LU5 - Publish a complete working game using Unity Engine

▼ SQL

▼ What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

▼ What can it do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

▼ Introduction

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data (columns).

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement selects all the records in the "Customers" table:

```
SELECT * FROM Customers;
```

Keep in Mind That...

- SQL keywords are NOT case sensitive: `select` is the same as `SELECT`

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

In this tutorial, we will use semicolon at the end of each SQL statement.

Some of The Most Important SQL Commands

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

▼ CRUD

- CRUD (**Create, Read, Update, Delete**) is an acronym for ways one can operate on stored data. It is a mnemonic for the four basic functions of persistent storage.

▼ Create

▼ Create Database

```
CREATE DATABASE DATABASE_NAME;
```

▼ Create Table

```
CREATE TABLE TABLE_NAME
(
  COLUMN_1 DATA_TYPE,
  COLUMN_2 DATA_TYPE,
  COLUMN_3 DATA_TYPE,
  COLUMN_4 DATA_TYPE,
  COLUMN_5 DATA_TYPE,
  ...
);

CREATE TABLE Persons
(
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);
```

▼ Primary Key

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
// Primary Key
CREATE TABLE Persons
(
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (ID)
);
```

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

```
-- MySQL
CREATE TABLE Persons
(
  Personid int NOT NULL AUTO_INCREMENT,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (Personid)
);

-- SQL Server
CREATE TABLE Persons
(
  Personid int IDENTITY PRIMARY KEY,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int
);
```

▼ Select

The `SELECT` statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

Example

```
-- Get Specific Columns
SELECT COLUMN_1, COLUMN_2 FROM TABLE_NAME

-- Get All Columns
SELECT * FROM TABLE_NAME
```

▼ Where

The `WHERE` clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

```
SELECT *
FROM TABLE_NAME
WHERE CONDITION
```

Note: The `WHERE` clause is not only used in `SELECT` statements, it is also used in `UPDATE`, `DELETE`, etc.!

Customers Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

```
SELECT *
FROM CUSTOMERS
WHERE Country='Germany'
```

Operators in The WHERE Clause

The following operators can be used in the `WHERE` clause:

Aa Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

▼ Update

The `UPDATE` statement is used to modify the existing records in a table.

```
UPDATE TABLE_NAME
SET COLUMN_1 = VALUE_1, COLUMN_2 = VALUE_2,...
WHERE CONDITION
```

Note: Be careful when updating records in a table! Notice the `WHERE` clause in the `UPDATE` statement. The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated!

Customers Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

```
-- Update 1 Record
UPDATE Customers
SET ContactName = 'Chocolate Cake', City = 'Frankfurt'
WHERE CustomerID = 1

-- Update Multiple Records
UPDATE Customers
SET ContactName = 'Unknown'
WHERE Country = 'Mexico'
```

Note: Be careful when updating records. If you omit the `WHERE` clause, ALL records will be updated!

```
UPDATE Customers
SET ContactName = 'Johnny'
```

▼ Insert

The `INSERT INTO` statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the `INSERT INTO` statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO TABLE_NAME (COLUMN_1, COLUMN_2,...)
VALUES (VALUE_1, VALUE_2, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the `INSERT INTO` syntax would be as follows:

```
INSERT INTO TABLE_NAME
VALUES (VALUE_1, VALUE_2, ...)
```

▼ Example 1

The following SQL statement inserts a new record in the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Resulting Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

▼ Example 2

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Resulting Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

▼ Delete

The `DELETE` statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the `WHERE` clause in the `DELETE` statement. The `WHERE` clause specifies which record(s) should be deleted. If you omit the `WHERE` clause, all records in the table will be deleted!

Delete Record

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Customers Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Resulting Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM TABLE_NAME;
```

▼ Date

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- `DATE` - format YYYY-MM-DD
- `DATETIME` - format: YYYY-MM-DD HH:MI:SS
- `TIMESTAMP` - format: YYYY-MM-DD HH:MI:SS
- `YEAR` - format YYYY or YY

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- `DATE` - format YYYY-MM-DD
- `DATETIME` - format: YYYY-MM-DD HH:MI:SS
- `SMALLDATETIME` - format: YYYY-MM-DD HH:MI:SS
- `TIMESTAMP` - format: a unique number

Note: The date types are chosen for a column when you create a new table in your database!

SQL Working with Dates

Look at the following table:

Orders Table

# OrderId	Aa ProductName	OrderDate
1	Geitost	@November 11, 2008
2	Camembert Pierrot	@November 9, 2008
3	Mozzarella di Giovanni	@November 11, 2008
4	Mascarpone Fabioli	@October 29, 2008

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following `SELECT` statement:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

The result-set will look like this:

# OrderId	Aa ProductName	OrderDate
1	Geitost	@November 11, 2008
3	Mozzarella di Giovanni	@November 11, 2008

Note: Two dates can easily be compared if there is no time component involved!

Now, assume that the "Orders" table looks like this (notice the added time-component in the "OrderDate" column):

# OrderId	Aa ProductName	OrderDate
1	Geitost	@November 11, 2008 1:23 PM
2	Camembert Pierrot	@November 9, 2008 3:45 PM
3	Mozzarella di Giovanni	@November 11, 2008 11:12 AM
4	Mascarpone Fabioli	@October 29, 2008 2:56 PM

If we use the same `SELECT` statement as above:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

we will get no result! This is because the query is looking only for dates with no time portion.

Tip: To keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

▼ Order By

The SQL ORDER BY Keyword

The `ORDER BY` keyword is used to sort the result-set in ascending or descending order.

The `ORDER BY` keyword sorts the records in ascending order by default. To sort the records in descending order, use the `DESC` keyword.

ORDER BY Syntax

```
SELECT COLUMN_1, COLUMN_2, ...
FROM TABLE_NAME
ORDER BY COLUMN_1, COLUMN_2, ... ASC|DESC;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDER BY Example

```
SELECT *
FROM Customers
ORDER BY Country
```

ORDER BY Examples

```
-- DESC
SELECT *
FROM Customers
ORDER BY Country DESC;

-- Order by Columns
SELECT * FROM Customers
ORDER BY Country, CustomerName;

-- Order by Columns ASC and DESC
SELECT *
FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

▼ Case

The `CASE` statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the `ELSE` clause.

If there is no `ELSE` part and no conditions are true, it returns NULL.

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  WHEN conditionN THEN resultN
  ELSE result
END;
```

Example

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

```
SELECT OrderID, Quantity,
CASE
  WHEN Quantity > 30 THEN 'The quantity is greater than 30'
  WHEN Quantity = 30 THEN 'The quantity is 30'
```

```
  ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

▼ Exists

The `EXISTS` operator is used to test for the existence of any record in a subquery.

The `EXISTS` operator returns TRUE if the subquery returns one or more records.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Example

Below is a selection from the "Products" table in the Northwind sample database:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi	Tokyo	100	Japan

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);
```

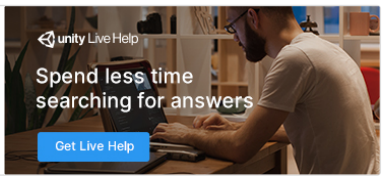
▼ Unity + Databases

▼ Understanding the Issue

Unity and MySQL Connector?

Seems Unity just wont support the new versions of MySQL itself. But I did manage to get it up and running. The MySQL 8 Server/Connector/NET parts will not work in Unity 2019.

https://answers.unity.com/questions/1785533/unity-and-mysql-connector.html



▼ Setup MySQL Unity

▼ ODBC

▼ Tested On

- Unity 2020.3.30
- Unity 2021.3

1. Download ODBC Driver here.
2. Install the ODBC Driver.
3. Find the ODBC Driver name
    a. Open Start Menu
    b. Search for Administrator Tools
    c. Open ODBC Data Sources 64bit
    d. Click Add
    e. Copy the Driver name of the Driver that you see, for example mine says “MySQL ODBC 8.0 Unicode Driver”
4. Open MAMP, XAMPP, MySQL Server or whatever MySQL instance you have available
    a. Create a Database called test
    b. Create a Table in that Database, name if whatever you want.
5. Open Unity
    a. Create a Script called DatabaseManager
    b. Overwrite it with the following code - `update the script with your connection information`
    c. For connection examples please have a look here

```
public class DatabaseManager : MonoBehaviour
{
    #region VARIABLES

    [Header("Connection Properties")]
    public string Host = "localhost";
    public string Port = "3306";
    public string User = "root";
    public string Password = "root";
    public string Database = "test";

    #endregion

    #region UNITY METHODS

    private void Start()
    {
        Connect();
    }

    #endregion

    #region METHODS

    private void Connect()
    {
        try
        {
            string connectionString = "DRIVER={MySQL ODBC 8.0 Unicode Driver};" +
                                      $"SERVER={Host};" +
                                      $"PORT={Port};" +
                                      $"DATABASE={Database};" +
                                      $"UID={User};" +
                                      $"PWD={Password};";
            using (OdbcConnection connection = new OdbcConnection(connectionString))
            {
                connection.Open();
                print("ODBC - Opened Connection");
            }
        }
        catch (OdbcException exception)
        {
            print(exception.Message);
        }
    }

    #endregion
}
```

d. Attach the script to an Object.

e. Press Play, you should get a log message that says “Open Connection”.

6. Start Scripting your SQL Queries

▼ .NET Connector Plugin

▼ Notes

- For Unity 2021.3 the working and tested version for the MySQL .Net Connector Plugin is v8.0.28
- The version of the .Net plugin that was tested in Unity 2021.3 was v4.5.2 an v4.8, both work!
  - v4.5.2 - delete the MySql.Data.EntityFrameWork.dll and .xml including the MySql.Web.dll and .xml files
  - v4.8 - delete the MySql.Web.dll and .xml files
- For Unity 2020.3.30 and possibly older(until LTS 2019), the working and tested version for MySQL .Net Connector Plugin is v6.9.11
  - The .Net plugin that was tested in Unity 2020.3.30 was v4.0 and v4.5, both work!
    - v4.x - delete everything except the MySql.Data.dll file
- If the MySQL plugin v6.9.11 does not work for your older version of Unity(older than Unity 2020), then download an older version of the MySQL .Net Connector Plugin. Try v6.9.5 as suggested here

▼ Tested On

- Unity 2020.3.30
- Unity 2021.3

1. Find the .NET Connector Plugin here
2. Choose .Net & Mono for Operating System.
3. For Unity 2020.3 download version 6.9.11 for Unity 2021.3 download version 8.0.28
4. For **Unity 2020.3**
   a. Navigate to the zip you downloaded [version 6.9.11]
   b. Extract the 4.0 or 4.5 folder
   c. Open the folder you extracted and Delete all the plugins except MySql.Data.dll
   d. Open Unity
      i. Go to Project Settings, Player, Other Settings, and Change the API Compatibility Level to .Net Framework 4.x
      ii. Create a Plugins folder in Unity
      iii. Drag and Drop the 4.0 or 4.5 folder you extracted into the Plugins folder
5. For **Unity 2021.3**
   a. Navigate to the zip you downloaded [version 8.0.28]
   b. Extract the 4.5.2 or the 4.8 folder
   c. Open the Folder you extracted and Delete the MySql.Data.EntityFramework.dll, MySql.Data.EntityFramework.xml and the MySql.Web.dll and MySql.Web.xml files
   d. Open Unity
      i. Go to Project Settings, Player, Other Settings, and Change the API Compatibility Level to .Net Framework 4.x
      ii. Create a Plugins Folder in Unity
      iii. Drag and Drop the 4.5.2 or the 4.8 folder you extracted into the Plugins folder

6. Open Unity
    a. If you did not change the API Compatibility Level to .Net Framework 4.x please do so
    b. If you did not Create a Plugins folder, create it or follow the instructions above for the Unity Version you are using
    c. If you did not Drag and Drop the folder you extracted into the Plugins folder, please do so
    d. Create a Script called DatabaseManager
    e. Overwrite the default code with the Following script - `update the script with your connection information`
    f. For connection examples please have a look here or here

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

public class DatabaseManager : MonoBehaviour
{
    #region VARIABLES

    [Header("Database Properties")]
    public string Host = "localhost";
    public string User = "root";
    public string Password = "root";
    public string Database = "test";

    #endregion

    #region UNITY METHODS

    private void Start()
    {
        Connect();
    }

    #endregion

    #region METHODS

    private void Connect()
    {
        MySqlConnectionStringBuilder builder = new MySqlConnectionStringBuilder();
        builder.Server = Host;
        builder.UserID = User;
        builder.Password = Password;
        builder.Database = Database;

        try
        {
            using (MySqlConnection connection = new MySqlConnection(builder.ToString()))
            {
                connection.Open();
                print("MySQL - Opened Connection");
            }
        }
        catch (MySqlException exception)
        {
            print(exception.Message);
        }
    }

    #endregion
}
```

    g. Attach the script to an Object
    h. Press Play, you should get a message that says “Opened Connection”.
7. Start scripting your SQL Queries.

▼ What you need to know

▼ Connection String

Before you can perform any SQL Commands on your SQL Server, you first need to create your Connection String.

Your connection string will consist of the following:

- Host/Server Link
- Username
- Password
- Database
- Port

A typical Connection String in C# will look like this:

`“server=Localhost;userid=root;password=root;database=your_database”`

You can find examples of connection strings here.

▼ Connect to SQL Server

Once you have your Connection String, you can now connect to your SQL Server.

For MySQL your connection code would look something like this:

```
MySqlConnection connection = new MySqlConnection("server=Localhost;user id=root;password=root;database=your_database");
connection.Open();
```

If you are using the .Net SQL Client in Unity you can try the following:

```
SqlConnection connection = new SqlConnection("server=Localhost;user id=root;password=root;database=your_database");
connection.Open();
```

After you are done executing your SQL Queries, you need to Close and Dispose of your Connection Object. This will close and cleanup your connection object from memory.

```
connection.Close();
connection.Dispose();
```

Note! The Connection object is disposable. This means that you can use the “using” statement to create your Connection Object and when code exits out of the using statement your object will automatically be disposed.

```
using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
{
  connection.Open();
}
// The connection object is disposed at this point
```

▼ Query

After you have established a connection to your database, you can start executing queries.

Your query would look similar to the one you write as an SQL statement.

Please make reference to the above SQL Examples for Create, Insert, Get, Update, Delete

```
// Insert
string query = $"INSERT INTO TABLE_VALUE (Username, Password) VALUES (USERNAME_VALUE, PASSWORD_VALUE)";

// Get
string query = $"SELECT * FROM TABLE_VALUE WHERE Username = USERNAME_VALUE AND Password = PASSWORD_VALUE";
string query = $"SELECT * FROM YOUR_TABLE";

// Update
string query = $"UPDATE YOUR_TABLE SET MONEY = MONEY_VALUE WHERE ID = ID_VALUE";

// Delete
string query = $"DELETE FROM YOUR_TABLE WHERE ID = ID_VALUE";
```

In SQL, there is a security risk in creating SQL Queries that are not Parameterized. By not creating creating Parameterized Queries, your Database is open to SQL Injections.

What is SQL Injection:

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

How to Create a Parameterized Query

```
string connectionString = "server=Localhost;user id=root;password=root;database=your_database";
using (MySqlConnection connection = new MySqlConnection(connectionString))
{
    connection.Open();
    string tableName = "";
    string username = "";
    string password = "";
    string query = $"INSERT INTO {tableName} (Username, Password) VALUES (@Username, @Password)";
}
```

How to Add values to your Parameterized Queries using a SQL Command Object.

```
string connectionString = "server=Localhost;user id=root;password=root;database=your_database";
using (MySqlConnection connection = new MySqlConnection(connectionString))
{
    connection.Open();
    string tableName = "";
    string username = "";
    string password = "";
    string query = $"INSERT INTO {tableName} (Username, Password) VALUES (@Username, @Password)";

    using (MySqlCommand command = new MySqlCommand(query, connection))
    {
        command.Parameters.AddWithValue("@Username", username);
        command.Parameters.AddWithValue("@Password", password);
    }
}
```

▼ Execute Command

Executing Commands takes your SQL Query and sends it to the SQL Server for Execution.
There are a few Variations of the Execution Query

- ExecuteNonQuery() - Returns the number of rows changed; Used when Executing CREATE, INSERT, UPDATE and DELETE based Queries
- ExecuteReader() - Returns the number of Rows in a Table; Used when Executing SELECT based Queries
- ExecuteScalar() - Gets the returned value from the Database

```
// Execute Non Query when Inserting into a Database
// Also used for CREATE, INSERT, UPDATE and DELETE
using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
{
    connection.Open();
    string tableName = "";
    string username = "";
    string password = "";
    string query = $"INSERT INTO {tableName} (Username, Password) VALUES (@Username, @Password)";

    using (MySqlCommand command = new MySqlCommand(query, connection))
    {
        command.Parameters.AddWithValue("@Username", username);
        command.Parameters.AddWithValue("@Password", password);

        int result = command.ExecuteNonQuery();
        if(result < 0)
          print("Nothing changed!")
    }
}

// ExecuteReader when Getting/Selecting data from a Database
using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
{
```

```
    connection.Open();
    string tableName = "";
    string username = "";
    string password = "";
    string query = $"SELECT * FROM {tableName} WHERE Username = @Username AND Password = @Password";

    using (MySqlCommand command = new MySqlCommand(query, connection))
    {
        command.Parameters.AddWithValue("@Username", username);
        command.Parameters.AddWithValue("@Password", password);

        using (MySqlDataReader reader = command.ExecuteReader())
        {
            player = new Player()
            {
                Username = username
            };

            while (reader.Read())
            {
                player.ID = reader.GetInt32("ID");
                player.Money = reader.GetInt32("Money");
            }
            // No More to Read
        }
        // Clean up reader
    }
    // clean up the command
    print($"Get Player - {username}.");
}
```

▼ Create

```
public void AddNewPlayer(string username, string password)
{
    try
    {
        using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
        {
            connection.Open();
            string query = $"INSERT INTO {TableName} (Username, Password) VALUES (@Username, @Password)";

            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Username", username);
                command.Parameters.AddWithValue("@Password", password);

                command.ExecuteNonQuery();
            }
        }

        print($"Insert Player - {username}");
    }
    catch (MySqlException exception)
    {
        print(exception.Message);
    }
}
```

▼ Get

```
public Player GetPlayer(string username, string password)
{
  try
  {
      using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
      {
          connection.Open();
          string query = $"SELECT * FROM {TableName} WHERE Username = @Username AND Password = @Password";

          using (MySqlCommand command = new MySqlCommand(query, connection))
          {
              command.Parameters.AddWithValue("@Username", username);
              command.Parameters.AddWithValue("@Password", password);

              using (MySqlDataReader reader = command.ExecuteReader())
              {
                  while (reader.Read())
                  {
```

```
                    print($"ID - {reader.GetInt32("ID")} : Username - {reader.GetString("Username")} : Money - {reader.GetInt3
                }
            }
        }

        print($"Get Player - {username}.");
    }
  }
  catch (MySqlException exception)
  {
      print(exception.Message);
  }
}
```

▼ Update

```
public void UpdatePlayer(int id, int money)
{
    try
    {
        using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
        {
            connection.Open();
            string query = $"UPDATE {TableName} SET MONEY = @Money WHERE ID = @ID";

            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@ID", id);
                command.Parameters.AddWithValue("@Money", money);
                command.ExecuteNonQuery();
            }

            print($"Updated Player - {id} : {money}.");
        }
    }
    catch (MySqlException exception)
    {
        print(exception.Message);
    }
}
```

▼ Delete

```
public void DeletePlayer(int id)
{
    try
    {
        using (MySqlConnection connection = new MySqlConnection(_connectionString.ToString()))
        {
            connection.Open();
            string query = $"DELETE FROM {TableName} WHERE ID = @ID";

            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@ID", id);
                command.ExecuteNonQuery();
            }

            print($"Deleted Player - {id}");
        }
    }
    catch (MySqlException exception)
    {
        print(exception.Message);
    }
}
```

▼ Publishing a Game