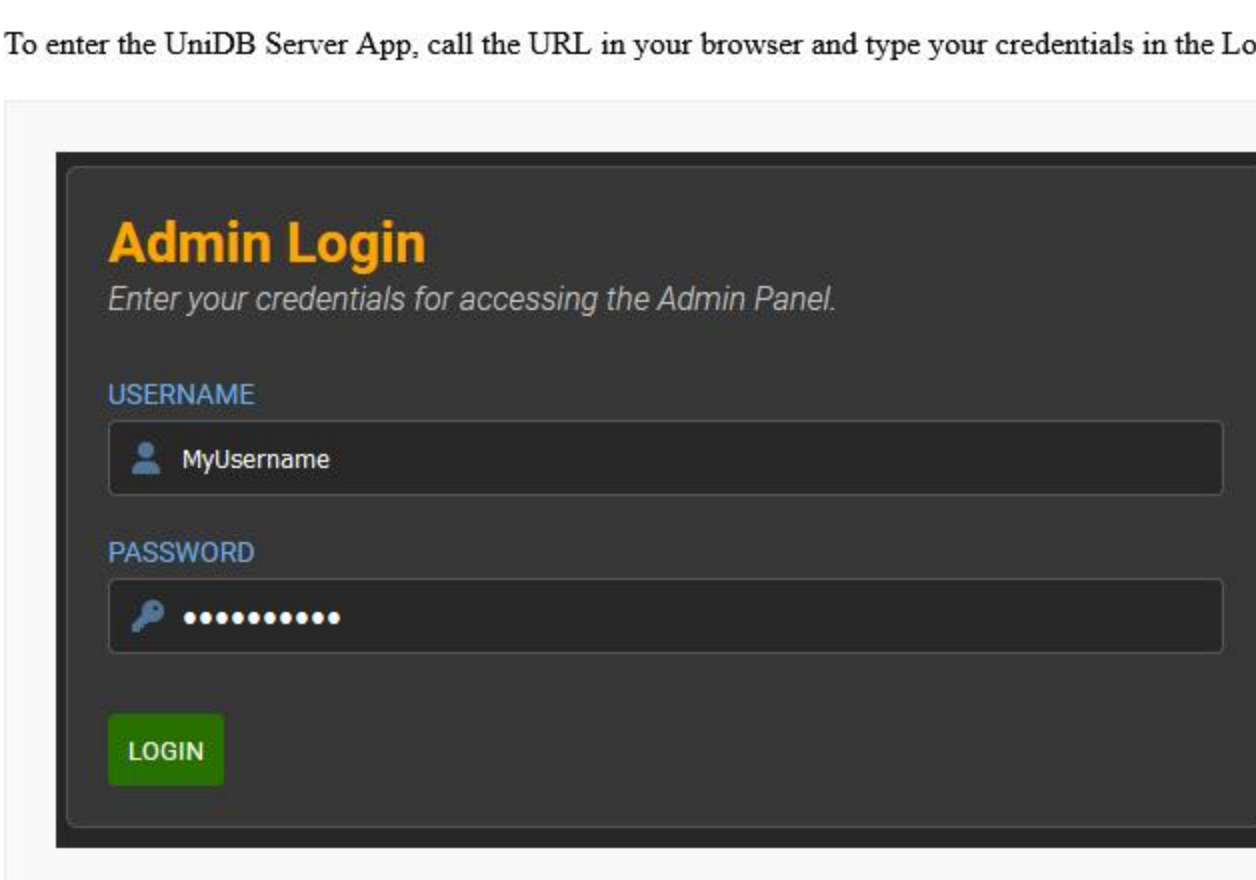


(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbfinished.png)

Clicking OK, you will be redirected to your UniDB Server App.

6. Login

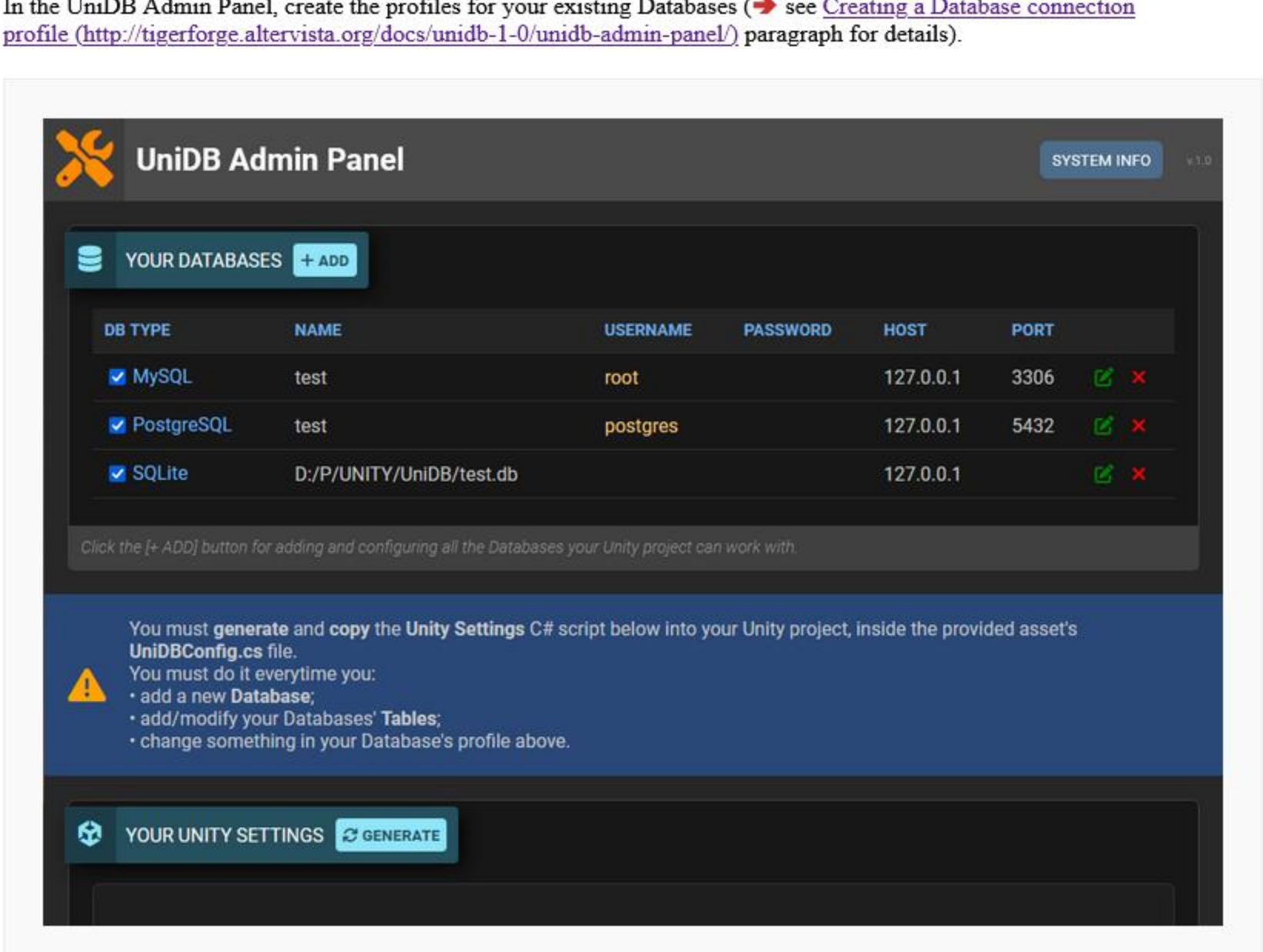
To enter the UniDB Server App, call the URL in your browser and type your credentials in the Login form:



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbadminlogin.png)

7. Add your Database profiles

In the UniDB Admin Panel, create the profiles for your existing Databases (➔ see Creating a Database connection profile (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/) paragraph for details).



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbpanel1.png)

8. Generate the Unity Settings C# script

Click the GENERATE button for generating a C# script containing everything required for working with Unity (➔ see Generating the "Unity Settings Script" (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/) paragraph for details). Then, select the whole test and copy it.

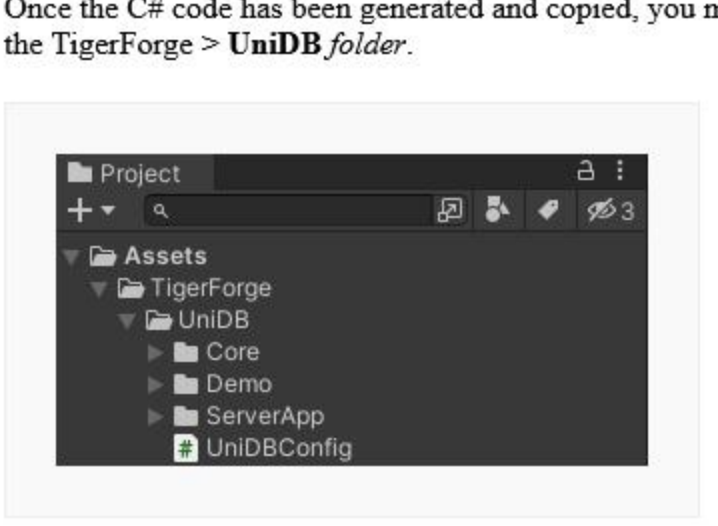
(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbpanel2.png)

UniDB Unity Asset

Because your Unity project **has no idea** where the **UniDB Server App** is located, you must "tell" the **UniDB Unity Asset** where it is (*basically its URL or local path*), which **Databases** it can control, and all the details about **Tables** and **their structure** (*their columns name, data type, and so on*).

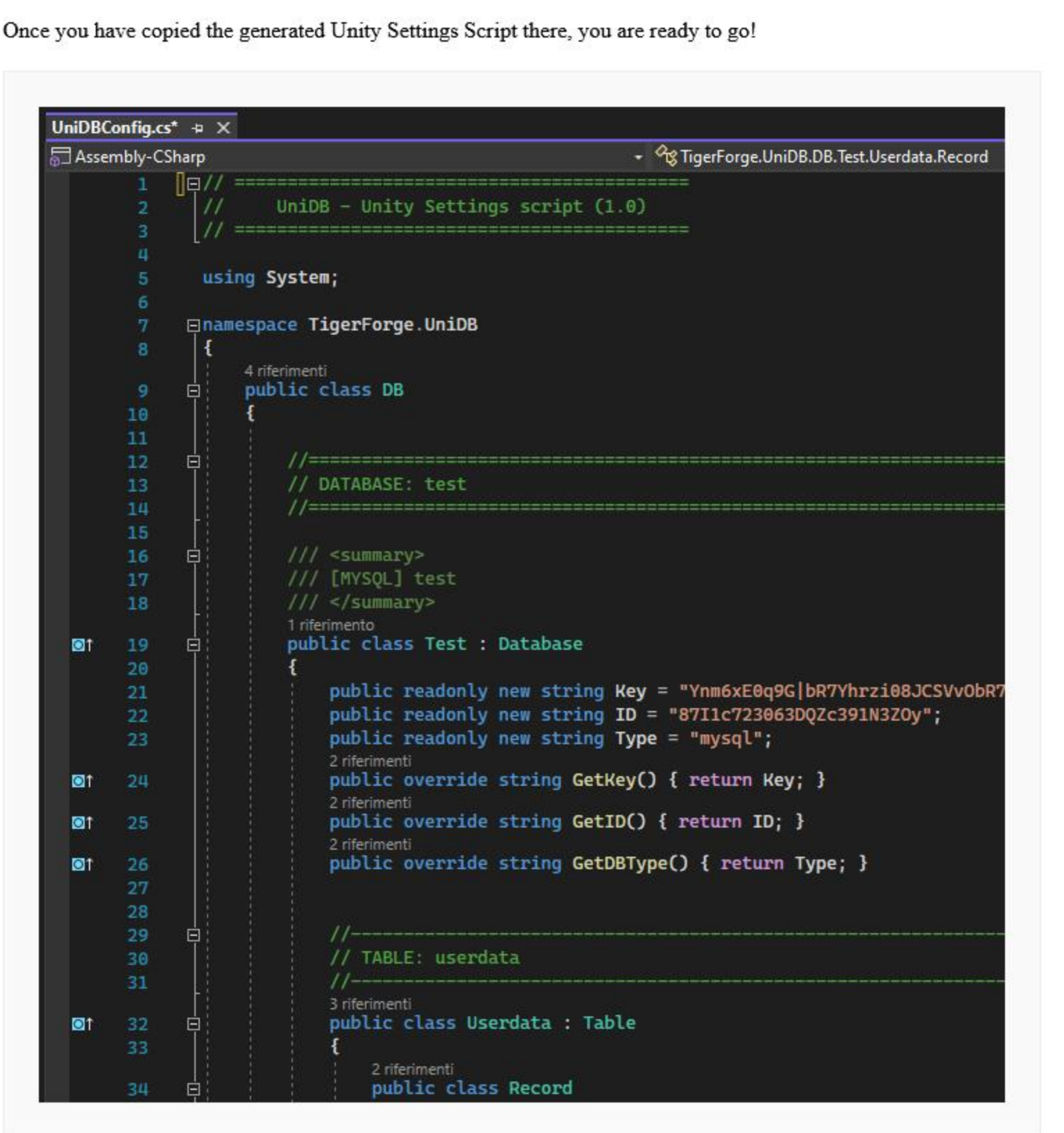
All this is contained in a C# code named **Unity Settings Script** (➔ see The Unity Settings Script (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/the-unity-settings-script/) page for details), which is **generated** by the **UniDB Server App** (*paragraph UniDB Server App, step n.8*).

Once the C# code has been generated and copied, you must **paste** it into the **UniDBConfig.cs** file, which is placed in the TigerForge > **UniDB** *folder*.



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbconfigfile.png)

Once you have copied the generated Unity Settings Script there, you are ready to go!



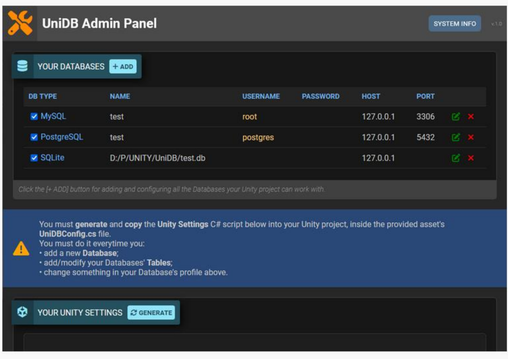
(http://tigerforge.altervista.org/wp-content/uploads/2023/01/uniddbscript.png)

Updated on 25 January 2023

UniDB Admin Panel

The **UniDB Admin Panel** is the *web application* you have to use for creating a **connection profile** for each of your existing **Databases**.

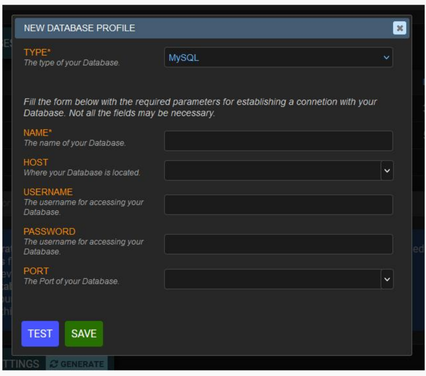
A **connection profile** is the **set of parameters** which are required for **establishing a connection** with your **Database**.



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbpanel1.png)

Creating a Database connection profile

For creating a new connection profile for your Database, just click the [+ ADD] button near the "YOUR DATABASE" title. This will open a modal **window** with a simple **form** you have to fill in with the required information.



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbform.png)

TYPE	This is the type of your Database. UniDB supports the following Databases: • MySQL / MariaDB • PostgreSQL • SQLite • MSSQL Depending on the selection, the fields below may change and the required information may differ.
NAME	This is the name of the Database you are connecting to. SQLite: *for an SQLite Database, it is the path to the .db file. In a local environment, the path should use the slash character "/".*
HOST	This is the location where the Database is hosted. In many typical situations, this parameter is **localhost**, which identifies a Database hosted in the same Server where the UniDB Server App is installed. If you are using a local environment, it may be the IP address generated by the tool you are using (for example, 127.0.0.1).
USERNAME	This is the username of the account which have access to and control of the Database. In some cases, it may be not necessary. For example, SQLite databases usually don't have an account.
PASSWORD	This is the password of the account which have access to and control of the Database.
PORT	This is the number of the port of the Database. Usually, it is a standard number (for example, 3306 for MySQL and 5432 for PostgreSQL) and may be unnecessary.

Once all the required parameters are in, you can click the [TEST] button for testing the connection with your Database. You will get a confirmation message if everything worked correctly or an alert if something went wrong.

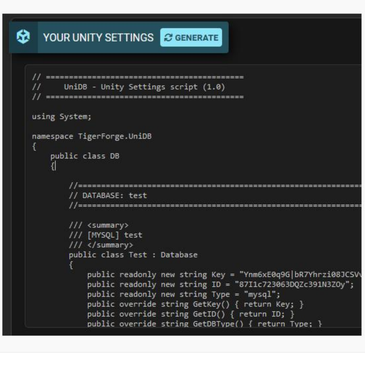
Click the [SAVE] button for adding the profile to the UniDB system.

For each profile, you can update its parameters by clicking the green "edit" button or delete it by clicking the red "delete" button.

Generating the "Unity Settings Script"

The **Unity Settings Script** is a **C#** code generated by the UniDB Server App. It contains your **Databases** and **Tables** represented as a collection of **C# classes** with some **built-in functionalities** (➔ see The Unity Settings Script (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/the-unity-settings-script/) page for details). It also contains the security **Keys** and the **URLs** for establishing a connection with your Server (or local environment).

For generating the script, just click the [GENERATE] button near the "YOUR UNITY SETTINGS" title. The process should take some seconds, depending on the complexity of your Databases (the button's icon will rotate during the process).



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbscript.png)

Important

Keep in mind that you may need to regenerate the script more times during your development process. In detail, it must be generated every time you:

- create a new **Database** or modify an existing Database;
- create a new **Table** or modify an existing Table;
- create a new **connection profile** or modify an existing profile.

Articles

- The Unity Settings Script (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/the-unity-settings-script/)
- Database Profiles (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/database-profiles/)

Updated on 25 January 2023

The Unity Settings Script

The **Unity Settings Script** is a **C# code** you have to **generate** with the **UniDB Server App**, **copy** and **paste** into the **UniDBConfig.cs** file of the **UniDB Unity Asset**, so as to tell your Unity project where your Databases are and which Tables they contain.

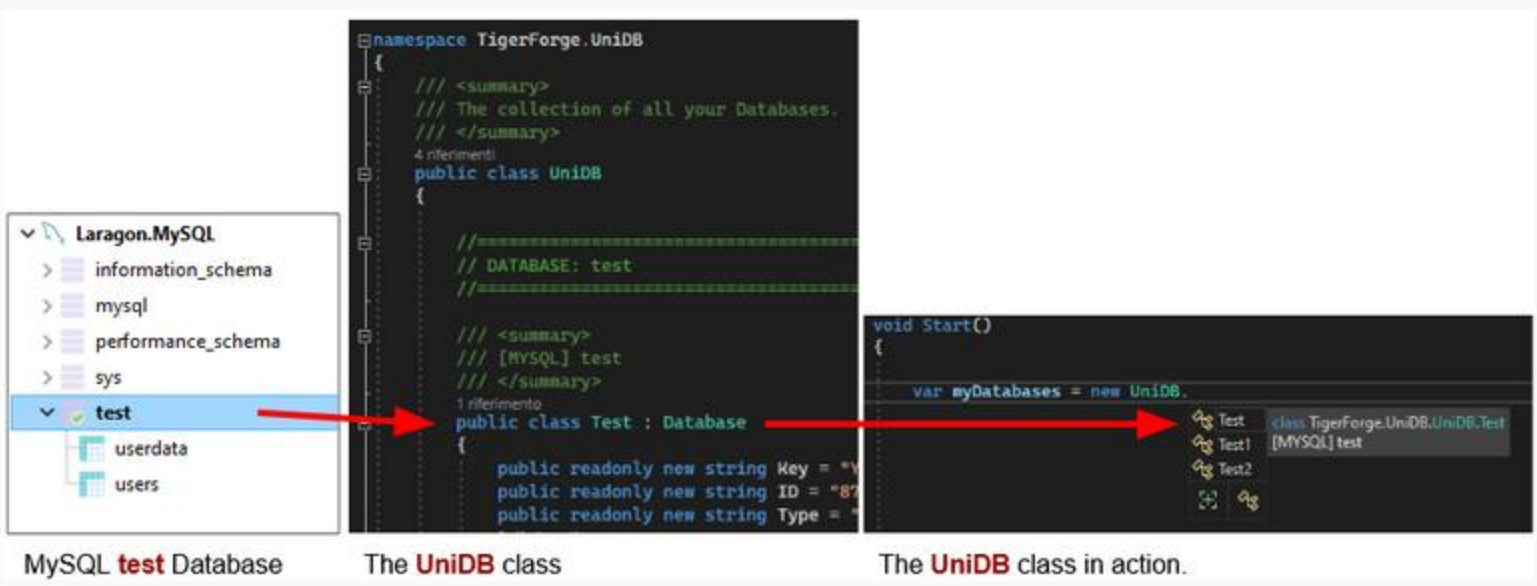
Basically, you can consider the Unity Settings Script as a photocopy of your Server, Databases and Tables characteristics.

In detail, the script contains:

- the security **Keys** and **Tokens** for establishing a secure connection with the UniDB Server App;
- the **URL** (or path) where the **UniDB Server App** is installed;
- a collection of organized **C# classes** which represent all your **Databases**, with their **Tables** and the Tables' **structure**.

The UniDB class

The Unity Settings Script exposes a macro *class* named **UniDB**. This *class* contains *references* to all your **Databases**.

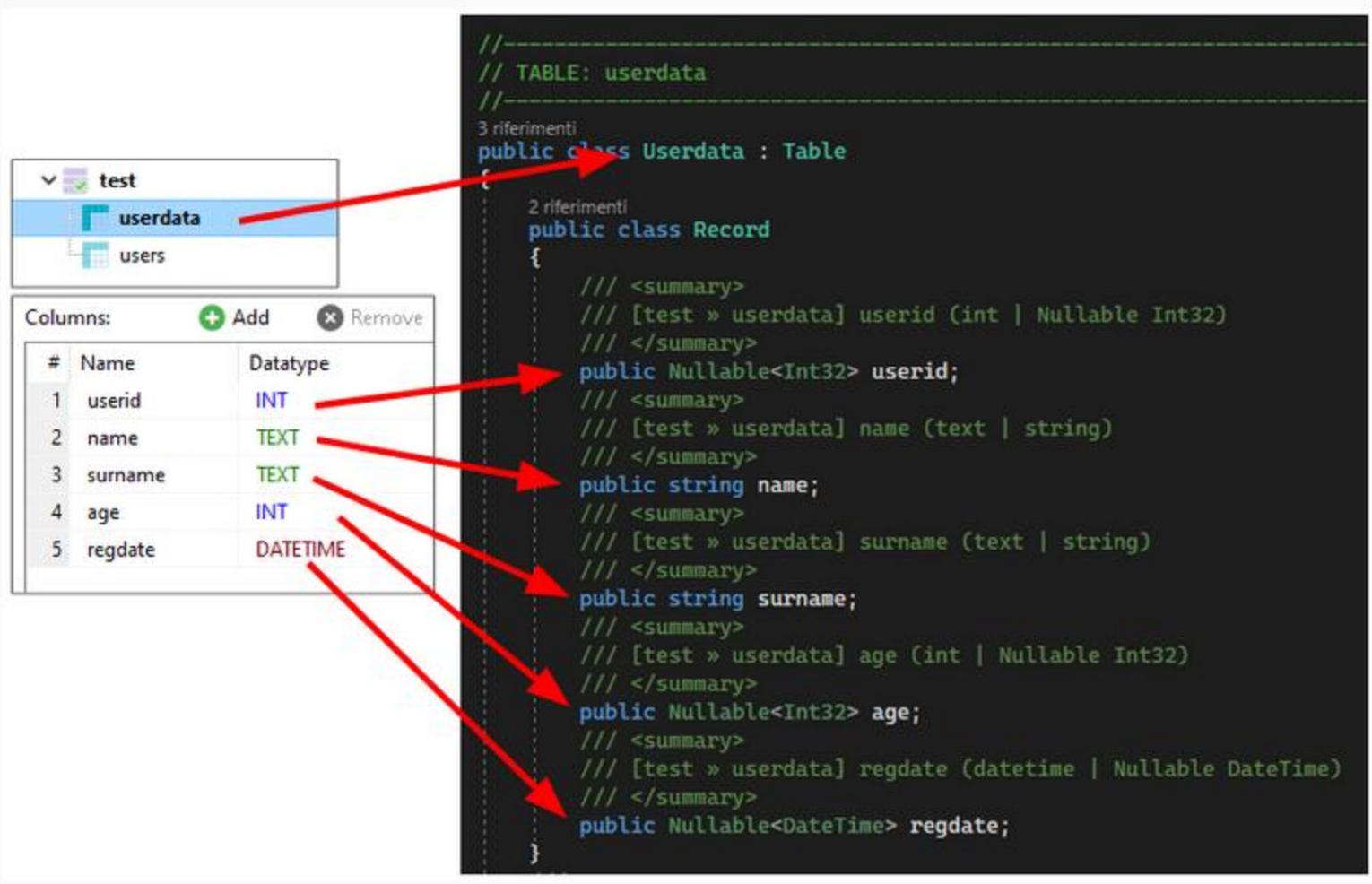


(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbexample1.png)

Under the UniDB class, all your Databases are represented by a **dedicated class** with the same Database name (*for the convention, the name will have a capital letter*).

A **Database** *class* has some properties and methods which are used internally by the UniDB system, like the various security Keys, your Server URL and so on. This part of the code is something you don't need to use or care about. Instead, this class contains a collection of important sub-*classes*: the **Table** *classes*.

A **Table** *class* replicates the **structure** of a Table, including a series of useful properties and methods that will help you develop Queries. During the Unity Settings Script generation, the UniDB Server App analyses the Table's structure, detects the columns' names and their data type, and translates them into C# properties with a suitable data type.



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbtables.png)

Updated on 12 January 2023

Database Profiles

A **Database Profile** is a set of **parameters** required for establishing a **connection** with a **Database**. Depending on the kind of Database you have, these parameters may change.

Database name

The **name** of the Database is a **mandatory** parameter. It's the name you gave to your Database.
For the **SQLite** Database, it usually is the name of the **.db file**.

Host

This is the **address** where your Database is hosted (installed).
A typical common parameter is **localhost**, which means that your Database is located in the same place the UniDB Server App is installed.
If you are using a **local** PHP environment on your PC, for example through a tool like Laragon or WinAMP, the Host is usually an IP address like 127.0.0.1. It depends on the tool you're using, but this information should be clearly documented by your tool.
If you are using a Database on a **cloud** service, like Azure or AWS, it may be a specific address. For example, if you are using an MSSQL Database on Azure, the Host may be the Azure "Server name", a string like *my_server_name.database.windows.net*.

Username and password

For comprehensible security reasons, a Database is usually protected by an **account**. It's not mandatory, but many server/hosting *providers* allow the use of Databases only if protected by an account.
For the **SQLite** Database, which typically is a local file used directly by your app, it may be omitted.

Port

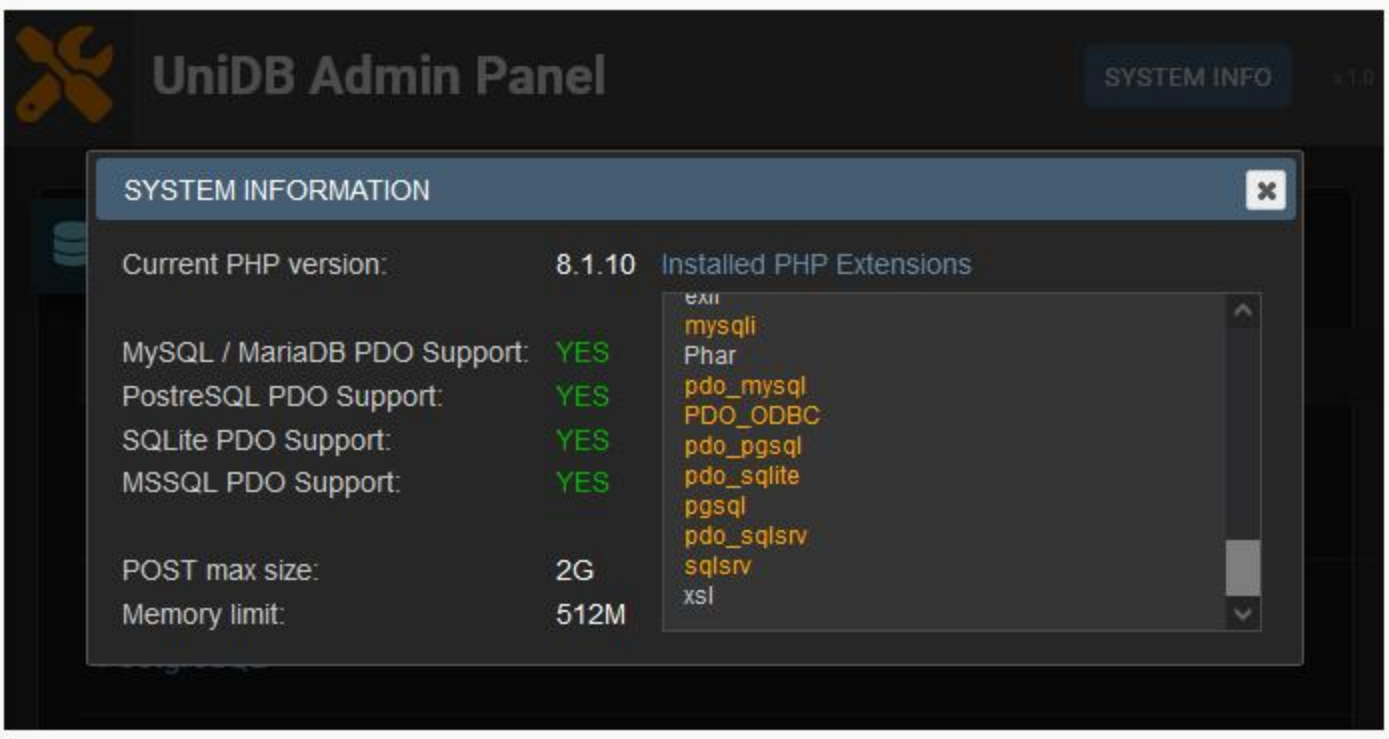
The **Port number** is often an **optional** parameter simply because your Server automatically assigns the right number during the Database's connection process. Moreover, it usually is a standard predefined number for specific kinds of Databases, like 3306 for MySQL and 5432 for PostgreSQL.

PHP PDO support

Because the connection to your Database relies on **PHP**, you must check if your server/hosting is able to work with the kind of Database you need. PHP uses specific **extensions** called **PDO**; for each Database's type PHP requires a specific PDO extension:

Database's Type	PDO Extension (Driver)
MySQL, MariaDB	php_pdo_mysql
MSSQL	php_pdo_sqlsrv / php_pdo_dblib
SQLite	php_pdo_sqlite
PostgreSQL	php_pdo_pgsql

The **UniDB Server App** has a utility function which scans your PHP's installed **extensions** looking for the **PDOs**. Just click the [SYSTEM INFO] button on the top right. It will show a popup window with some useful information:



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbsysteminfo.png)

The most important information is the **supported PDO** analysis, which tells you what kinds of Databases your PHP can work with. On the right, the list of the installed PHP Extensions gives you all the extensions you have, highlighting in orange those related to the Databases connections.

Tips

A typical issue you may encounter creating a profile is a **Database connection failure**. When it happens, the UniDB Server App tries to give you the reason why it happened, so that you can find a solution.
There are a lot of reasons why you can't connect to your Database. However, the most common reason is the use of incorrect parameters for the Database Profile.

Using **Database management software** like DBeaver or HeidiSQL can help you better understand if the parameters you have are correct for establishing a connection. If there is a wrong parameter, these tools can give you technical information about how to resolve the issue.

Updated on 25 January 2023

Getting Started

1.

The first thing to do is create an *instance* of a **Database** using the **UniDB** *class*, which contains the collection of all your Databases:

var database_instance = new UniDB *my_database_name*();

database_instance	The name of the variable that will be an *instance* of one of your Databases.
my_database_name	The name of the *class* which represents the Database you are going to work with.

```
Example
var myTestDatabase = new UniDB.Test();
// 'Test' is the name of the Database.
```

2.

When you work with Databases and Queries, you are practically working with **Tables**. Everything you are going to do with UniDB will have an effect on your Tables. For this reason, now you have to create an *instance* for each **Table** you are going to work with. The instance you have created in **step 1** contains a set of built-in methods named **GetTable_*()**, which return an *instance* of a Database's Table.

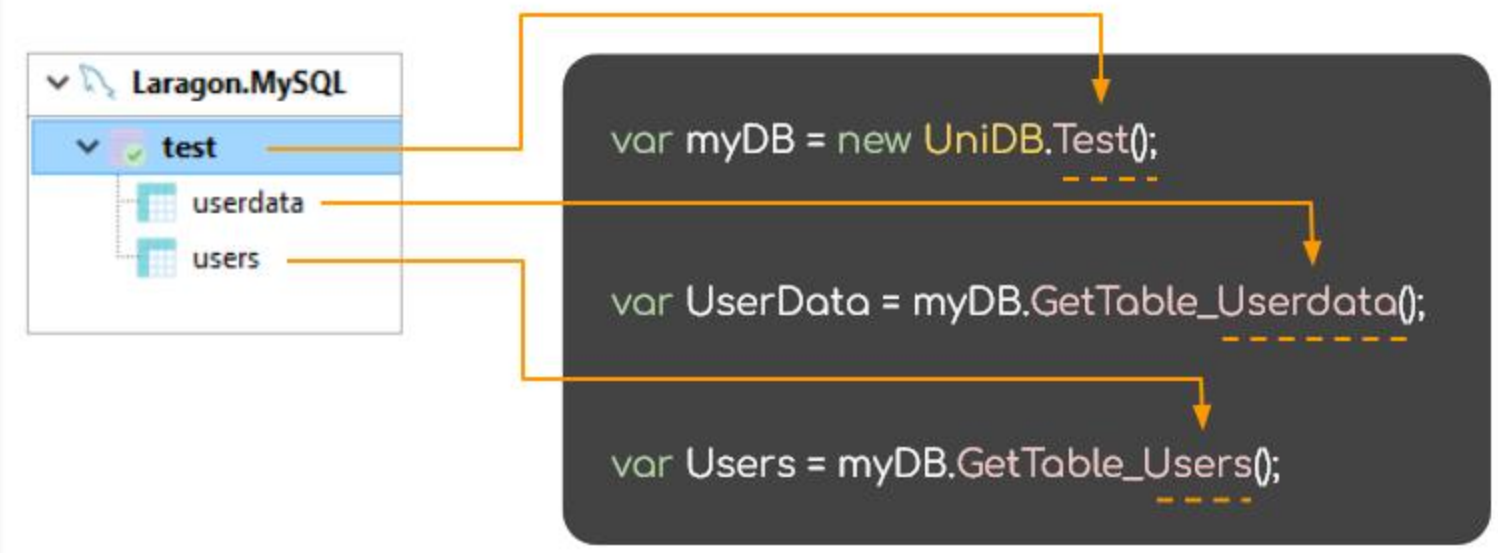
var table_instance = database_instance GetTable_*my_table_name*();

table_instance	The name of the variable that will be an *instance* of one of your Tables.
database_instance	The name of the variable which represents the Database you are working with.
my_table_name	The name of the Table you are working with, integrated into the method's name GetTable_*()

```
Example
var myTestDatabase = new UniDB.Test();
// 'Test' is the name of the Database.
var myUsers = myTestDatabase.GetTable_Users();
// 'Users' is the name of a Table inside the 'Test' Database.
```

3.

Now you can start assembling your **Queries** using the table's *instance* created in **step 2** (➔ see the Creating Queries (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/creating-queries/) page for details).



(http://tigerforge.altervista.org/wp-content/uploads/2023/01/unidbschema3.png)

Async logic

Keep in mind that all the UniDB methods are **asynchronous** *tasks*. Always use the **Run()** method for **catching the result** of the queries and manage the flow of your Unity project in consideration of this async logic.

Updated on 31 January 2023

Creating Queries

UniDB uses a built-in set of **methods** for allowing you to create **SQL Queries** in a very practical way. Even if the fastest way would be directly writing queries by yourself, with the system offered by UniDB you don't have to remember all the SQL statements, how you should write them and, above all, you don't have to worry about typos, security issues and many other things that would waste your time.

Promise logic

UniDB uses a **promise logic** similar to the one adopted by *Javascript*; this means you have to call and configure a **sequence** of methods in a specific order and, finally, call the **Run()** method for **executing** the built Query.

In general, this is the structure of a Query assembled with UniDB:

_ = table_instance *Method*(parameters) *Method*(parameters) … Run(parameters);

table_instance	The name of the variable which represents the Table you are working with.
Method	A series of specific methods (with their expected parameters) which build a Query.
Run	The final mandatory method to call for executing the Query and catching the result of the operation (*the reply from the UniDB Server App*).

```
Example
var TestDB = new UniDB.Test();
// 'Test' is the name of the Database.

var myUsers = TestDB.GetTable_Users();
// 'Users' is the name of a Table inside the 'Test' Database.

_ = myUsers
    .SelectOne()
    .Where(myUsers.C.username.equal("mario"))
    .Run(
       (UniDB.Test.Users.Record data, Info info) =>
       {
           if (info.isOK) Debug.Log("Mario's password: ", data.password);
       }
    );
```

Methods order

The Methods should be used respecting the following order:

.Select* .Insert .Update .Delete .Drop .Join .Min .Max .Sum .Avg .Replace .Count .Exists	Operations The operations methods should be defined as the first method.
.Data	Data The Data method is used by Insert and Update, so it should be defined after them.
.Where	Where The Where condition is used by many operations methods. It should be defined after them.
.OrderBy .Limit	OrderBy and Limit They have effect on the returned values. They should be used after Where.
.Run	Run The Run method must be used as the last method. It executes the Query.

Updated on 25 January 2023

Insert

The **Insert()** method **writes a new Record** into the Table. The method itself has no parameter, but it requires the use of the **Data()** method for specifying a **value** for each Table's **column** (*the omitted columns will have the default value as defined in your Table's structure*).

… Insert() …

Data()

In the Insert operation, the **Data()** method is used for specifying the **values** of the Record's **columns**. The method requires a list of the columns you want to set with their assigned value. For doing this, you have to use the **Value()** method exposed by each column:

… Data(Column Value(value jsonEncode = false) …) …

Parameter	Data-type	Description
value	object	A value for this column. The parameter is an **object** *data type* and can receive any kind of value. However, this value should be of the **same** type as the column's type. For example, if the column is an *int* column, the value has to be an *integer* number. The parameter supports **SQL Functions** (➔ see the SQL Functions (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/sql-functions/) page for details).
jsonEncode	bool (*Optional*)	If set to true, the value will be translated into a JSON string.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// INSERT INTO users (username, password) VALUES ("pluto", "mypass1234")
_ = myUsers
    .Insert()
    .Data(
          myUsers.C.username.Value("pluto"),
          myUsers.C.password.Value("mypass1234")
    )
    .Run( ... )
```

Multiple Insert

The Insert() method supports the writing of **more than one Record** with a **single call**. You have just to use the **Data() method more times**, for each Insert operation you want to perform.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// INSERT INTO users (username, password) VALUES ("user1", "pass1234");
// INSERT INTO users (username, password) VALUES ("user2", "pass4567");
// INSERT INTO users (username, password) VALUES ("user3", "pass8901");
_ = myUsers
    .Insert()
    .Data(
          myUsers.C.username.Value("user1"),
          myUsers.C.password.Value("pass1234")
    )
    .Data(
          myUsers.C.username.Value("user2"),
          myUsers.C.password.Value("pass4567")
    )
    .Data(
          myUsers.C.username.Value("user3"),
          myUsers.C.password.Value("pass8901")
    )
    .Run( ... )
```

Run()

For the **Insert()** operation, you must use override 3 of the Run() (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/run/) method.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// INSERT INTO users (username, password) VALUES ("pluto", "mypass1234")
_ = myUsers
    .Insert()
    .Data(
          myUsers.C.username.Value("pluto"),
          myUsers.C.password.Value("mypass1234")
    )
    .Run(
          (Info info) =>
          {
              if (info.isOK)
                 Debug.Log("Inserted record ID: " + info.id);
              else
                 Debug.LogWarning("No record inserted!");
          }
    )
```

The **Info** parameter will contain the following specific data for the Insert operation:

id	int	The ID of the **last** inserted Record. **NOTE** If for some reason the ID is not available, a **0 value** is returned.

Updated on 31 January 2023

Update

The **Update()** method **updates an existing Record** in the Table. The method itself has no parameter, but it requires the use of the **Data()** method for specifying the new **values** for the Table's **columns** (*the omitted columns won't be updated*).

… Update() …

Note

Even if in the SQL Update statement the use of a condition is not mandatory, for security reasons the Update() method of the UniDB system requires you to use the **Where()** method.

Data()

In the Update operation, the **Data()** method is used for specifying the new **values** of the Record's **columns** (those you want to update). The method requires a list of the columns you want to set with their new value. For doing this, you have to use the **Value()** method exposed by each column:

… Data(Column Value(value jsonEncode = false) …) …

Parameter	Data-type	Description
value	object	A new value for this column. The parameter is an **object** *data type* and can receive any kind of value. However, this value should be of the **same** type as the column's type. For example, if the column is an *int* column, the value has to be an *integer* number. The parameter supports **SQL Functions** (➔ see the SQL Functions (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/sql-functions/) page for details).
jsonEncode	bool (*Optional*)	If set to true, the value will be translated into a JSON string.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// UPDATE users SET username = "pluto" WHERE ...
_ = myUsers
    .Update()
    .Data(
          myUsers.C.username.Value("pluto"),
    )
    .Where( ... )
    .Run( ... )
```

Mathematical operations

In the Update operation, the Data() method supports the four basic **mathematical operations**: addition, subtraction, multiplication, and division. Instead of **replacing** a numeric value with a new one, you can **recalculate** that value. For activating this feature, you have to write a **string** containing the following **symbols** and the **number** involved in the calculation:

OPERATION	SYMBOL	EXAMPLE
Addition	{+}	{+}10 The column's value will be increased by 10.
Subtraction	{-}	{-]5 The column's value will be decreased by 5.
Multiplication	{*}	{*}2 The column's value will be multiplied by 2.
Division	{/}	{/}2 The column's value will be divided by 2.

```
Examples
_ = myUsers
    .Update()
    .Data(
          myUsers.C.money.Value("{+}1200"),
    )
    .Where( ... )
    .Run( ... )
```

Run()

For the **Update()** operation, you must use override 3 of the Run() (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/run/) method.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// UPDATE users SET username = "pluto" WHERE id = 6
_ = myUsers
    .Update()
    .Data(
          myUsers.C.username.Value("pluto"),
    )
    .Where( myUsers.C.id.Equal(6) )
    .Run(
          (Info info) =>
          {
              if (info.isOK)
                 Debug.Log("I have updated " + info.affectedRows + " Records.");
              else
                 Debug.LogWarning("No record inserted!");
          }
    )
```

The **Info** parameter will return the following specific data for the Update operation:

affectedRows	int	The number of Records which has been updated. **NOTE** If 0 (zero) is returned, it doesn't mean that an error occurred. It simply means the no Records have been updated.

Updated on 31 January 2023

Delete

The **Delete()** method **deletes existing Records** in the Table. The method itself has no parameter, but it requires the use of the **Where()** method for specifying which Records have to be deleted.

... Delete() ...

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// DELETE FROM users WHERE ...
_ = myUsers
    .Delete()
    .Where( ... )
    .Run( ... )
```

Run()

For the **Delete()** operation, you must use override 3 of the Run() (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-class/run/) method.

```
Examples
var TestDB = new UniDB.Test();
var myUsers = TestDB.GetTable_Users();

// SQL equivalent:
// DELETE FROM users WHERE age > 50
_ = myUsers
    .Delete()
    .Where( myUsers.C.age.Greater(50) )
    .Run(
          (Info info) =>
          {
              if (info.isOK)
                 Debug.Log("I have deleted " + info.affectedRows + " Records.");
              else
                 Debug.LogWarning("No record inserted!");
          }
    )
```

The **Info** parameter will return the following specific data for the Delete operation:

affectedRows	int	The number of Records which has been deleted.

Updated on 25 January 2023

Run

The **Run()** method **executes the Query** you have built using the various SQL-related methods. It must be used as the **last method** to complete what has been defined as a **Query**.

… Run(onDone onError = null);

Parameter	Data-type	Description
onDone	Action	The callback function to call when the operation is completed. The function parameters changes accordingly to the various Run() overrides.
onError	Action< info > (*Optional*)	The callback function to call when a system error occurs. The function must contain a parameter of Info type which will contain the issue details.

onDone

The **onDone** parameter is the **Action** *function* to call when the operation **is completed** (the Query has been executed and the UniDB Server App has returned the result of the operation).

Because the various operations methods return different kinds of results, the Run() method has three **overrides** which cover the possible **results scenarios**:

1. One Record only

For those methods that return only one Record (SelectOne), you must use this override:

… Run(< T info > onDone onError = null);
1

2. List of Records

For those methods that may return more Records (Select, SelectRand), you must use this override:

… Run(< List<T> info > onDone onError = null);
2

3. Specific information

For those methods that return specific information (Insert, Update, Delete, etc.), you must use this override:

… Run(< info > onDone onError = null);
3

Parameter	Data-type	Description
T	Record object	The class which describes a Table's Record (see the **Record class** paragraph below).
info	Info object	A parameter of type Info that will contain specific information about the operation result.

Record class

For the overrides 1 and 2 of the Run() method, you have to use the **Record** *class* of the involved **Table**. The Record class is defined inside the Unity Settings Script (http://tigerforge.altervista.org/docs/unidb-1-0/unidb-admin-panel/the-unity-settings-script/), for each Table. It always has to be called with the following ordinary syntax:

UniDB database_class table_class Record

database_class	The *class* which describes a **Database**.
table_class	The *class* which describes that Database's **Table**.

Info class

The **Info** *class* exposes a set of **properties** for the information returned by some operations methods. Which property will contain specific data depends on the executed operation. For example, methods like Update() and Delete() initialize the affectedRows *property*, whereas the Insert() method initializes the id *property*.

Property	**Data type**	**Description**	**Initialized by**
query	string	The SQL Query string that has been executed.	All
id	int	The ID of the Record.	Insert()
affectedRows	int	The number of Records involved in the operation.	Update() Delete() Replace()
result	float	The numeric result.	Count() Min(), Max() Sum(), Avg()
exists	bool	Whether the searched Record exists.	Exists()
error	string	The error message.	All
status	string	The response status (OK or ERROR)	All
hasData	bool	True if the response contains data.	All
isEmpty	bool	True if the response doesn't contain data.	All
isOK	bool	True if the response is OK.	All
isError	bool	True if the response contains an error.	All
rawData	string	The raw string data received by the UniDB Server App.	All

Updated on 25 January 2023