|  |
| --- |
|  |
| The Noise Database Documentation |
| **Matthew Lindsay** |
|  |
|  |
|  |

|  |
| --- |
|  |

The database consists of two parts. The actual Database, and the Database Access Layer written in C#.

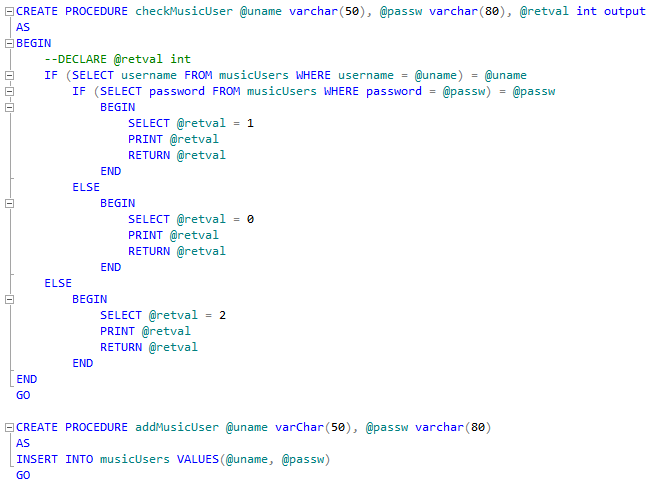
# The Database

## The Tables

The database itself is fairly simple. It consists of three tables: musicUsers (which contains a list of users and their passwords), directories (which contains the directories on the server users have access to), and userDirectories (which connects each user to any and all directories they have access to). The real work is done in the procedures.

## The Procedures

There are only two procedures, pictured below:



The procedures fulfills all of the logic that needs to be done to check and add users without having of it be in the C# access layer. This makes the database both more secure and user friendly by allowing for simple procedural calls, and not relying on typed out SQL commands in the C# file. This method also prevents in line SQL, preventing malevolent users from accessing secure data.

# The Database Access Layer

This is the C# file made as an in between for the server and the Database. It allows the server to simply call functions and work with the returned information. It has chained constructors (thanks, Tom) that allow a user to log in through windows authentication or with a username and password.

The DAL has only two functions that the server calls. Add user, and Validate User. These are to add a user to the database, and check to see if a user is already in the database.

## Add User:

public UserAddResult addUser(LoginData sentUser)

{

//GlobalEnumerations.UserValidationResult result;

UserAuthenticationResult preCheck;

preCheck = validateUser(sentUser);

//If validateUser returns 1, user is not in database

if (preCheck == UserAuthenticationResult.InvalidUser)

{

using (SqlConnection connection = new SqlConnection(sqlStrBldr.ConnectionString))

{

SqlCommand command = new SqlCommand("addMusicUser", connection);

command.CommandType = CommandType.StoredProcedure;

SqlParameter userName = new SqlParameter("@uname", SqlDbType.VarChar);

userName.Direction = ParameterDirection.Input;

userName.Value = sentUser.username;

userName.Size = 50;

command.Parameters.Add(userName);

SqlParameter password = new SqlParameter("@passw", SqlDbType.VarChar);

password.Direction = ParameterDirection.Input;

password.Value = sentUser.password;

password.Size = 80;

command.Parameters.Add(password);

command.Connection.Open();

command.ExecuteNonQuery();

}

// Check user again to make sure things went well

UserAddResult result = UserAddResult.UnknownResult;

UserAuthenticationResult postCheck;

postCheck = validateUser(sentUser);

switch (postCheck)

{

case UserAuthenticationResult.UnknownResult:

break;

case UserAuthenticationResult.Success:

result = UserAddResult.Success;

break;

case UserAuthenticationResult.InvalidUser:

break;

case UserAuthenticationResult.InvalidPassword:

break;

default:

break;

}

return result;

}

else

{

return UserAddResult.AlreadyExists;

}

}

## Validate user:

public UserAuthenticationResult validateUser(LoginData sentUser)

{

try

{

using (SqlConnection connection = new SqlConnection(sqlStrBldr.ConnectionString))

{

SqlCommand command = new SqlCommand("checkMusicUser", connection);

command.CommandType = CommandType.StoredProcedure;

SqlParameter userName = new SqlParameter("@uname", SqlDbType.VarChar);

userName.Direction = ParameterDirection.Input;

userName.Value = sentUser.username;

userName.Size = 50;

command.Parameters.Add(userName);

SqlParameter password = new SqlParameter("@passw", SqlDbType.VarChar);

password.Direction = ParameterDirection.Input;

password.Value = sentUser.password;

password.Size = 80;

command.Parameters.Add(password);

SqlParameter returnValue = new SqlParameter("@retval", SqlDbType.Int);

returnValue.Direction = ParameterDirection.Output;

command.Parameters.Add(returnValue);

command.Connection.Open();

command.ExecuteNonQuery();

return (UserAuthenticationResult)(int)command.Parameters["@retval"].Value;

}

}

catch

{

return UserAuthenticationResult.UnknownResult;

}

}

These return the enumeration UserAuthenticationResult and UserAddResult so the server knows exactly what to do with the result.