21. ADO.Net and SQL Database Creation

Watch these video series

The heart of creating databases is the language most used to access and manipulate the data. That language is SQL - Structured Query Language.

SQL is just as important as any programming language you want to learn. Every programmer needs to have a good grasp of SQL.

https://msdn.microsoft.com/en-us/library/aa302325.aspx Learn ADO.net

Introduction to SQL

This course is a gentle introduction to Structured Query Language (SQL).

http://pluralsight.com/training/Courses/TableOfContents/introduction-to-sql

Database Fundamentals

Developing Microsoft SQL Server Databases

Microsoft MTA: Database Administration Fundamentals

This is a practical course for someone new to relational databases that moves you through basic concepts right into real-world usage, demonstrating core tasks in Microsoft SQL Server itself.

http://pluralsight.com/training/Courses/TableOfContents/database-admin-fundamentals

More Advanced – Integration with Visual Studio

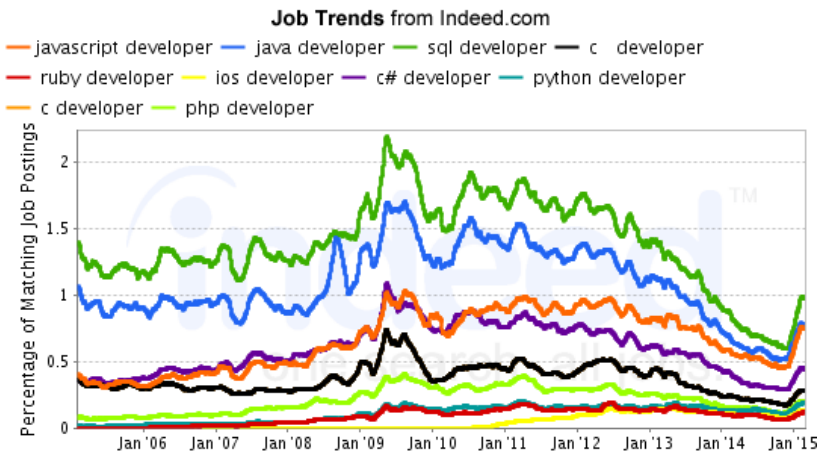
http://pluralsight.com/training/Courses/TableOfContents/sql-server-tsql

ADO.Net Very important in C#

http://pluralsight.com/training/Courses/TableOfContents/adodotnet-fundamentals

Learn SQL

http://www.tutorialspoint.com/sql/ https://www.codeschool.com/courses/try-sql



indeed.com

TSQL Cheat Sheet

String Functions

Exact Numerics

Bit	Tinyint	Smallint
Bigint	Decimal	Money
Numeric		

Approximate Numerics

Float	Real

Date and Time

Smalldatetime	Datetime
Timestamp	

Strings

Char	Varchar	Text

Unicode Strings

Nchar	Nvarchar	Ntext

Binary Strings

Binary	Image
Varbinary	

Miscellaneous

Cursor	Table	Xml
Sql_variant		

Common Functions

AVG()	Returns the average value
COUNT()	Returns the number of rows
FIRST()	Returns the first value
LAST()	Returns the last value
MAX()	Returns the largest value
MIN()	Returns the smallest value
SUM()	Returns the sum

UCASE() - Converts a field to upper case
LCASE() - Converts a field to lower case
MID() - Extract characters from a text field
LEN() - Returns the length of a text field
ROUND() - Rounds a numeric field to the number of decimals specified
NOW() - Returns the current system date and time
FORMAT() - Formats how a field is to be displayed

Date Functions

DATEADD (datepart, number, date)
DATEDIFF (datepart, start, end)
DATENAME (datepart, date)
DATEPART (datepart, date)
DAY (date)
GETDATE()
GETUTCDATE()
MONTH (date)
YEAR (date)

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	**Description**
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Note: In some versions of SQL the <> operator may be written as !=

Logical processing order of select

FROM table
ON join condition
JOIN table
WHERE clauses
GROUP BY columns
WITH CUBE / WITH ROLLUP
HAVING condition
SELECT columns
DISTINCT
ORDER BY columns
TOP % or number

CTES - Common Table Expressions

WITH cteName (columnList)
AS (**SELECT** statement)
SELECT columns
FROM cteName
INNER JOIN table **ON** condition

Recursive CTEs

WITH cteName (columnList)
AS (-- Anchor statement:
SELECT columns **FROM** table**…**
UNION ALL
Recursion statement:
SELECT columns **FROM** table**…**
INNER JOIN cteName **ON …**

```
)
SELECT columns
FROM cteName
```

Over and partition by

```
/* Aggregate functions include COUNT,
MIN, MAX, AVG, ROW_COUNT(), etc. */
SELECT
agg_func(col1) OVER(),
agg_func(col1)
OVER(PARTITION BY col2),
columns
FROM table…
```

Create a Stored Procedure

```
CREATE PROCEDURE name
@variable AS datatype = value
AS
-- Comments
SELECT * FROM table
GO
```

Create a Trigger

```
CREATE TRIGGER name
ON
table
FOR
DELETE, INSERT, UPDATE
AS
-- Comments
SELECT * FROM table
GO
```

22. Using SQL Server Management Studio

Setting up SQL Server Management Studio

We are running **Microsoft's SQL Server Management Studio** which can be easily downloaded for free from the Microsoft website. Get the **SQLEXPWT.exe** version.

https://www.microsoft.com/en-us/download/details.aspx?id=42299

Express with Tools (SQLEXPRWT_Architecture_Language.exe)

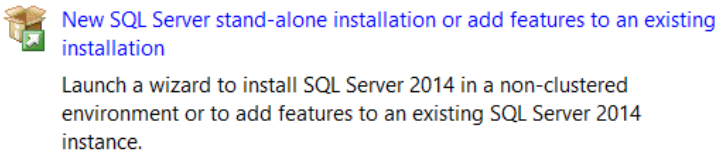
This package contains everything needed to install and configure SQL Server as a database server including the full version of SQL Server 2014 Management Studio

Choose the download you want

File Name	Size
ExpressAndTools 32BIT\SQLEXPRWT_x86_ENU.exe	840.8 MB
ExpressAndTools 64BIT\SQLEXPRWT_x64_ENU.exe	833.2 MB

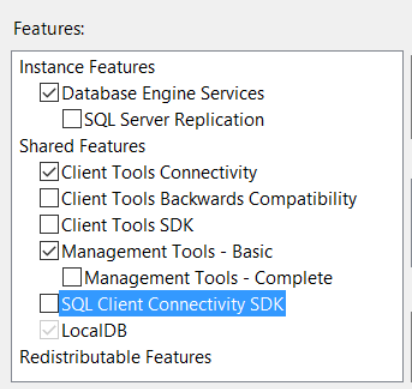
Don't download the whole SQL Server, its not needed and will make your computer run slow.

Run the installer

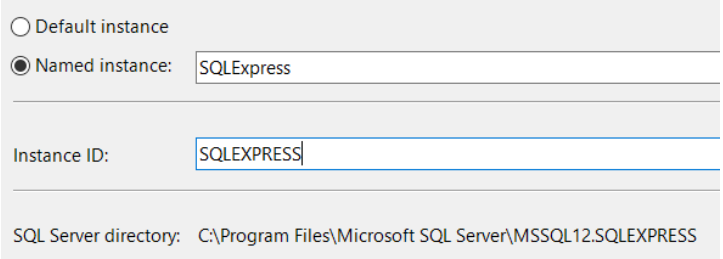


These next features are what you can change. Otherwise just click NEXT to bounce through the windows.

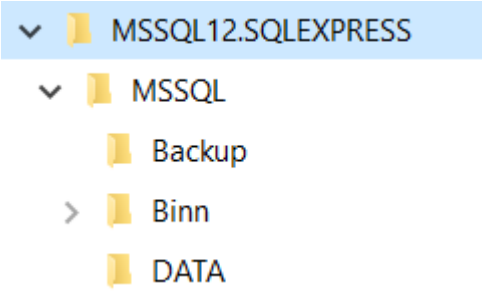
I think you can get by with just these selected Instance Features, otherwise just Click Select All and install it (I have space issues)



This option will create the folder with the name of the Server Instance on it, You can call it what you like, but inside of it will be all your databases.



Here is where your Databases will be stored as a default, in the Data Folder.

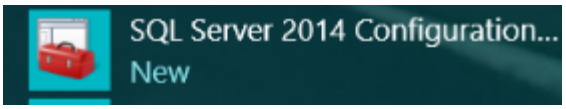


When you have installed it you have the following shortcuts loaded onto your Start menu →

Click on SQL Management Studio to activate the SQL Server

.

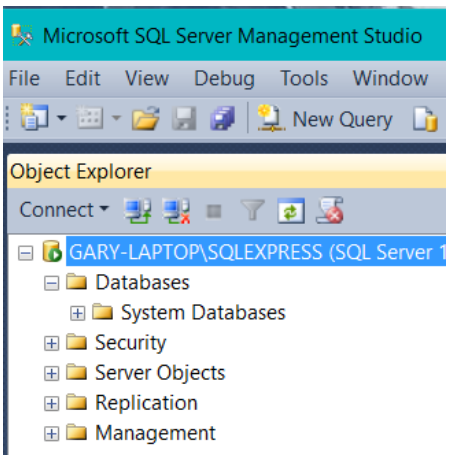
You will want this



if your server doesn't like to run.

Start the 

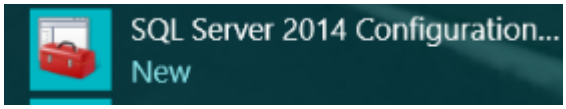
and it will load the program.



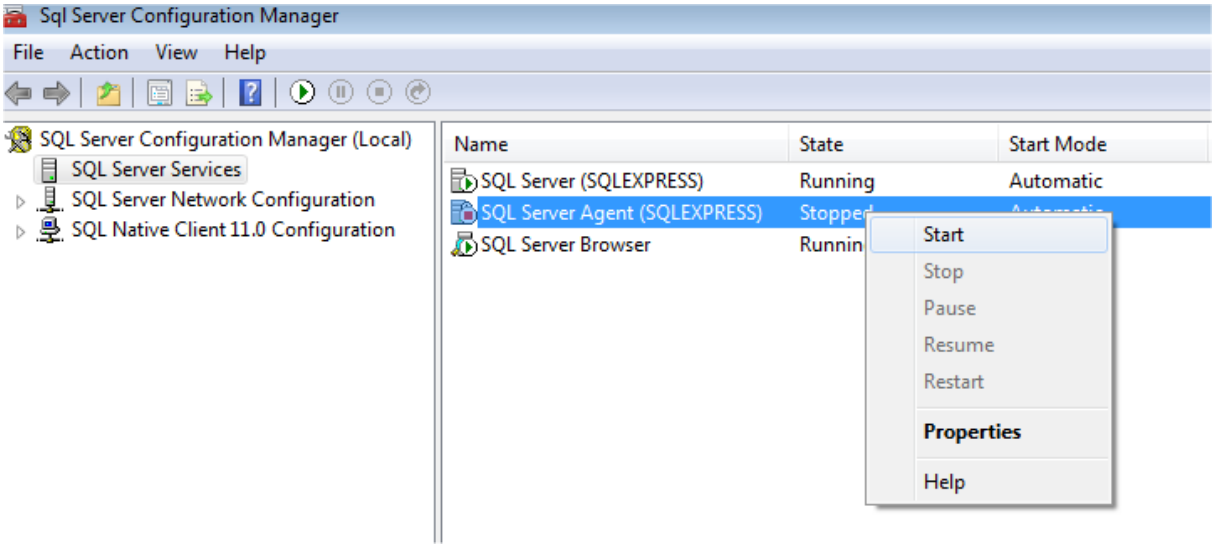
The top line is the name of your **computer\SQLExpress**

If you DON"T see this then you need to do the following.

Go to the SQL Server Configuration Manager back on your Start Menu bar



Right Click on the Server Agent and start it. Turn on the Server Agent.

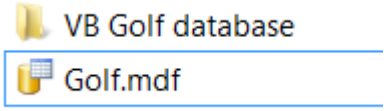


Hopefully that will get you going.

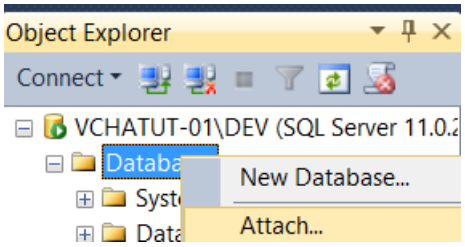
If you see nothing in that window, you are really screwed, the wrong version of SQL Management Studio has been installed and the Server part is missing. Uninstall your version and get the version mentioned earlier.

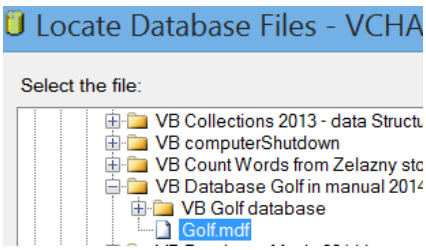
How to Attach an existing Database

I put my databases in the same folder as my project.
So move it there first or do it later.

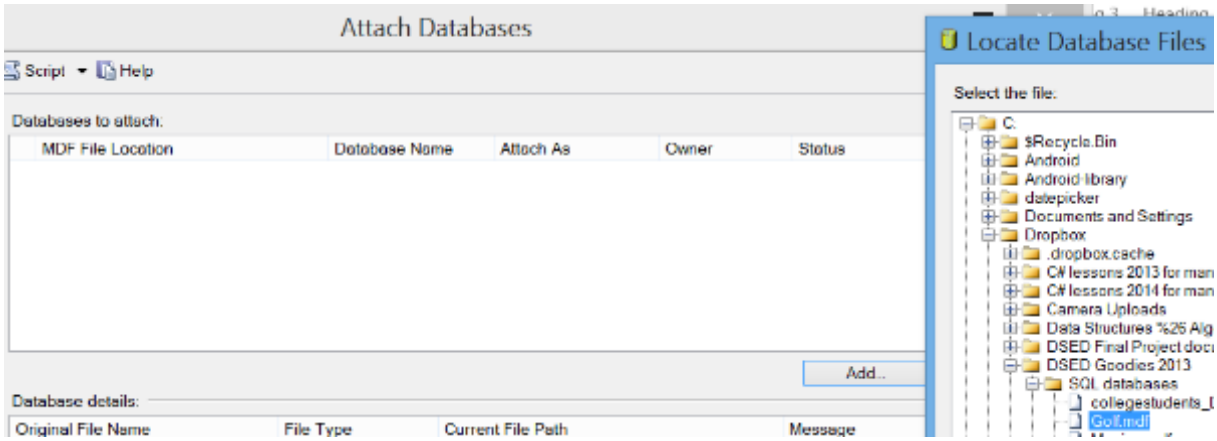


Right Click on the Database folder and then Choose **Attach**.

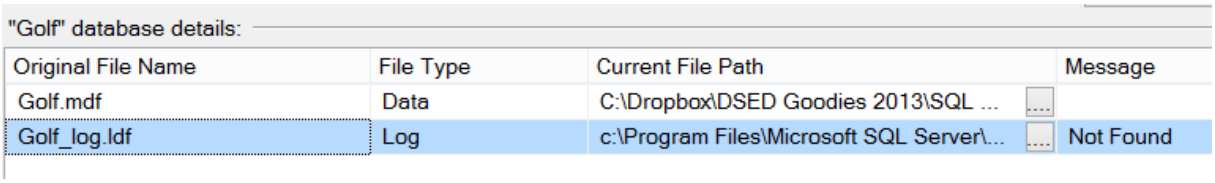




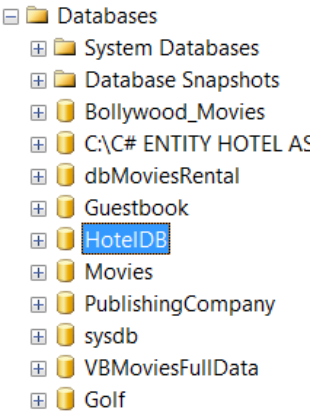
Click **Add** and find your Database.



The program will also look for the Log file, but won't find it, as there is only the Main file. (That's not a problem), just click **Remove** to remove it. Then Click **OK** to connect it.



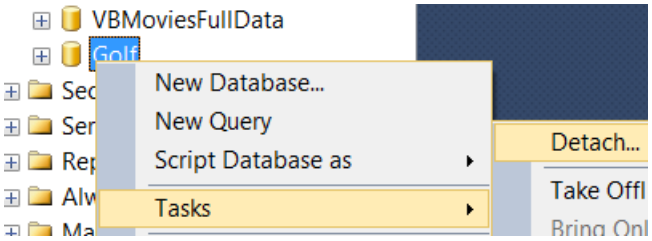
The Database should be Attached, although you might have to right click and choose Refresh on the Database folder to see it.



How to Remove a Database from SQLSMS

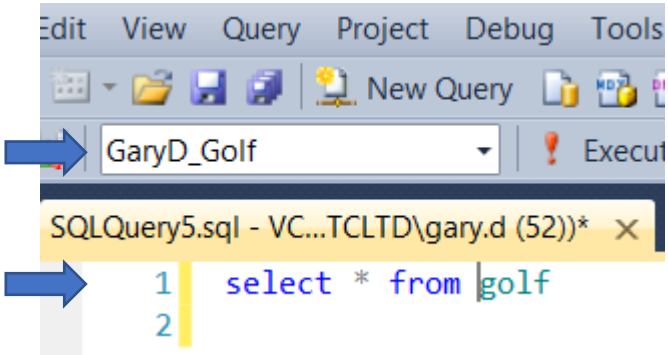
To remove a Database from the Server **Right click** on it, Go **Tasks**, then **Detach.** This doesn't delete the database, it's still in its folder, it just removes it from the server.

YOU WILL NEED TO DO THIS IF YOU WANT TO COPY OR MOVE IT TO ANOTHER PLACE.



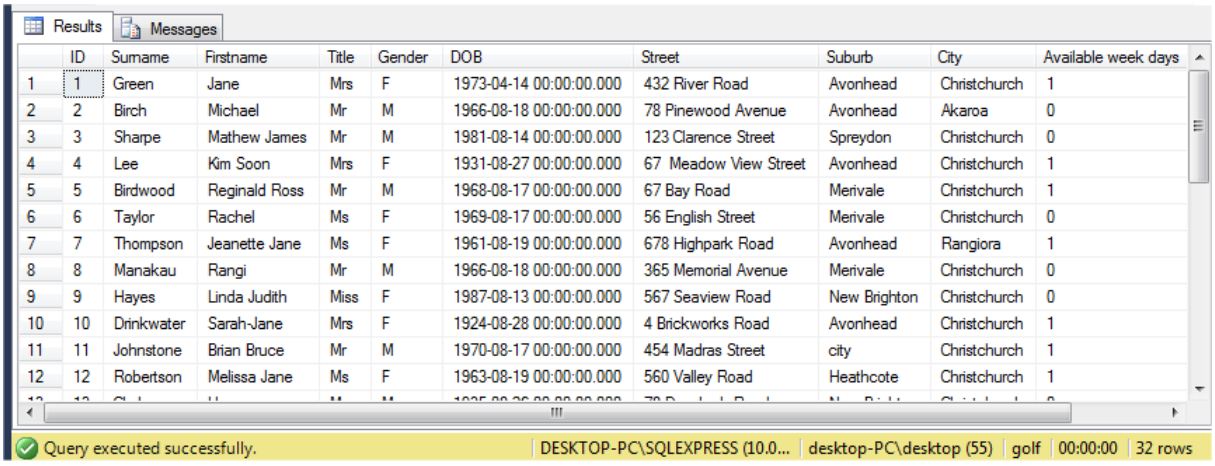
How to Run a Query in SQL Express Management Studio

Click on New Query and in the window below type **SELECT * FROM golf**. This will select all the fields from the Golf table. Make sure you choose the Golf Database in the pull down.



Then click Execute to see your fields.

You should have "Query Executed Successfully" and 32 rows at the bottom of the screen



Results | Messages

	ID	Surname	Firstname	Title	Gender	DOB	Street	Suburb	City	Available week days
1	1	Green	Jane	Mrs	F	1973-04-14 00:00:00.000	432 River Road	Avonhead	Christchurch	1
2	2	Birch	Michael	Mr	M	1966-08-18 00:00:00.000	78 Pinewood Avenue	Avonhead	Akaroa	0
3	3	Sharpe	Mathew James	Mr	M	1981-08-14 00:00:00.000	123 Clarence Street	Spreydon	Christchurch	0
4	4	Lee	Kim Soon	Mrs	F	1931-08-27 00:00:00.000	67 Meadow View Street	Avonhead	Christchurch	1
5	5	Birdwood	Reginald Ross	Mr	M	1968-08-17 00:00:00.000	67 Bay Road	Merivale	Christchurch	1
6	6	Taylor	Rachel	Ms	F	1969-08-17 00:00:00.000	56 English Street	Merivale	Christchurch	0
7	7	Thompson	Jeanette Jane	Ms	F	1961-08-19 00:00:00.000	678 Highpark Road	Avonhead	Rangiora	1
8	8	Manakau	Rangi	Mr	M	1966-08-18 00:00:00.000	365 Memorial Avenue	Merivale	Christchurch	0
9	9	Hayes	Linda Judith	Miss	F	1987-08-13 00:00:00.000	567 Seaview Road	New Brighton	Christchurch	0
10	10	Drinkwater	Sarah-Jane	Mrs	F	1924-08-28 00:00:00.000	4 Brickworks Road	Avonhead	Christchurch	1
11	11	Johnstone	Brian Bruce	Mr	M	1970-08-17 00:00:00.000	454 Madras Street	city	Christchurch	1
12	12	Robertson	Melissa Jane	Ms	F	1963-08-19 00:00:00.000	560 Valley Road	Heathcote	Christchurch	1

Query executed successfully. | DESKTOP-PC\SQLEXPRESS (10.0... | desktop-PC\desktop (55) | golf | 00:00:00 | 32 rows

Right click on the Database then rename it to **Golf**

Owing to issues with the dates later on, set the DOB field to the NvarChar(50) or smaller

Column Name	Data Type	Al
ID	int	
Surname	nvarchar(50)	
Firstname	nvarchar(50)	
Title	nvarchar(50)	
Gender	nvarchar(50)	
DOB	nvarchar(50)	
Street	nvarchar(50)	

Set the ID field to be Auto Incrementing.

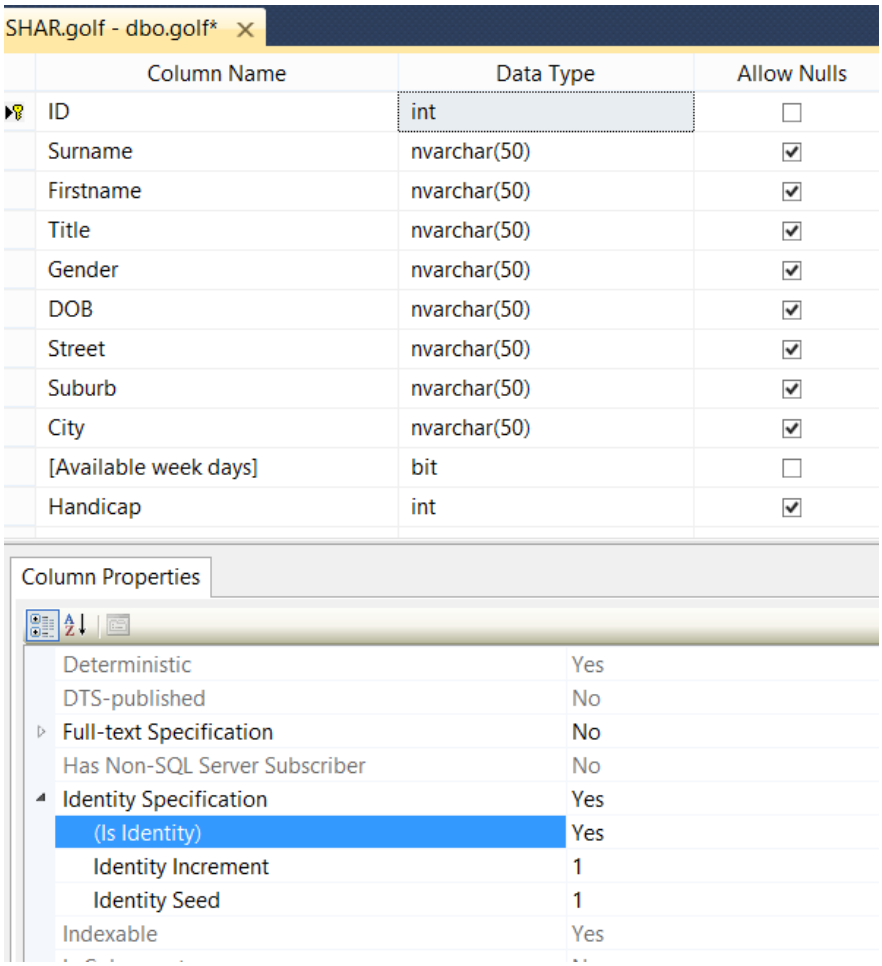
Although we will cover it later in the next exercise, you could also set the ID to be a Primary Key  with the Identity Specification set to **Yes**. This means that you won't have to set the ID any more as each new entry will automatically have a new one.

The drawback with this is if you try to pass data to the ID field you will get an error.

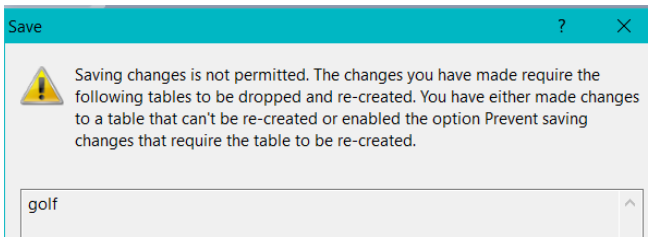
However if you do add it, and why not, then you **don't have to give the Primary Key a value** each time you add a new row of data to the table.

Right click on the ID Field and choose **Modify**.

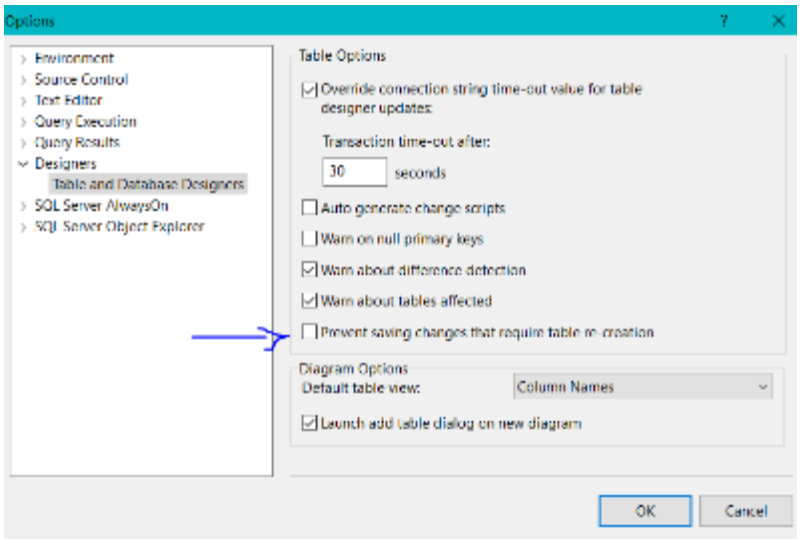
Set **Is Identity** to Yes.



When you try to save it however you get an error so cancel out of the error and change the following.



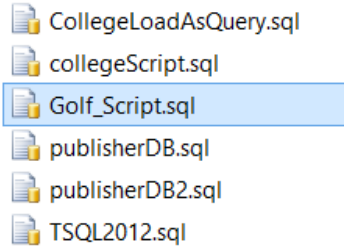
To fix this go **Tools / Options** and **Table and Database Designers**.



Untick the field above and click OK.

Now you can save your Table.

How to Run an SQL Script to create a Database (Reference)



If you have an SQL script you can drag it into the SQL window to load it.

Find the script Golf_Script.sql and drag it into the window.

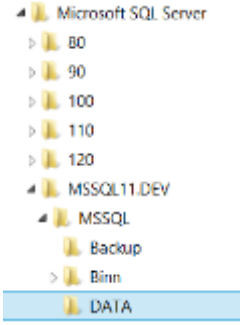
Here is the script when loaded in the program. You can scroll through it and see the data that is being held.

This is a wonderful way to move small databases around, by scripting them out and then running the script.

Golf_Script.sql - V...(ATCLTD\gary.d (55))

```
USE [master]
GO
/****** Object:  Database [GaryD_GolfFromScript]    Script Date: 26/03/2014 1:35:05 p.m. ******/
CREATE DATABASE [GaryD_Golf] ON  PRIMARY
( NAME = N'GaryD_Golf', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA
\GaryD_Golf.mdf' , SIZE = 5000KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'GaryD_Golf_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA
\GaryD_Golf_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [GaryD_Golf] SET COMPATIBILITY_LEVEL = 100
```

However before you can run your Script you have to **change the path to where you store your Databases.**

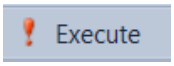


The path shown in the script is the path to the where that database was created. It will be different to your installed path as the server name at least can be different.

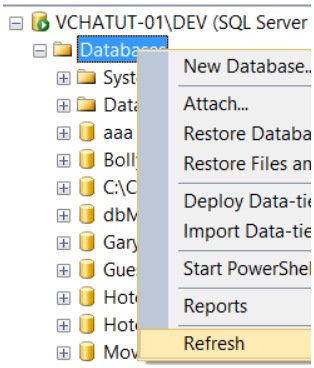
Here is the one from above `N'C:\Program Files\Microsoft SQL Server\MSSQL11.DEV\MSSQL\DATA\GaryD_Golf.mdf'`

This PC ▸ Windows (C:) ▸ Program Files ▸ Microsoft SQL Server ▸ MSSQL11.DEV ▸ MSSQL ▸ DATA

Databases are by default stored in the data folder in the folder with the name of your Server. Mine is MSSQL11.DEV so the path is actually the same however you need to replace it with the path to YOUR data file. The big difference might be just the name of the server.

Once you have changed that, then run the Execute 

and see if the Database is created.



If successful then right click on Database and click Refresh to see your new database. If you look in your Data folder you will find it stored there.

Here is a YouTube video on how to do it as well.

https://www.youtube.com/watch?v=olgJOG70-vg

Check that your data has arrived safely by running a simple SQL query

Why Can't SQL Server Management Studio Access The "My Documents" Folder

This seems to be a problem with some students machines.

http://superuser.com/questions/69879/why-cant-sql-server-management-studio-access-the-my-documents-folder-in-windo

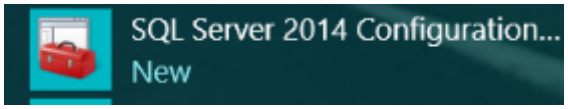
http://bidn.com/Blogs/unable-to-browse-to-database-or-backup-file-location-from-ssms

While the folder that contains my backup file is located within the folder sharrison, it appears as if there is not.

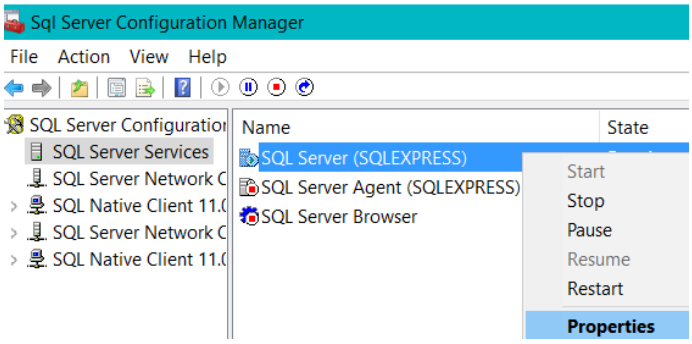
This is because the SQL Server account does not have permission to reach the folder you are trying to browse.

To give your SQL Server account access to this folder, we first have to verify the account used to login to SQL Server.

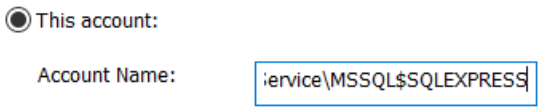
Open SQL Server Manager Configuration Manager and navigate to SQL Server Services.



Right-click SQL Server (MSSQLSERVER) and select properties.



On the Log On tab, you should see where the account to login as is specified.



NT Service\MSSQL$SQLEXPRESS

Once you know the account being used by SQL Server, give that account the needed permissions on your directory.

23. The Golf Project using ADO.net

What is ADO.Net overview

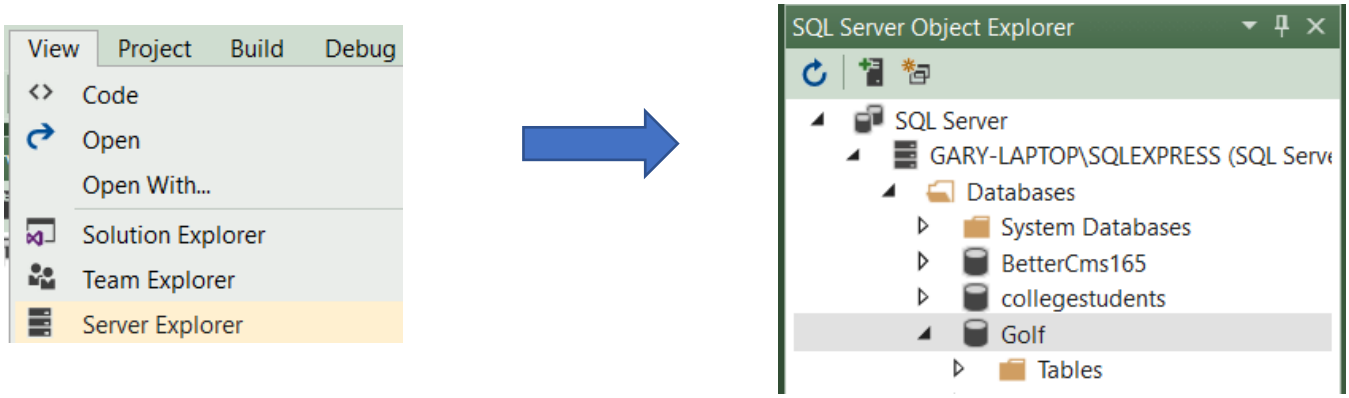
Learn ADO.net Read this!

In Visual Studio create a new project called Golf. From the earlier examples attach the Golf DB to your SQL Server. Below we then join the two together via a connection string.

How to find your Connection String.

Connection strings can be a pain in the neck, but there is an easy way to find out what your one is to connect to the Database you want.

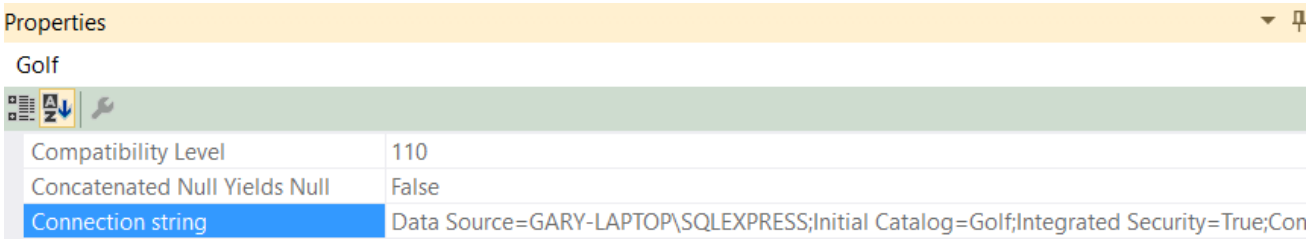
Open Server Explorer in Visual Studio go **View / Server Explorer**



Find your Database, right click and go to Properties.

Then just look for the Connection string, Copy and Paste the first bit into your program.

```
//Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=Golf;Integrated
Security=True;
```



In your code later you will add in your Connection String

```
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
```

The @ symbol tells the compiler that the following string is a string literal. Which means the \ won't be seen as an escape character, but as a part of the string.

The Data Source key indicates which server to access. In this case, the GARY-LAPTOP\, SQLEXPRESS value refers to the SQL Server Express Edition installation on the local workstation.

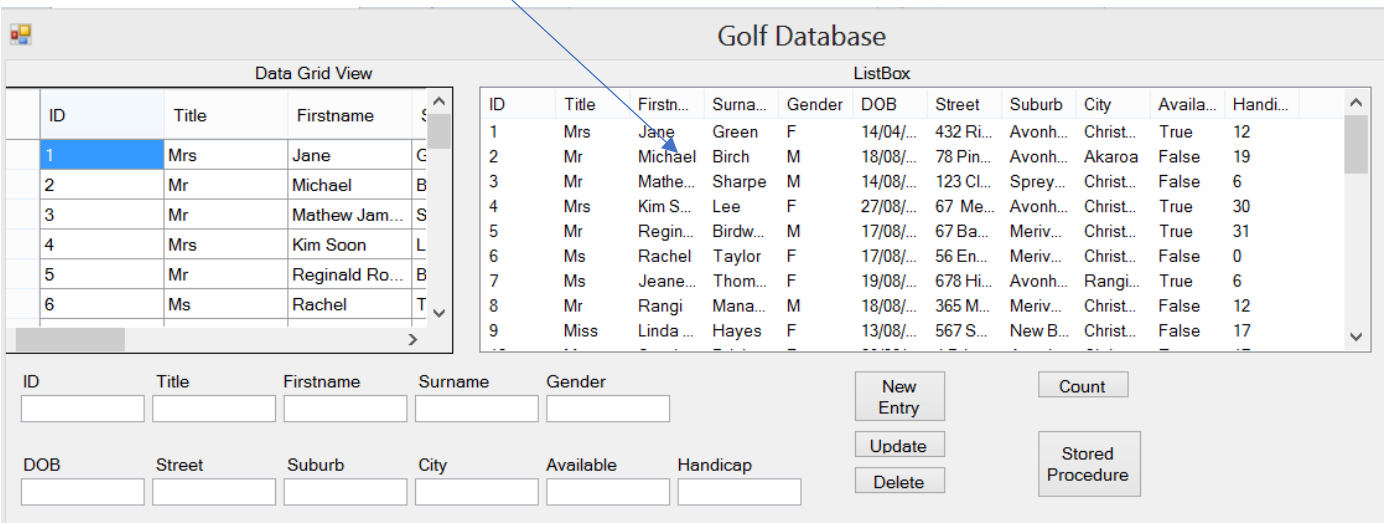
The Initial Catalog=golf key tells the connection which database within the hosted database engine to use as the default. In this sample string, golf is the name of the default database catalog to use. You must have the appropriate security credentials to access this database.

The Integrated Security=True key with a value of True tells ADO.NET to use your existing Microsoft Windows security credentials to access the database

Display data in a ListView

Let's create a **Listview** to load the data in so that we know everything is working OK.

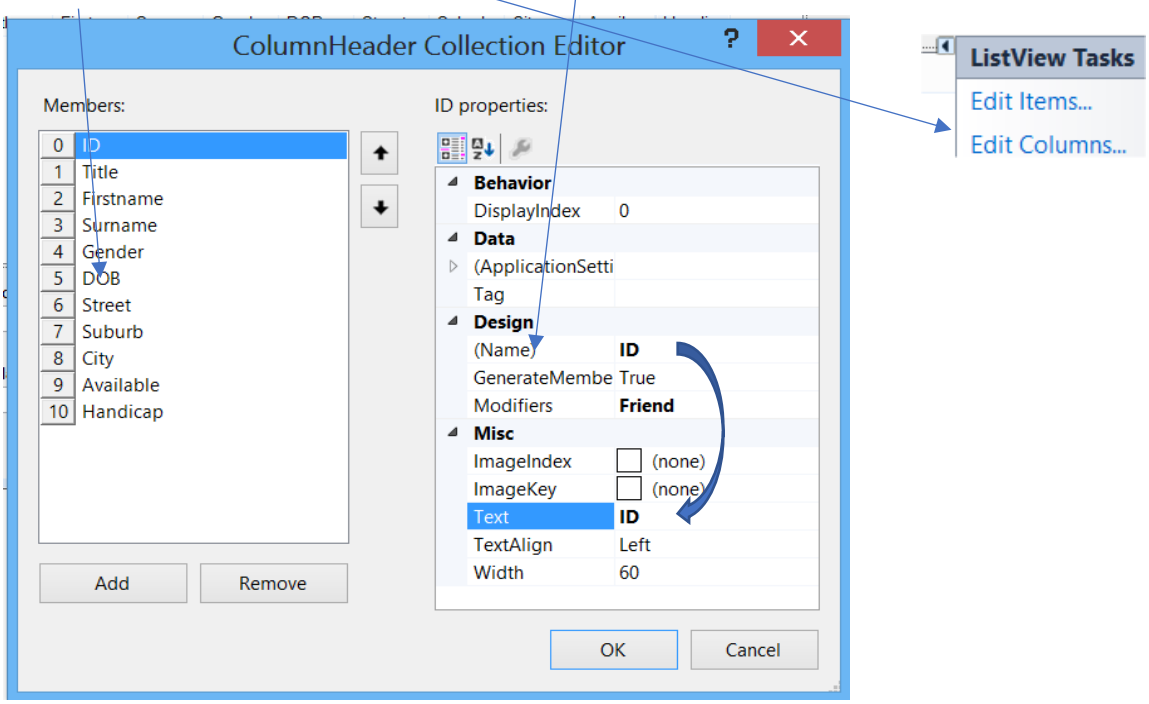
The Listview in **final project**, on the right, with the DataGrid on the left. We just want the Listview at this stage.



From the Toolbox drag on a ListView  and name it **LVGolf**

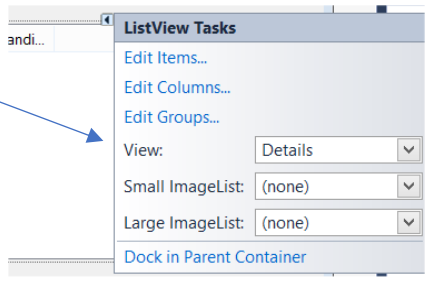
Create the Headings for the Listview.

In the **Edit Columns Option** name 10 columns with the appropriate names as shown below. I changed the TEXT property and the NAME property to the same.



In the **View** Property set it to **Details**

We need to add the code to read the data into a Listview.



Three things we need to do.

```
Using (SqlConnection linkToDB = new SqlConnection(ConnectionString)) {
linkToDB.Open();
// -----Do stuff with the data you get here.
}
```

1 Make a connection with the Database on the server by passing in the connection string to establish the connection.

This connection is wrapped in a using Statement, so that when the data has finished being moved, the connection is closed, and any unwanted resources are deleted.

```
using (SqlConnection connection = new SqlConnection(connectionString)) {
```

```
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
```

2 We need send a command to the server. In this case the command is send us back all the data in the golf table.

```
string queryString = "SELECT * FROM Golf ORDER by ID";

//read in the data with a datareader open the data connection
                SqlCommand Command = new SqlCommand(queryString, connection);
```

Down the pipe we have made we open the connection, Connection.Open();

3 Send a request to the database for data. Do stuff with it.

Get that data back and close the connection. Connection.Close();

The Code for Golf

(Really 90% of the following should be in a class, but that's in the future)

The columns are really pseudo columns. Although we have a whole bunch of columns you can't just specify them in a Listview as it only really holds one item down the page.

The code below declares two objects, one for the Command and one for the DataReader.

The Command object holds and executes the SELECT statement to send to the data source. SqlCommand Command = new SqlCommand(queryString, connection);

The DataReader is the object that retrieves the data from the result set that comes back from the SELECT statement. SqlDataReader reader = Command.ExecuteReader();

Using the ADO.NET DataReader

DataSets, DataTables, and DataReaders are used to retrieve records from data sources.

The DataReader object is a forward-only type of cursor that provides the fastest way to retrieve records from a data source.

Because its direction is limited to forward-only, it provides great performance for programmatically processing results or loading list boxes, combo boxes, etc.

You loop through each of the rows in the DataReader by invoking the Read method. reader.Read().

The Read method moves the cursor from one row to the next. After the Read method has executed, you can pass the name of the column you wish to retrieve to the Item property on the DataReader. This returns the actual data from that column.

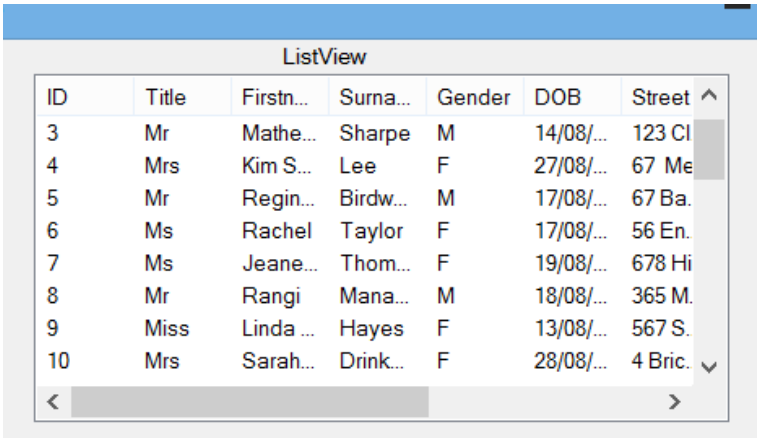
The data returned from the Item property comes back as an Object data type. Because the items you add to the list box are of type Object, no conversion is required when using the Add method of the Items collection on the list box.

In this manner, you continue looping until the Read method returns a False. This means you have hit the end of the rows of data that were returned from the SELECT statement.

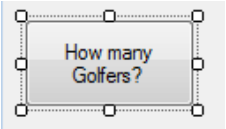
```
public void loadDatabase() {
  lvGolf.Items.Clear(); //clear out old data in the listbox, we will need this
later
           //load the connection string and pass it to the command
           string connectionString = @"Data Source=GARY-
LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated Security=True";

using (SqlConnection connection = new SqlConnection(connectionString)) {
string queryString = "SELECT * FROM Golf ORDER by ID";

//read in the data with a datareader open the data connection
SqlCommand Command = new SqlCommand(queryString, connection);
               connection.Open();
               SqlDataReader reader = Command.ExecuteReader();
               while (reader.Read()) {
  //add each row to the listbox
ListViewItem item = new ListViewItem(new[] {
reader["ID"].ToString(), reader["Title"].ToString(),
reader["Firstname"].ToString(), reader["Surname"].ToString(),
reader["Gender"].ToString(), reader["DOB"].ToString(), reader["Street"].ToString(),
reader["Suburb"].ToString(), reader["City"].ToString(), reader["Available week
days"].ToString(), reader["Handicap"].ToString()
                       });
                   LVGolf.Items.Add(item);
                   }
               reader.Close();
}
```

Because the columns in the ListView are not ‘real’ they only **look** like columns, you can’t operate on them as columns, you can’t click on them, or select a column, they are pretty useless as tools but just display data.

Run and see the data load.



Returning a Scalar (single) value

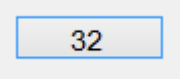


Create a new button. Call it btnCount

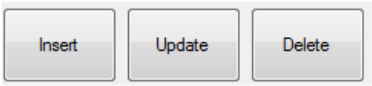
You create Scalar queries to return a **single value**. In this example we will use a simple COUNT to show **on the button** how many Golfers there are.

- Create a **SqlConnection** object and give it a **connection string**. SqlConnection Con = new SqlConnection();
- Create a **SqlCommand** object and assign to it the connection object you opened. Place an SQL statement into the Command object SqlCommand update = new SqlCommand(updatestatement, Con)
- Open the connection to the database. connection.Open();
- Execute the SQL statement by invoking the **ExecuteNonQuery** method on the Command object. update.ExecuteNonQuery();
- Close the Connection. connection.Close();

```
private void btnCount_Click(object sender, EventArgs e) {
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
// a simple Scalar query just returning one value.
          string queryString = "SELECT COUNT(ID) FROM Golf";

using (SqlConnection connection = new SqlConnection(connectionString)) {
              SqlCommand Command = new SqlCommand(queryString, connection);
              connection.Open();
              btnCount.Text = Command.ExecuteScalar().ToString();
              connection.Close();
              }
          }
```

Button will show 32 on the text field 

Insert /New Command



The **Insert command** has three main areas, the **Textboxes** txtFirstname.Text where you type in the data, the **Parameters** you pass the data to, and the **Database fields you want the data to go to**..

TextBox	goes to	**Parameter**	goes to	**Database Field**
txtSurname.Text	→	@Surname	→	Surname

Using Parameters

Adding a Parameter between the Textbox and the Database gives an extra layer of protection for your database.

We could just add Data from the Textbox and pass it directly to the SQL, but that's nasty and leaves you open to SQL attacks.

With parameters you don't need to know the name of the textbox in the SQL command, it's much cleaner although can be more tedious to write. .

Placeholders, as mentioned, always start with an @ symbol @Title. They do not need to be named after the database column that they represent, but it is often easier if they are, and it helps to self-document your code.

The term *parameters* here refers to the parameters required to **provide data to your SQL statement** or stored procedure.

Using Parameters to hold the data we can create text boxes that will allow us to input our data to our database. In this example data from the txtTitle textbox is passed to the @Title parameter, then the @Title passes to the SQL Title command.

```
newdata.Parameters.AddWithValue("@Title", txttitle.Text)

string NewEntry = "INSERT INTO Golf (Title, Firstname, Surname, Gender, DOB,
Street, Suburb, City, [Available week days], Handicap) VALUES ( @Title,
@Firstname, @Surname, @Gender,  @DOB, @Street,  @Suburb, @City, @Available,
@Handicap)";
```

However using Parameters **cmd.Parameters.AddWithValue("@Parameter", txtTextBox1.Text);** does not solve everything because the value inserted isn't restricted to a type, it goes in as an **object**.

If you want a Date make sure that ONLY a Date can be added in.

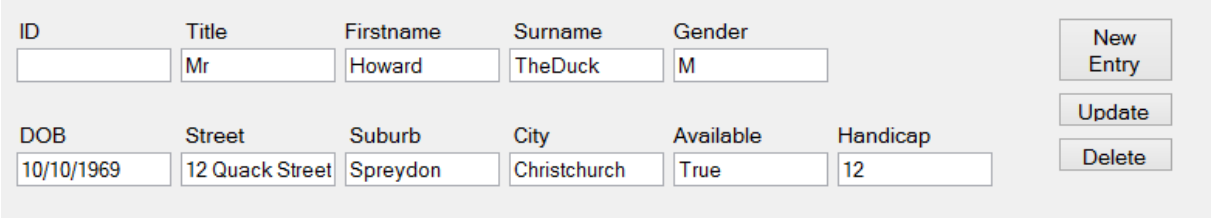
```
cmd.Parameters.Add("@Parameter", SqlDbType.DateTime).Value = MyDateTimeVariable;
```

Read this article here, and how to specify the Type so that errors, and attacks are prevented. [Can we stop using AddWithValue() already?]

The **AddWithValue** method here accepts the name of the parameter and the object that you want to add.

Create the extra textboxes and their label names below. Get the names from the next section of code.

The insert command inserts the data into the table at the last row.



No Id field is needed as it is added automatically.

Note in the code that [ ] go around names that have spaces in them [Available week days].

Here is the main piece of code. Adding = "James"; to the end gives you some default data so you don't have to type into the boxes every time.

```
newdata.Parameters.AddWithValue("@Firstname", txtFirstname.Text).Value = "James";
```

Delete out the .Value = "James"; and the other values when you get your code working.

```
private void btnNew_Click(object sender, EventArgs e) {
    // this puts the parameters into the code so that the data in the text boxes is added to the
database
     string NewEntry = "INSERT INTO Golf (Title, Firstname, Surname, Gender, DOB, Street, Suburb, City, [Available
week days], Handicap) VALUES ( @Title, @Firstname, @Surname, @Gender,  @DOB, @Street,  @Suburb, @City, @Available,
@Handicap)";
     SqlConnection Con = new SqlConnection();
     string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial Catalog=golf;Integrated
Security=True";
     Con.ConnectionString = connectionString;
     using (SqlCommand newdata = new SqlCommand(NewEntry, Con)) {

newdata.Parameters.AddWithValue("@Title", txttitle.Text).Value = "Mr";
newdata.Parameters.AddWithValue("@Firstname", txtFirstname.Text).Value = "James";
newdata.Parameters.AddWithValue("@Surname", txtSurname.Text).Value = "Bond";
newdata.Parameters.AddWithValue("@Street", txtStreet.Text).Value = "123 Bond Street";
newdata.Parameters.AddWithValue("@Suburb", txtSuburb.Text).Value = "Merivale";
newdata.Parameters.AddWithValue("@City", txtCity.Text).Value = "Christchurch";
newdata.Parameters.AddWithValue("@Gender", txtGender.Text).Value = "M";
newdata.Parameters.AddWithValue("@DOB", txtDOB.Text).Value = "1/2/1935";
newdata.Parameters.AddWithValue("@Handicap", txtHandicap.Text).Value = "2";
newdata.Parameters.AddWithValue("@Available", txtAvailable.Text).Value = "";

        Con.Open(); //open a connection to the database
        //its a NONQuery as it doesn't return any data its only going up to the server
```

```
        newdata.ExecuteNonQuery();  //Run the Query
       //a happy message box
MessageBox.Show("Data has been Inserted  !! ");
        }
    //Run the LoadDatabase method we made earler to see the new data.
    loadDatabase();
    }
```

There is so much that can be optimised, but to date I have left it untouched to keep it easier to follow.

We are repeating heaps of code under each button that can be globalised, such as

```
SqlConnection Con = new SqlConnection();
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
Con.ConnectionString = connectionString;
```

So this should be looking familiar for each Method we make. We should only have one connection string for the entire program, to make maintenance easier.

Update Command

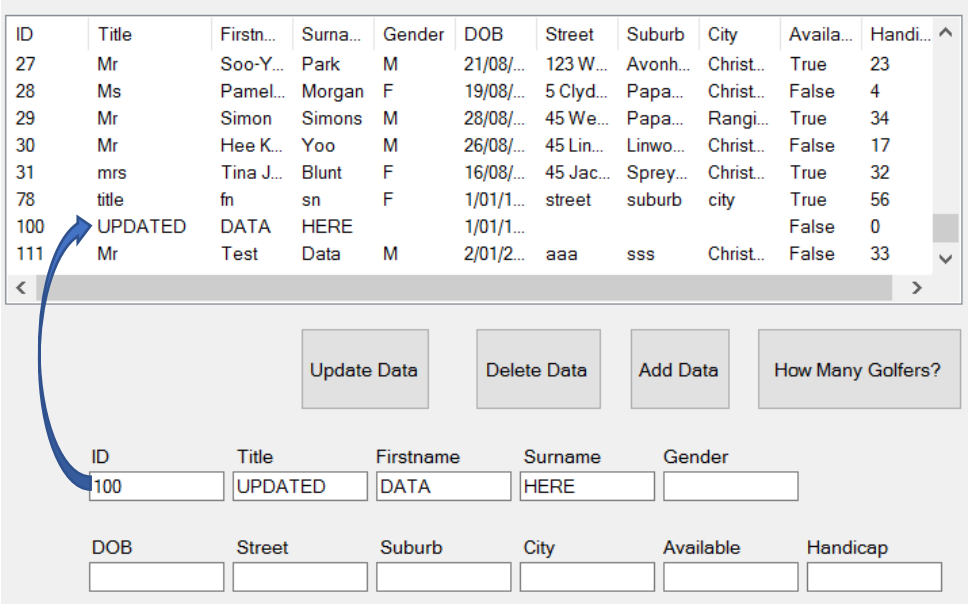
Update is similar to Insert. It changes the data and uploads it over the existing data.

To modify data within a database

- Create a **SqlConnection** object and give it a **connection string**. SqlConnection Con = new SqlConnection();
- Create a **SqlCommand** object and assign to it the connection object you opened. Place an SQL statement into the Command object SqlCommand update = new SqlCommand(updatestatement, Con)
- Open the connection to the database. Con.Open();
- Execute the SQL statement by invoking the **ExecuteNonQuery** method on the Command object. update.ExecuteNonQuery();
- Close the Connection. Con.Close();

```
private void btnUpdate_Click(object sender, EventArgs e) {
         //this updates existing data in the database where the ID of the data equals the ID in
the text box

string updatestatement = "UPDATE Golf set  Title=@Title, Firstname=@Firstname,
Surname=@Surname, Gender=@Gender, DOB=@DOB, Street=@Street, Suburb=@Suburb, City=@City,
[Available week days]=@Available, Handicap=@Handicap  where ID = @ID";

          SqlConnection Con = new SqlConnection();
          string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
          Con.ConnectionString = connectionString;

SqlCommand update = new SqlCommand(updatestatement, Con)
//create the parameters and pass the data from the textboxes

             update.Parameters.AddWithValue("@ID", txtID.Text);
             update.Parameters.AddWithValue("@Title", txtTitle.Text);
             update.Parameters.AddWithValue("@Firstname", txtFirstname.Text);
             update.Parameters.AddWithValue("@Surname", txtSurname.Text);
             update.Parameters.AddWithValue("@Street", txtStreet.Text);
             update.Parameters.AddWithValue("@Suburb", txtSuburb.Text);
             update.Parameters.AddWithValue("@City", txtCity.Text);
             update.Parameters.AddWithValue("@Gender", txtGender.Text);
             update.Parameters.AddWithValue("@DOB", txtDOB.Text);
             update.Parameters.AddWithValue("@Handicap", txtHandicap.Text);
             update.Parameters.AddWithValue("@Available", txtAvailable.Text);

               Con.Open();
               //its NONQuery as data is only going up
               update.ExecuteNonQuery();
               Con.Close();
               loadDatabase();
          }
```

My example below shows #100 changing with the update command.



Delete Command

Delete is easy to set up, all we have to do is tell the command to delete the entry where the ID equals one you choose. string DeleteCommand = "Delete Golf where ID = @ID";

```
private void btnDelete_Click(object sender, EventArgs e) {
            SqlConnection Con = new SqlConnection();
            string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
            Con.ConnectionString = connectionString;

            string DeleteCommand = "Delete Golf where ID = @ID";

SqlCommand DeleteData = new SqlCommand(DeleteCommand, Con);
DeleteData.Parameters.AddWithValue("@ID", txtID.Text);

                Con.Open();
                DeleteData.ExecuteNonQuery();
                Con.Close();
            loadDatabase();
            }
```

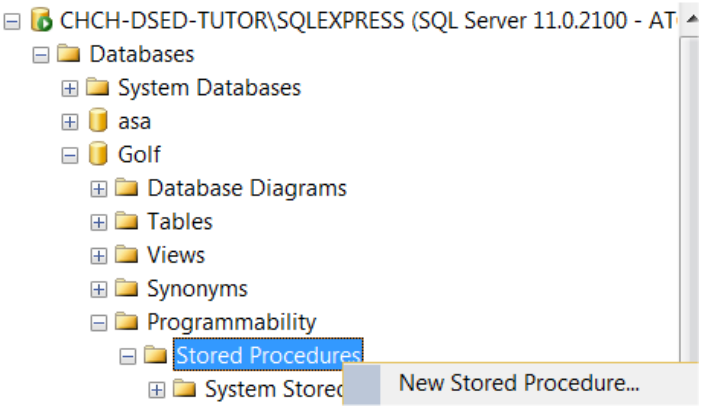
Stored Procedure - The SelectCommand Property

https://msdn.microsoft.com/en-nz/library/ms345415.aspx

Instead of writing SQL commands (Select, Insert, Update, Delete) you could instead execute a stored procedure - a group of SQL statements stored in the database under a unique name and executed as a unit.

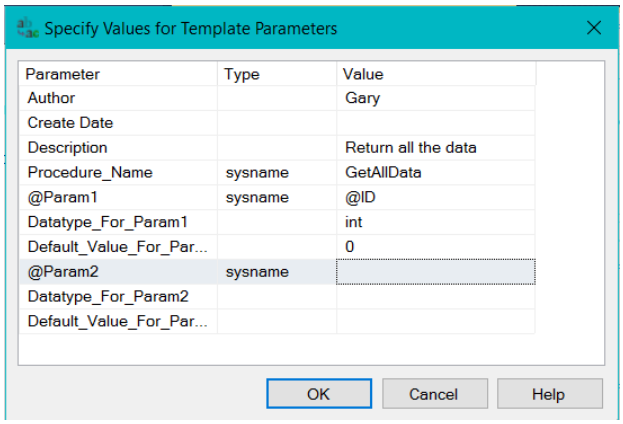
> The benefits of Stored Procedures are reduced server/client network traffic, stronger security, reusable code, easier maintenance and improved performance.

Go to your SQL Management program and follow the path below. Right click on **Stored Procedures** and Choose **Stored Procedure**



On the Query menu, click Specify Values for Template Parameters.

In the Specify Values for Template Parameters dialog box, enter the following values for the parameters shown, with your name.




```
-- ==================================
-- Author:      Gary
-- Create date:
-- Description: Return all the data
-- ==================================
CREATE PROCEDURE GetAllData
    -- Add the parameters for the sto
    @ID int = 0,
AS
BEGIN
    -- SET NOCOUNT ON added to preven
    -- interfering with SELECT statem
    SET NOCOUNT ON;

    -- Insert statements for procedur
    SELECT @ID,
END
GO
```

Click OK, you will see some errors, it wanted two parameters.

Tidy it up and add in the Query we want.

```
-- ========================================
-- Author:      Gary
-- Create date:
-- Description: Return all the data
-- ========================================
CREATE PROCEDURE GetAllData
    -- Add the parameters for the stored p
    @ID int = 0
AS
BEGIN
    -- SET NOCOUNT ON added to prevent ext
    -- interfering with SELECT statements
    SET NOCOUNT ON;

    -- Insert statements for procedure her
    SELECT *
    FROM golf
    WHERE ID = @ID
END
GO
```

To test the syntax, on the Query menu, click Parse or click the tick 

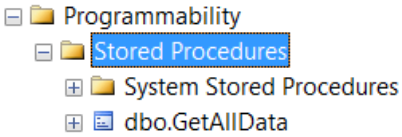
If an error message is returned, compare the statements with the information above and correct as needed.

Results

```
Command(s) completed successfully.
```

To create the procedure, from the Query menu, click Execute or 

The procedure is created as an object in the database.



To see the procedure listed in Object Explorer, right-click Stored Procedures and select Refresh.

To run the procedure, in Object Explorer, right-click the stored procedure name and select Execute Stored Procedure.

In the Execute Procedure window, enter 20 as the value for the parameter @ID.

Parameter	Data Type	Output Para...	Pass Null V...	Value
@ID	int	No	☐	20

Output shows its working below.

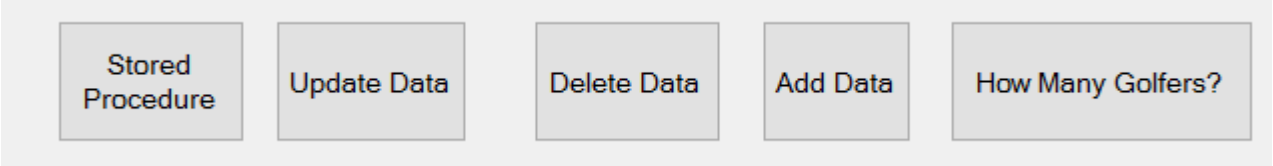
100 %

Results | Messages

	ID	Surna...	Firstname	Ti...	Gen...	DOB	Street	Suburb	City
1	20	Lassiter	Lucy Anne	Ms	F	1982-08-14 00:00:00.000	76 Parkhouse Road	New Brighton	Christchurch

Calling the Stored procedure from your code

https://msdn.microsoft.com/en-us/library/d7125bke.aspx



Accessing a stored procedure is more verbose (but not more difficult) than accessing a normal SQL. The approach is as follows:

1. Create a SqlCommand object.
2. Configure it to access a stored procedure by setting the CommandType property.
3. Add parameters that exactly match those in the stored procedure itself.
4. Execute the stored procedure using one of the SqlCommand object's ExecuteX methods.

The CommandText property now specifies the name of the stored procedure that you want to execute instead of the SQL string that was specified in the previous example.

cmd.CommandText = "GetAllData";

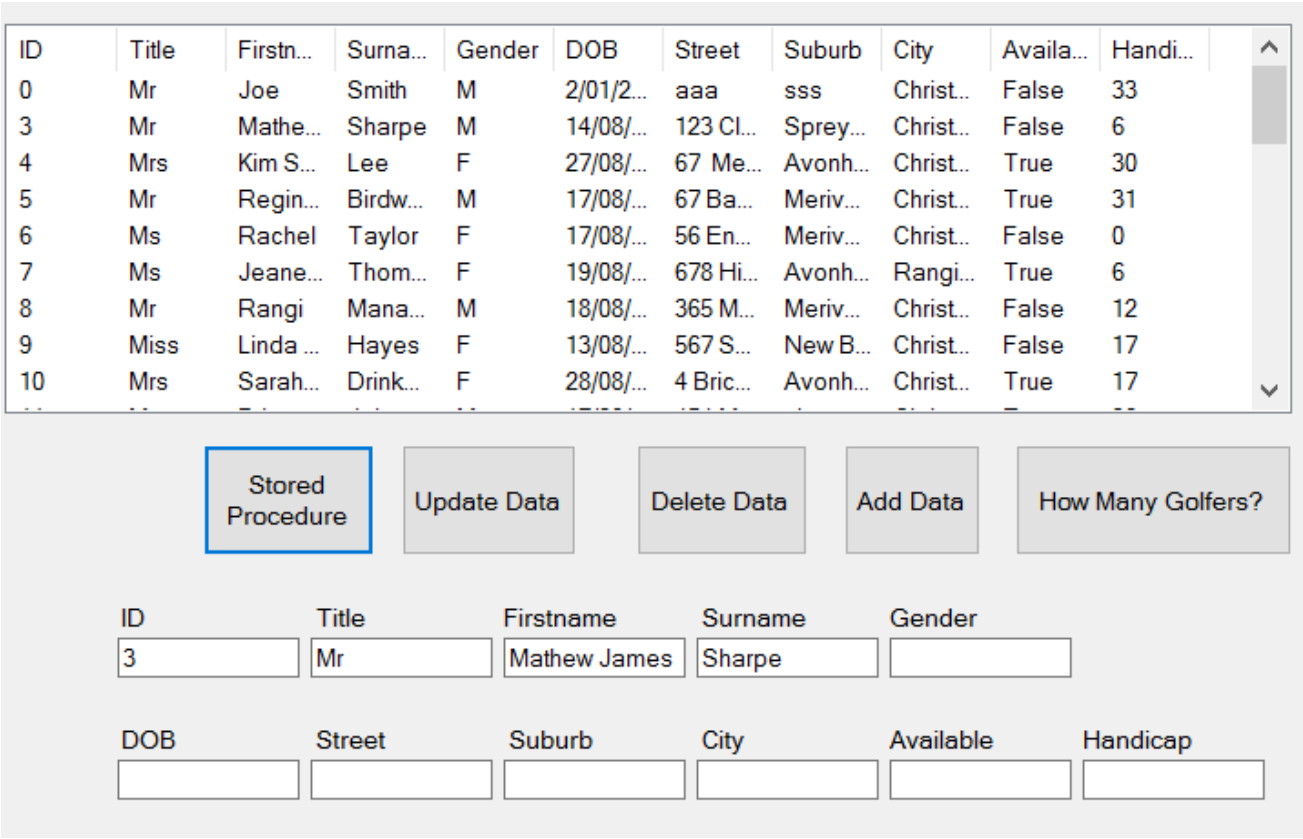
Also notice the CommandType property. It is set to a value of **CommandType.StoredProcedure**, which indicates that the CommandText property contains the name of a stored procedure to be executed.

cmd.CommandType = CommandType.**StoredProcedure**;

```
private void btnStoredP_Click(object sender, EventArgs e) {
string connectionString = @"Data Source=GARY-LAPTOP\SQLEXPRESS;Initial
Catalog=golf;Integrated Security=True";
using (SqlConnection Con = new SqlConnection(connectionString))
          using (SqlCommand cmd = Con.CreateCommand()) {
               cmd.CommandText = "GetAllData";
               cmd.CommandType = CommandType.StoredProcedure;
               cmd.Parameters.AddWithValue("@ID", txtID.Text);

               Con.Open();
               SqlDataReader reader = cmd.ExecuteReader();
               while (reader.Read()) {
                    //add each row to the listbox
                    ListViewItem item = new ListViewItem(new[]
                         {
reader["ID"].ToString(), reader["Title"].ToString(), reader["Firstname"].ToString(),
reader["Surname"].ToString(), reader["Gender"].ToString(), reader["DOB"].ToString(),
reader["Street"].ToString(), reader["Suburb"].ToString(), reader["City"].ToString(),
reader["Available week days"].ToString(), reader["Handicap"].ToString()
                         });
```

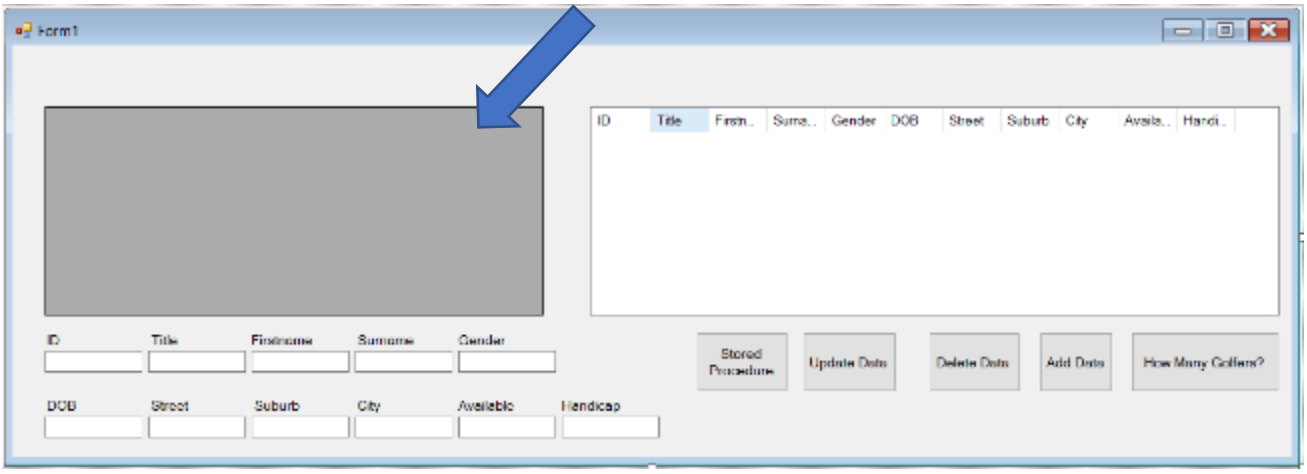
```
                    LVGolf.Items.Add(item);
//add names to text boxes just to show how it can be done
                    txtTitle.Text = reader["Title"].ToString();
                    txtFirstname.Text = reader["Firstname"].ToString();
                    txtSurname.Text = reader["Surname"].ToString();
                    }
                reader.Close();
                Con.Close();
                }
            }
        }
```

Here I have selected person #3. It has added that person to the bottom of the ListView (not shown), and also back into the text boxes.



Adding a DataGridView

Using a ListView might look nice, but really it's not very practical. Here is a DataGridView which we will add in. I also moved the textboxes to the left.



First we need to add the columns and the data to a **DataTable** named GolfTable

Add a DataTable GolfTable = new DataTable(); to your program.

```
Public partial class Form1 : Form {
        //We need a connection to the Database
        SqlConnection Con = new SqlConnection();
        DataTable GolfTable = new DataTable();
//http://www.dotnetperls.com/datatable
```

Create the following sub that generates the table Column Titles

```
public void datatablecolumns() {
           //clear the old data
           GolfTable.Clear();
           //add in the column titles to the datatable
           try {
               GolfTable.Columns.Add("ID");
               GolfTable.Columns.Add("Title");
               GolfTable.Columns.Add("Firstname");
               GolfTable.Columns.Add("Surname");
               GolfTable.Columns.Add("Gender");
               GolfTable.Columns.Add("DOB");
               GolfTable.Columns.Add("Street");
               GolfTable.Columns.Add("Suburb");
               GolfTable.Columns.Add("City");
               GolfTable.Columns.Add("Available week days");
               GolfTable.Columns.Add("Handicap");
               } catch {
               }
           }
```

Add the Method name to the LoadDB

```
private void loaddb(){
        //load datatable columns
        datatablecolumns()}
```

To get data into the DataGridView add the following in your existing LoadDB sub so that the reader reads the data in after the `Do While reader.Read()`

```
while (reader.Read()) {
                    //add in each row to the datatable
GolfTable.Rows.Add(reader["ID"], reader["Title"], reader["Firstname"],
reader["Surname"], reader["Gender"], reader["DOB"], reader["Street"],
reader["Suburb"], reader["City"], reader["Available week days"],
reader["Handicap"]);
                    //add the datatable to the Data Grid View
```

At the end pass the data in GolfTable to the DataGridView

```
reader.Close();
               // connection.Close();
               DGVgolf.DataSource = GolfTable;
```

Click on a DGV row and fill the text boxes

If you double click on the DataGridView it creates a method

```
private void DGVgolf_CellContentClick(object sender,
DataGridViewCellEventArgs e) {
            }
```

This is a pain in the neck.

CellContentClick means it only runs when you click on any **content in the cell**, not when you click on the cell itself. So empty cells, or cells with only a little bit of data sometimes don't trigger the event. Then you spend hours wondering why the click event runs sometimes and doesn't run other times.

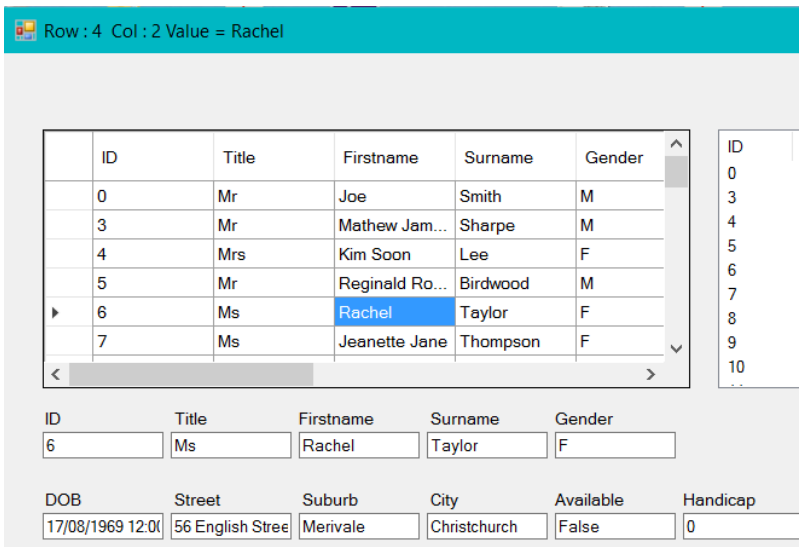
So under your Events in your Properties, change it to be the CellClick event.

CellBorderStyleChanged	
CellClick	**DGVgolf_CellContentClick**
CellContentClick	

Using **CellClick** however does mean that when you click just outside the cell, in the header for example it triggers an error, there is no data it can pull out, so wrap it in a try catch to stop those errors.

When everything works you should be able to click on a row in your DataGridView and see the data in the Text Boxes.

```
private void DGVgolf_CellContentClick(object sender, DataGridViewCellEventArgs e) {
try {
            //show the data in the DGV in the text boxes
string newvalue = DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
            //show the output on the header
this.Text = "Row : " + e.RowIndex.ToString() + "  Col : " +
e.ColumnIndex.ToString() + " Value = " + newvalue;
            //pass data to the text boxes
txtID.Text = DGVgolf.Rows[e.RowIndex].Cells[0].Value.ToString();
txtTitle.Text = DGVgolf.Rows[e.RowIndex].Cells[1].Value.ToString();
txtFirstname.Text = DGVgolf.Rows[e.RowIndex].Cells[2].Value.ToString();
txtSurname.Text = DGVgolf.Rows[e.RowIndex].Cells[3].Value.ToString();
txtGender.Text = DGVgolf.Rows[e.RowIndex].Cells[4].Value.ToString();
txtDOB.Text = DGVgolf.Rows[e.RowIndex].Cells[5].Value.ToString();
txtStreet.Text = DGVgolf.Rows[e.RowIndex].Cells[6].Value.ToString();
txtSuburb.Text = DGVgolf.Rows[e.RowIndex].Cells[7].Value.ToString();
txtCity.Text = DGVgolf.Rows[e.RowIndex].Cells[8].Value.ToString();
txtAvailable.Text = DGVgolf.Rows[e.RowIndex].Cells[9].Value.ToString();
txtHandicap.Text = DGVgolf.Rows[e.RowIndex].Cells[10].Value.ToString();
              } catch {
              }
           }
```



You can even capture the **Row** e.RowIndex and **Column** e.ColumnIndex as well as the contents of the cell you click on. DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value

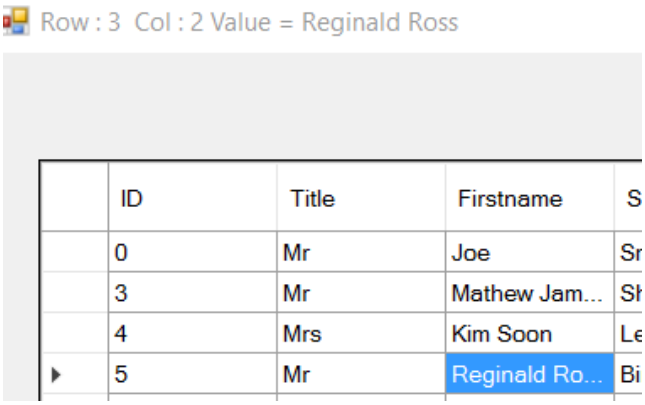
The following code allows you to extract out information such as the ID from your DataGridView and then use it to make another SQL call.

```
//show the output on the header
this.Text = "Row : " + e.RowIndex.ToString() + "  Col : " +
e.ColumnIndex.ToString() + " Value = " + newvalue;
```

e.RowIndex.ToString() Row : 3

e.ColumnIndex.ToString() Col : 2

newvalue Value = Reginald Ross




```
string newvalue =
DGVgolf.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
```

Golf Exercise using SQL

Create Buttons, or any other method to run your queries.

Find the following

1. Which people have a handicap under 11
2. Which Golfers are from Spreydon?
3. Which Golfers are NOT from Christchurch?
4. How many people are male?
5. Order the handicap in ascending order
6. How many people are available to play on the weekday?
7. How many people live in a street?
8. Without using the gender field, how many people are female?
9. How many people have a surname starting with B?
10. How many people are not from Christchurch or Rangiora?

24. Presentation, Business, and Data Layers / Tiers

> "Programs should be written for people to read, and only incidentally for machines to execute." — Structure and Interpretation of Computer Programs by Abelson and Sussman

https://msdn.microsoft.com/en-nz/library/ee658109.aspx?f=255&MSPPError=-2147217396

http://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET

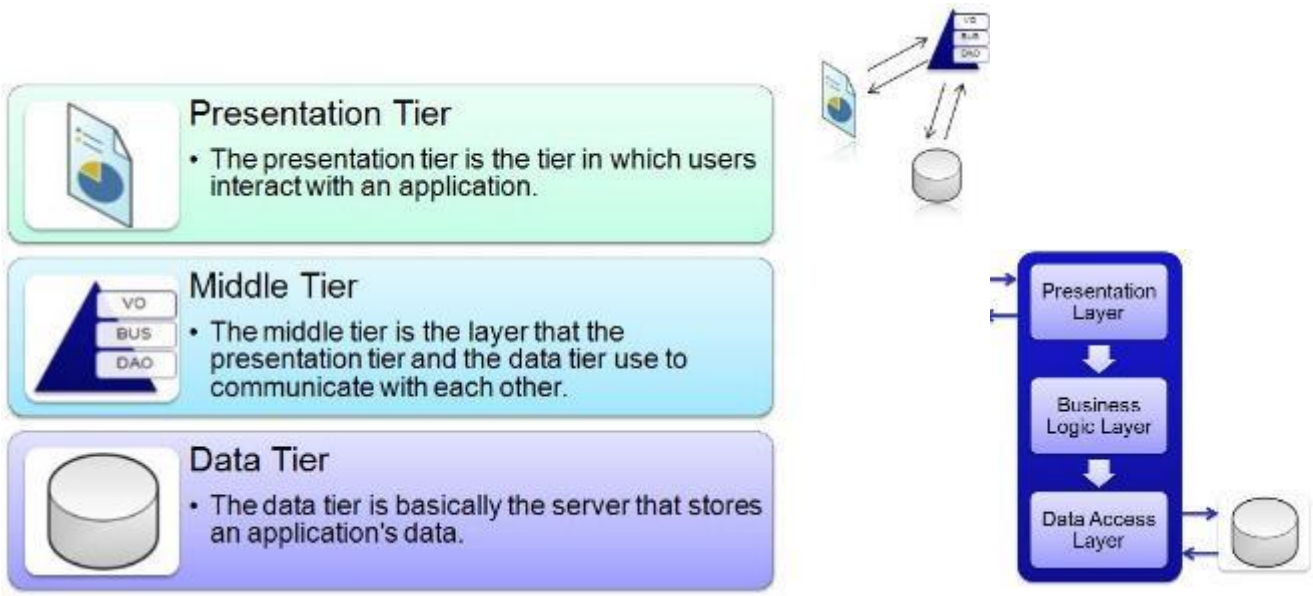
http://www.codeproject.com/Articles/679185/Understanding-Three-Layer-Architecture-and-its

https://medium.com/@msandin/strategies-for-organizing-code-2c9d690b6f33#.3z451rjt4

When a unit of code grows too large and contains too many elements it becomes hard to navigate, hard to get an overview of, and hard to understand: it becomes complex. Our main weapon against this complexity is **divide and conquer**: we split the unit into smaller parts which we can understand in isolation.

At the moment we have all our code stuffed under our Form. But that is not ideal, its hard to find, it gets really messy, and it makes growing the program hard.

We need to use **Layers** or **Tiers**.



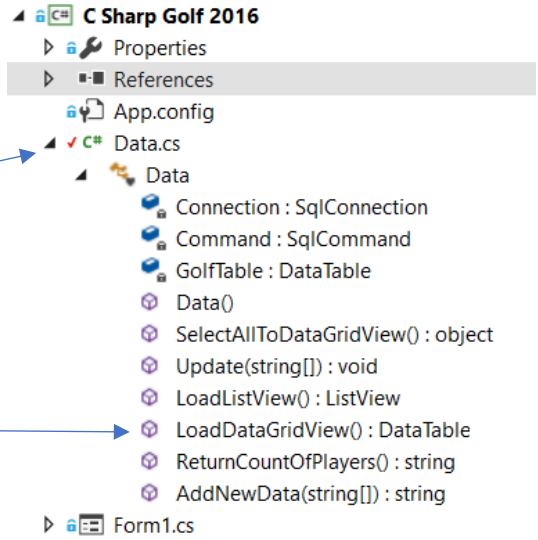
The main benefits of the N-tier/3-tier architectural style are:

- **Maintainability**. Because each tier is independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.
- **Scalability**. Because tiers are based on the deployment of layers, scaling out an application is reasonably straightforward.
- **Flexibility**. Because each tier can be managed or scaled independently, flexibility is increased.
- **Availability**. Applications can exploit the modular architecture of enabling systems using easily scalable components, which increases availability.

Create a Data Layer for your Golf Project

Here is my rough draft at present, with the Database calls now in their own place.

The hardest part is working out how to move the data between the different layers.



On my Form I instantiated the Data class as myData under the globals.

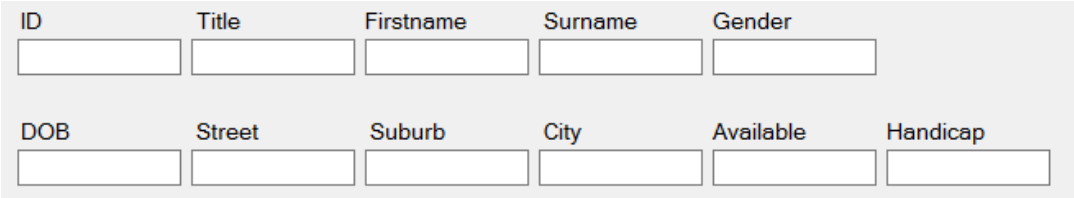
```
Data myData = new Data();
```

The I could use it to call the methods in the class

```
DGVgolf.DataSource =
myData.LoadDataGridView();
```

Three ways of moving Data to the Data layer from the Form

On our Form we have a 11 text boxes holding the data that we need to move to our Data Class.



We can do this in a number of ways and which one you use will depend on what you want to do with that data, and how tidy you are with your coding.

Here is the final code (if you want to skip ahead)

The easiest is to create fields, or properties, in your class and move the textbox data to them, like this **example** of the **Update Query**.

In the **Data Class**

```
public string Firstname, Surname, Title;
```

In the **Form** you just pass the Textboxes to the fields.

```
myData.Firstname = txtFirstname.Text;
myData.Surname = txtSurname.Text;
myData.Title = txtTitle.Text;
```

Then in your **Class** Update **Method** you just add the fields to the parameters for whatever you are doing.

```
update.Parameters.AddWithValue("@Title", Title);
update.Parameters.AddWithValue("@Firstname", Firstname);
```

```
update.Parameters.AddWithValue("@Surname", Surname);
```

This does mean that your code fills up with lots of connections however and can get messy.

Another way is to pass the data to the myData.Update method via the parameters.

```
myData.Update(txtFirstname.Text, txtSurname.Text, txtTitle.Text);
```

Then get them out and use them and use them in your code as above.

```
Update(String Firstname, String Surname, String Title);
```

However this quickly runs into problems, you don't want more than **3** parameters, and 11 is a REALLY BAD look.

So I used an Array instead.

In the Form I made a Global variable …

```
Private String[] AllTextBoxes;
… and under the btnUpdate_Click method I passed in the text boxes and called
the myData.Update(AllTextBoxes) method passing through the array.
private void btnUpdate_Click(object sender, EventArgs e)
        {
AllTextBoxes = new string[]{txtID.Text, txtTitle.Text, txtFirstname.Text,
txtSurname.Text, txtStreet.Text, txtSuburb.Text, txtCity.Text, txtGender.Text,
txtDOB.Text, txtHandicap.Text, txtAvailable.Text };

            myData.Update(AllTextBoxes);
```

Then in the Update method in Data I just called the Array holding the data by its place in the array

```
public void Update(string[] AlltextBoxes) {
string updatestatement ="UPDATE Golf set  Title=@Title, Firstname=@Firstname, Surname=@Surname,
Gender=@Gender, DOB=@DOB, Street=@Street, Suburb=@Suburb, City=@City, [Available week days]=@Available,
Handicap=@Handicap  where ID = @ID";
using (SqlCommand update = new SqlCommand(updatestatement, Connection)) {
//create the parameters and pass the data from the textboxes
update.Parameters.AddWithValue("@ID", AlltextBoxes[0]);
update.Parameters.AddWithValue("@Title", AlltextBoxes[1]);
update.Parameters.AddWithValue("@Firstname", AlltextBoxes[2]);
```

Don't forget this is an example, I had 11 textboxes, here I only show three.

Returning data to the Form

To the DataGridView.

Sending data back to the form in a way that can be consumed can take some thinking. Some are easy.

For example the LoadDataGridView ( ) Class has a DataTable GolfTable = new DataTable(), which you have to fill with data and send to the DataGridView.

Here is the code for the first part of the Data Class.

```
class Data {
private SqlConnection Connection = new SqlConnection();
private SqlCommand Command = new SqlCommand();
DataTable GolfTable = new DataTable();
       public Data() {
Connection.ConnectionString = @"Data Source = GARY-LAPTOP\SQLEXPRESS; Initial Catalog = Golf;
Integrated Security = True";
Command.Connection = Connection;    }

public DataTable LoadDataGridView() {
            string queryString = "SELECT * FROM Golf ORDER by ID";
            GolfTable.Clear();
            //add in the column titles to the datatable
          GolfTable.Columns.Add("ID");
          GolfTable.Columns.Add("Title");
          GolfTable.Columns.Add("Firstname");
          GolfTable.Columns.Add("Surname");
          GolfTable.Columns.Add("Gender");
          GolfTable.Columns.Add("DOB");
          GolfTable.Columns.Add("Street");
          GolfTable.Columns.Add("Suburb");
          GolfTable.Columns.Add("City");
          GolfTable.Columns.Add("Available week days");
          GolfTable.Columns.Add("Handicap");
using (SqlCommand Command = new SqlCommand(queryString, Connection)) {
                Connection.Open();
                SqlDataReader reader = Command.ExecuteReader();
                while (reader.Read()) {
                    //add in each row to the datatable
GolfTable.Rows.Add(reader["ID"], reader["Title"], reader["Firstname"], reader["Surname"],
reader["Gender"], reader["DOB"], reader["Street"], reader["Suburb"], reader["City"],
reader["Available week days"], reader["Handicap"]);
                    }
                reader.Close();
                Connection.Close();
                return GolfTable;
                }
            }
```

Meanwhile back on the form I made a method called LoadFormFromClass() that will hold the loading of the DataGrid and the ListView. The DGV gets the data from the class and passed to its Data Source.

```
public Form1() {
            InitializeComponent();
            LoadFormFromClass();
            }
        public void LoadFormFromClass() {
            DGVgolf.DataSource = myData.LoadDataGridView();
            }
```

Add new Data

Here is the AddNewData Method with a Happy message returning when its successful. You can expand this to return error messages as well. See how the Array string[] AlltextBoxes takes in the data from the Text boxes. Note that when we add new data we don't add an **ID field.**

```
public string AddNewData(string[] AlltextBoxes) {
            // this puts the parameters into the code so that the data in the
text boxes is added to the database
            string QueryString ="INSERT INTO Golf (Title, Firstname, Surname, Gender, DOB, Street,
Suburb, City, [Available week days], Handicap) VALUES ( @Title, @Firstname, @Surname, @Gender,  @DOB,
@Street,  @Suburb, @City, @Available, @Handicap)";
            SqlConnection Con = new SqlConnection();
            try {
using (SqlCommand Com = new SqlCommand(QueryString, Con)) {
Com.Parameters.AddWithValue("@Title", AlltextBoxes[1]);
Com.Parameters.AddWithValue("@Firstname", AlltextBoxes[2]);
Com.Parameters.AddWithValue("@Surname", AlltextBoxes[3]);
Com.Parameters.AddWithValue("@Street", AlltextBoxes[4]);
Com.Parameters.AddWithValue("@Suburb", AlltextBoxes[5]);
Com.Parameters.AddWithValue("@City", AlltextBoxes[6]);
Com.Parameters.AddWithValue("@Gender", AlltextBoxes[7]);
Com.Parameters.AddWithValue("@DOB", AlltextBoxes[8]);
Com.Parameters.AddWithValue("@Handicap", AlltextBoxes[9]);
Com.Parameters.AddWithValue("@Available", AlltextBoxes[10]);

                    Con.Open();
//its a NONQuery as it doesn't return any data its only going up to the server
                    Com.ExecuteNonQuery();
                    Con.Close();
                    //a happy message box
return AlltextBoxes[2] + " " + AlltextBoxes[3] + " has been Inserted  !! ";
                    }
                } catch (Exception ex) {
```

```
                return ex.ToString();
                }
            }
        }
```

On the Form. The MessageBox shows the happy message (This is debatable as to its code smell. The question I would ask is “Why does making a post to the DB return back a string?”) It just feels wrong.

```
private void btnNew_Click(object sender, EventArgs e) {
            AllTextBoxes = new string[] {txtID.Text, txtTitle.Text,
txtFirstname.Text, txtSurname.Text, txtStreet.Text, txtSuburb.Text,
txtCity.Text, txtGender.Text, txtDOB.Text, txtHandicap.Text,
txtAvailable.Text
            };
            MessageBox.Show(myData.AddNewData(AllTextBoxes));
```

To the ListView

This was harder than I thought owing to Obtuse error messages. But we return a ListView that is passed to the ListView on the Form.

This side, in the Data class is straightforward, each time the data loops in pass it to a ListViewItem, ListViewItem item = new ListViewItem then pass that row of data to a fake ListView. fakeLVGolf.Items.Add(item)

```
public ListView LoadListView() {
            string queryString = "SELECT * FROM Golf ORDER by ID";
            ListView fakeLVGolf = new ListView();
using (SqlCommand Command = new SqlCommand(queryString, Connection)) {
                Connection.Open();
                SqlDataReader reader = Command.ExecuteReader();

                while (reader.Read()) {
//add each row to the listbox
ListViewItem item = new ListViewItem(new[]{reader["ID"].ToString(),
reader["Title"].ToString(), reader["Firstname"].ToString(), reader["Surname"].ToString(),
reader["Gender"].ToString(), reader["DOB"].ToString(),reader["Street"].ToString(),
reader["Suburb"].ToString(), reader["City"].ToString(),reader["Available week days"].ToString(),
reader["Handicap"].ToString() });
                    fakeLVGolf.Items.Add(item);
                    }
                reader.Close();
                Connection.Close();
                }
            return fakeLVGolf;
            }
```

The problem happens at the other end, getting the ListView data to pass to another ListView, the one on the form. This Answer solved it for me, using items.Clone() on the end.

I have added this to the existing LoadFormFromClass() class that is called on load from the form.

```
public void LoadFormFromClass() {
//Clear the old items in the Listview
LVGolf.Items.Clear();
foreach (ListViewItem items in myData.LoadListView().Items) {
LVGolf.Items.Add((ListViewItem)items.Clone());
                }
DGVgolf.DataSource = myData.LoadDataGridView();
            }
```

Returning the Count of Players.

This method is easy, we are just returning a string.

```
public string ReturnCountOfPlayers() {
          // a simple Scalar query just returning one value.
          string queryString = "SELECT COUNT(ID) FROM Golf";
using (SqlConnection connection = new
SqlConnection(Connection.ConnectionString)) {
SqlCommand Command = new SqlCommand(queryString, connection);
              connection.Open();
              return Command.ExecuteScalar().ToString();
              }
          }
```

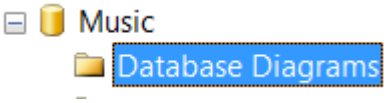
On the form its just ….

```
private void btnCount_Click(object sender, EventArgs e) {
          btnCount.Text = myData.ReturnCountOfPlayers();
```

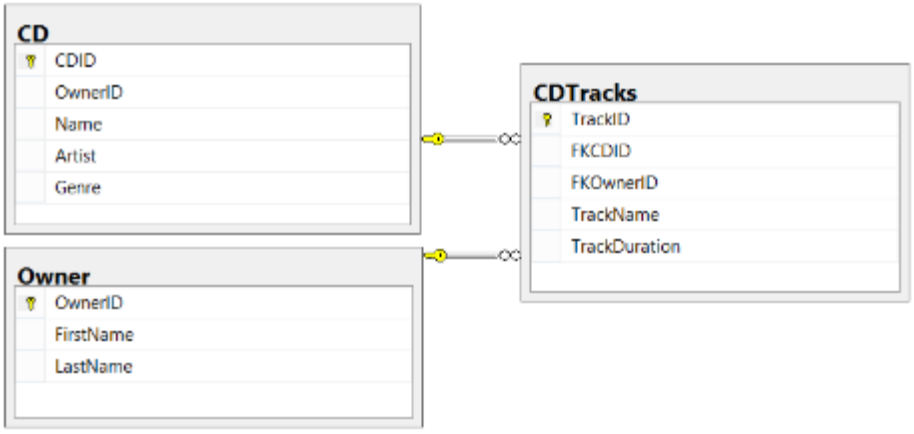
25. Music Database - Relational Database

Primary Key / Foreign Key relationships

The Music database is a RELATIONAL database. A database with more than one table and each table is connected to each other. This relationship scheme from Database Diagrams shows how the three tables are related.



Each table has a Primary Key, which indicates that the field is one that will never repeat, it will always increment automatically 1, 2, 3, 4 etc. If you delete 3 then it will never replace it, the next record will be 5, then 6 then 7.



The tables are joined together with A **One to Many** Relationship.

The **Primary Key** side (One) shows the table that has the Primary Key in it, and the Infinite sign (Many) shows the table that can have many entries for that single key (**Foreign Key**).

Foreign keys are the alien invaders from the other primary key tables.

The Primary Key – Foreign Key connection establishes the relationship between the tables in a One to Many relationship.

For example, ONE Owner (OwnerID in the Owner Table) can have MANY CD's. (Owner ID in the CD Table).

ONE CD (CDID in the CD Table) has MANY tracks (CDID in the CDTracks Table).

Thus, by following the relationships you can easily see how many music tracks each Owner has, and their name and duration.

To make the next section easier (it will hopefully avoid the problems you will read later) go **Tools / Options / Designers** and **UNTICK** the **Prevent saving changes** that require table re-creation. This is not recommended for a 'real' database as it could mess up your data but for this practice it's an easy workaround.

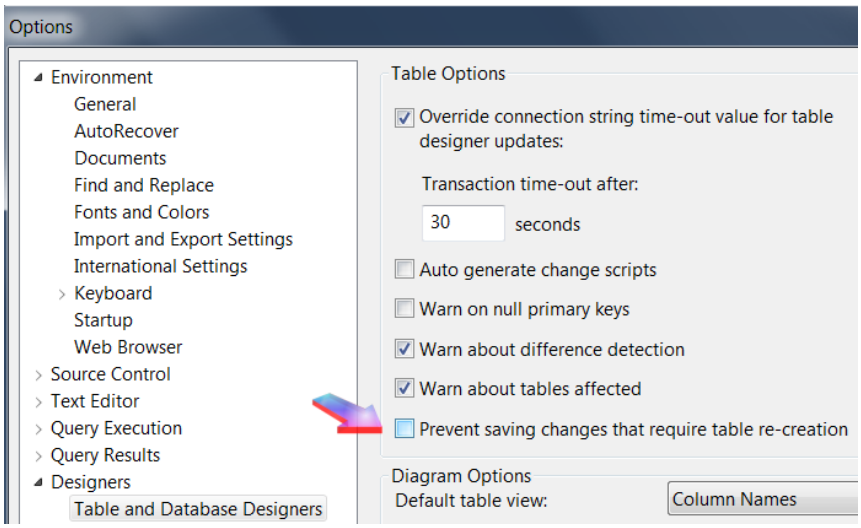


Table structure to date.

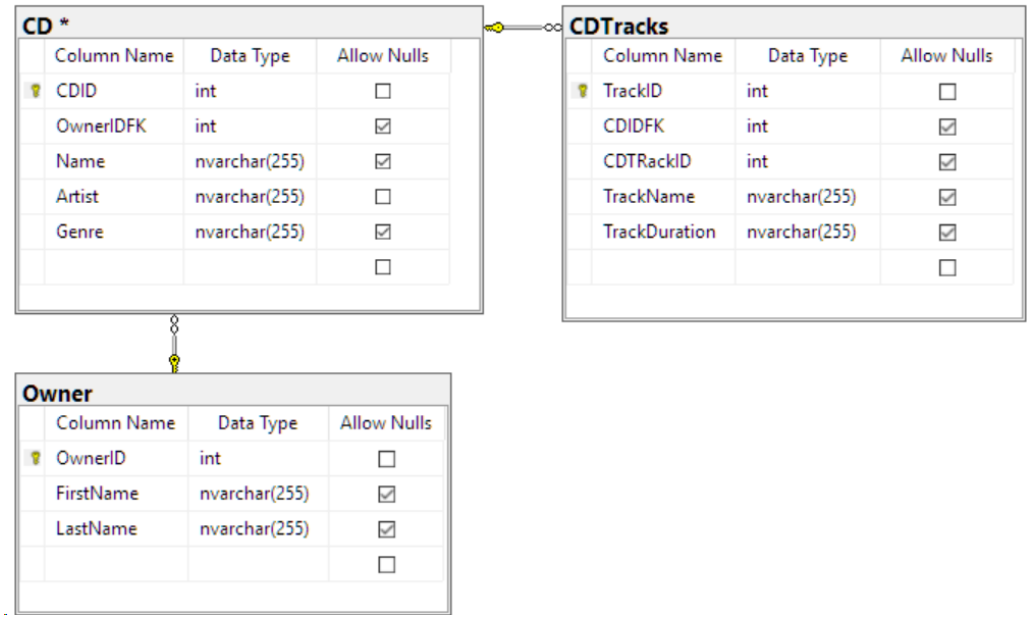


Table Data

Owner Table

	OwnerID	FirstName	LastName
1	1	John	Smith
2	2	Arnold	Swartznager
3	3	Harry	Houdini
4	4	Barry	Bartholomew
5	5	Craig	Carick
6	6	Pablo	Rod

CD Table

	CDID	OwnerIDFK	Name	Artist	Genre
1	1	1	ABBA Gold: Greatest Hits	ABBA	Pop
2	2	1	Led Zeppelin IV	Led Zeppelin	Pop
3	3	2	Their Greatest Hits (1971–1975)	Eagles	Hard rock
4	4	2	Saturday Night Fever	Bee Gees	Soundtrack
5	5	3	The Dark Side of the Moon	Pink Floyd	Progressive rock
6	6	3	Bat Out of Hell	Meat Loaf	Rock
7	7	4	Rumours	Fleetwood Mac	Rock
8	8	4	Sgt. Pepper's Lonely Hearts Cl...	The Beatles	Rock
9	9	5	Goodbye Yellow Brick Road	Elton John	Rock
10	10	5	Born in the U.S.A.	Bruce Springs...	Rock
11	11	6	1984	David Bowie	Rock
12	16	6	Rock and Roll Forever	Bavid Dowie	Soundtrack

Tracks Table

TrackID	CDIDFK	CDTRackID	TrackName	TrackDurati...
1	1	1	Dancing Queen	3:45
2	1	2	Knowing Me, Knowing You	4:00
3	1	3	Take a Chance on Me	4:01
4	1	4	Mamma Mia	3:42
5	1	5	Lay All Your Love on Me	3:32
6	1	6	Ring Ring	3:02
7	1	7	I Do, I Do, I Do, I Do, I Do	3:15
8	1	8	The Winner Takes It All	4:54
9	1	9	Money, Money, Money	3:05
10	1	10	S.O.S	3:19
11	2	1	Black Dog	4:54
12	2	2	Rock and Roll	3:40
13	2	3	The Battle of Evermore	5:51
14	2	4	Stairway to Heaven	8:02
15	3	1	Take It Easy	3:29
16	3	2	Witchy Woman	4:10
17	3	3	Lyin' Eyes	6:21
18	3	4	Already Gone	4:13
19	4	1	Stayin' Alive	4:45
20	4	2	How Deep Is Your Love	4:05
21	4	3	Night Fever	3:33
22	4	4	More Than a Woman	3:18
23	5	1	Speak to Me	1:30
24	5	2	Breathe	2:43
25	5	3	On the run	3:30
26	5	4	Time	6:53
27	5	5	The great gig int eh sky	4:15
28	6	1	Bat Out of Hell	9:45
29	6	2	You Took the Words Right ou...	5:04
30	6	3	Heaven Can Wait	4:38
31	6	4	All Revved Up with No Place ...	4:19
32	7	1	Second Hand News	2:43
33	7	2	Dreams	4:14
34	7	3	never going back again	2:02
35	7	4	Don't Stop	3:11
36	7	5	Go your own way	3:38

37	8	1	With a Little Help from My Fr...	2:02
38	8	2	Lucy in the Sky with Diamonds	2:44
39	8	3	Getting better	3:28
40	8	4	Fixing a Hole	2:48
41	9	1	Funeral for a Friend/Love Lies...	11:09
42	9	2	Candle in the Wind	3:50
43	9	3	Bennie and the Jets	5:23
44	9	4	Goodbye Yellow Brick Road	3:13
45	10	1	Born in the U.S.A.	4:39
46	10	2	Cover Me	3:27
47	10	3	Darlington County	4:49
48	10	4	Working on the Highway	3:11
49	10	5	Downbound Train	3:35
50	10	6	I'm on fire	2:37
51	11	1	No Idea	3:00
52	11	2	Still No Idea	3:00
53	11	3	Oh Dear	3:00

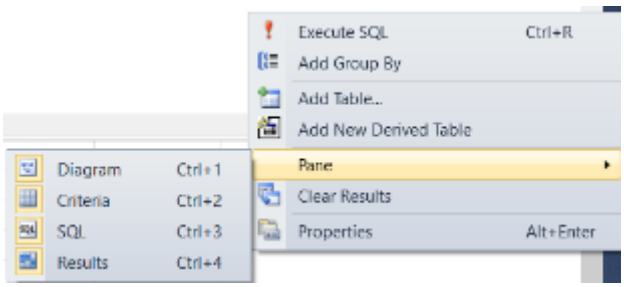
Creating Database Views – Queries

CDs and Owners

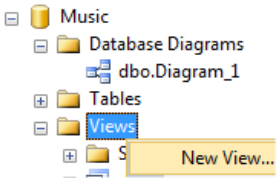
Views are Queries, or questions that you make from your tables using SQL. Luckily Views generate the SQL for you.

The *Views* window is divided in four parts: **Diagram Panel, Criteria Panel, SQL Panel and Results Panel.**

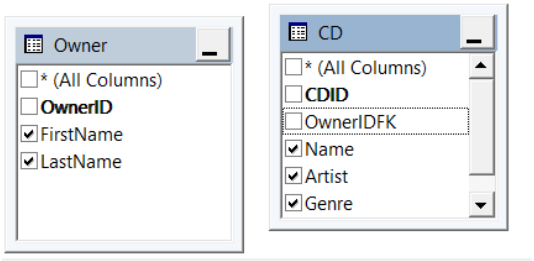
Through these panels you can assemble its *views* through the SQL language or through mouse selection.



Create a new view

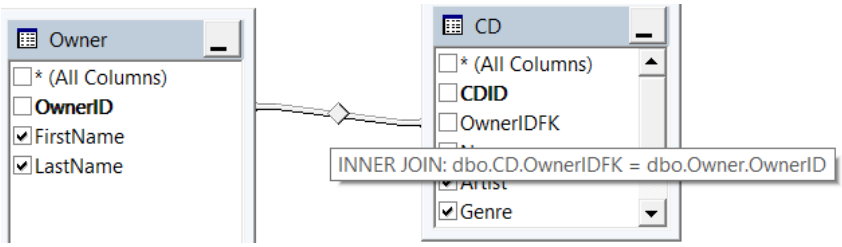


Add the Owner and CD tables by right clicking on the first section and going **Add Table**.



Click on the OwnerIDFK and drag it home to the Owner Table

This builds a connection and chooses the most obvious SQL See how the SQL is written for you.




```
SELECT dbo.CD.Name, dbo.CD.Artist, dbo.CD.Genre, dbo.Owner.FirstName, dbo.Owner.LastName
FROM    dbo.CD INNER JOIN
            dbo.Owner ON dbo.CD.OwnerIDFK = dbo.Owner.OwnerID
```

Hit the Execute/Run button to see the results generate at the bottom of the screen. TaDa!

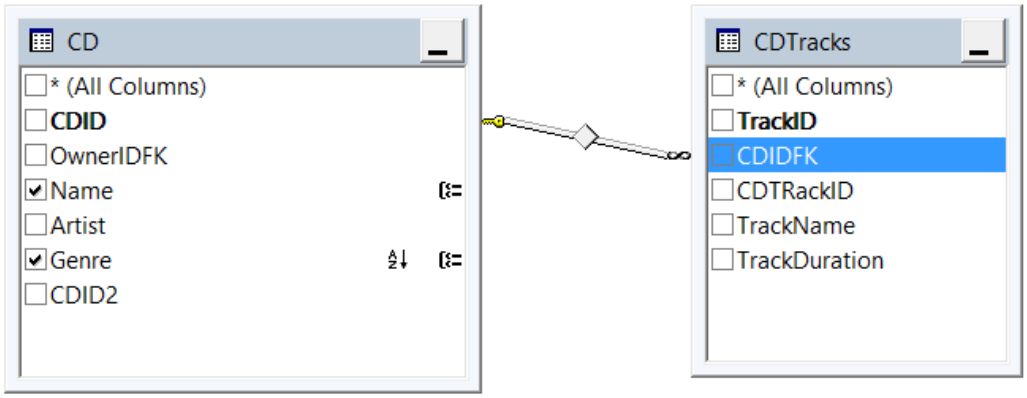
Name	Artist	Genre	FirstName	LastName
ABBA Gold: ...	ABBA	Pop	John	Smith
Led Zeppeli...	Led Zeppelin	Pop	John	Smith
Their Greate...	Eagles	Hard rock	Arnold	Swartznager
Saturday Ni...	Bee Gees	Soundtrack	Arnold	Swartznager
The Dark Si...	Pink Floyd	Progressive rock	Harry	Houdini

Save as **OwnerCD**

Count of Tracks by CD

Lets make a more interesting View.

We want to **count how many Tracks** there are on each CD. Create a new View and add CD and CDTracks Table. Drag CDIDFK home to the CD table.



Click a tick beside Name and TrackID or choose them from the Column and Table field pull downs.

Click the GroupBy button then when the Group By column appears choose Count for TrackID

Column	Alias	Table	Output	Sort Type	Sort Order	Group By	F
Name		CD	☑			Group By	
TrackID	Expr1	CDTracks	☑			Count	

You will now have the view below laid out Click the Run button and see what you get.

Name	Expr1
Sgt. Pepper's Lonely Hearts Club Band	4
Their Greatest Hits (1971–1975)	4
ABBA Gold: Greatest Hits	10
Bat Out of Hell	4

Note that the column that shows how many tracks on each CD is called **Expr1**. Change the name in the Alias column to read **CountofTracks**. Now run again and see the new column name. Save the View as **CountOfTracks.**

Group By Genre

Create a new view grouping all the records by Genre. Save as **GroupByGenre**.

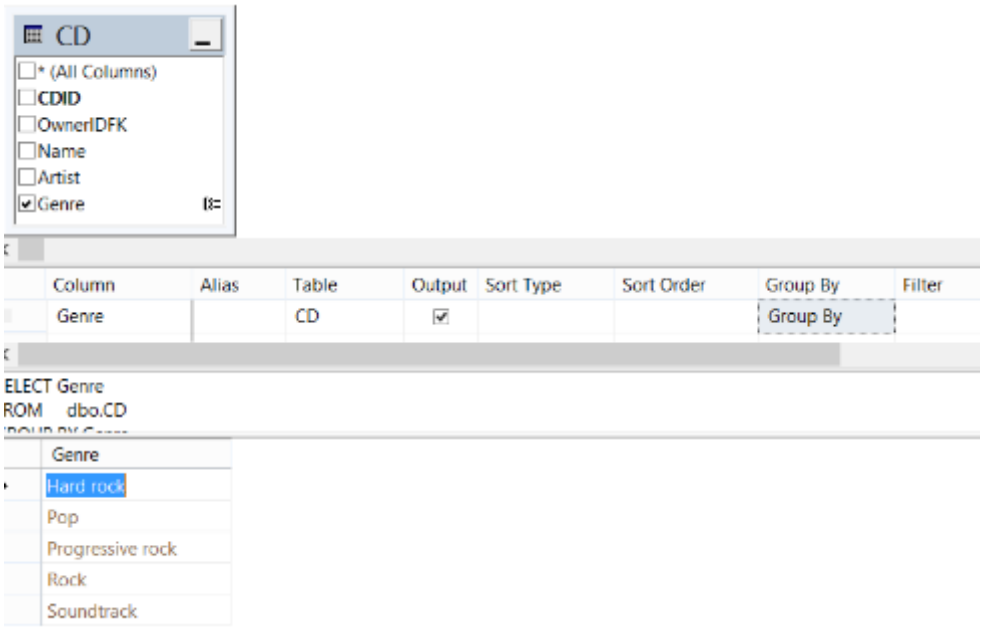
Column	Alias	Table	Output	Sort Type	Sort Order	Group By
Genre		CD	☑			Group By
Genre	GenreCount	CD	☑			Count
			▣			
			▣			
			▣			

```
ECT TOP (100) PERCENT dbo.CD.Genre, COUNT(dbo.CD.Genre) AS GenreCount
OM    dbo.CD INNER JOIN
        dbo.CDTracks ON dbo.CD.CDID = dbo.CDTracks.CDIDFK
OUP BY dbo.CD.Genre
```

Genre	GenreCount
Hard rock	4
Pop	14
Progressive rock	5
Rock	23
Soundtrack	4

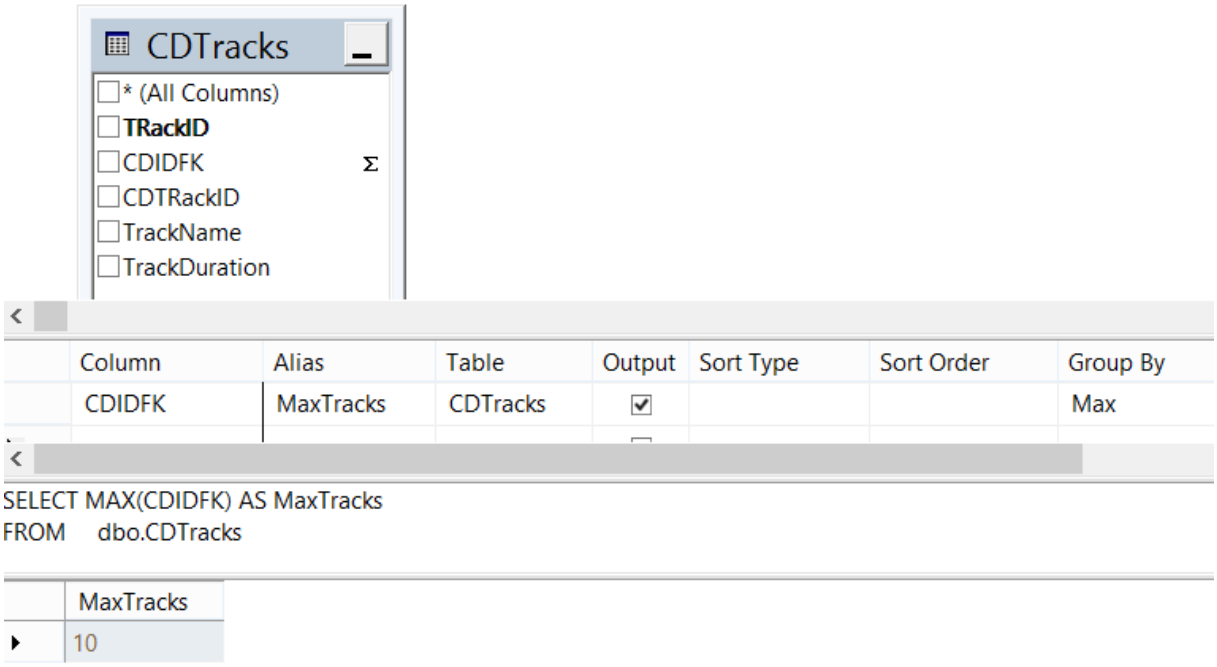
Unique Genre

Make a new View Save it as **UniqueGenre** – to show just unique names from the Genre list instead of each entry



Who has the CD with the most Tracks? Max of Tracks

To answer this question we need to first find the CD with the most Tracks and save it as MaxTracks Save and close it.



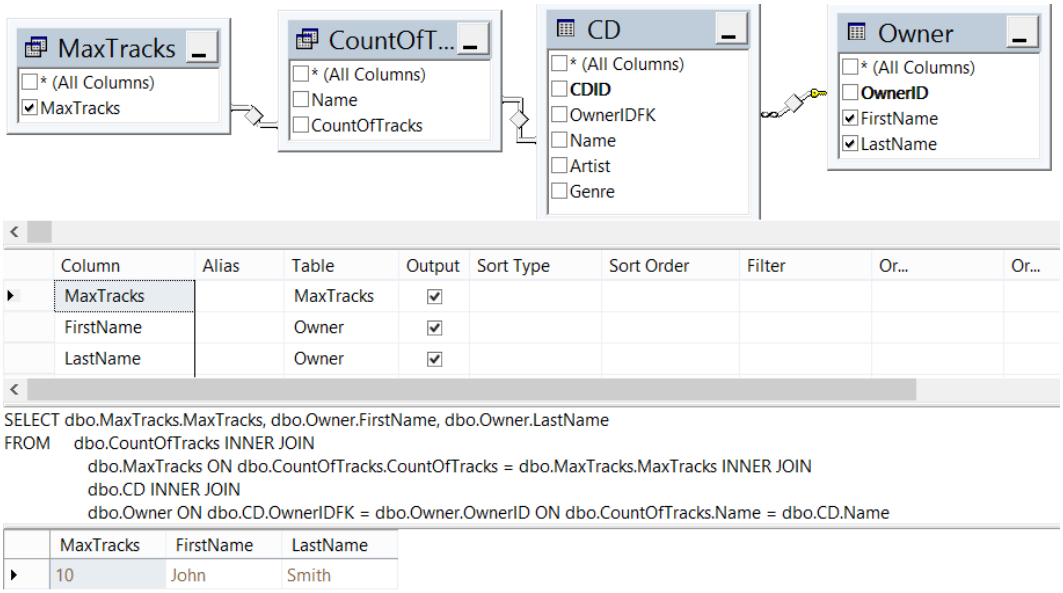
Make a new View and add the following two views and two tables below.

Drag the **MaxTracks** field (the one with the tick) from its Table box over to CountofTracks and drop it over the **CountofTracks**.

This will show the program that MaxTracks is related to CountofTracks and build the little connection.

Do the same for CountOfTracks **Name** and CD **Name**. That will make a path all the way to Owner showing John Smith owns the record Abba with the most tracks.

Save as **OwnerMaxTracks**



26. Music Database in Visual Studio

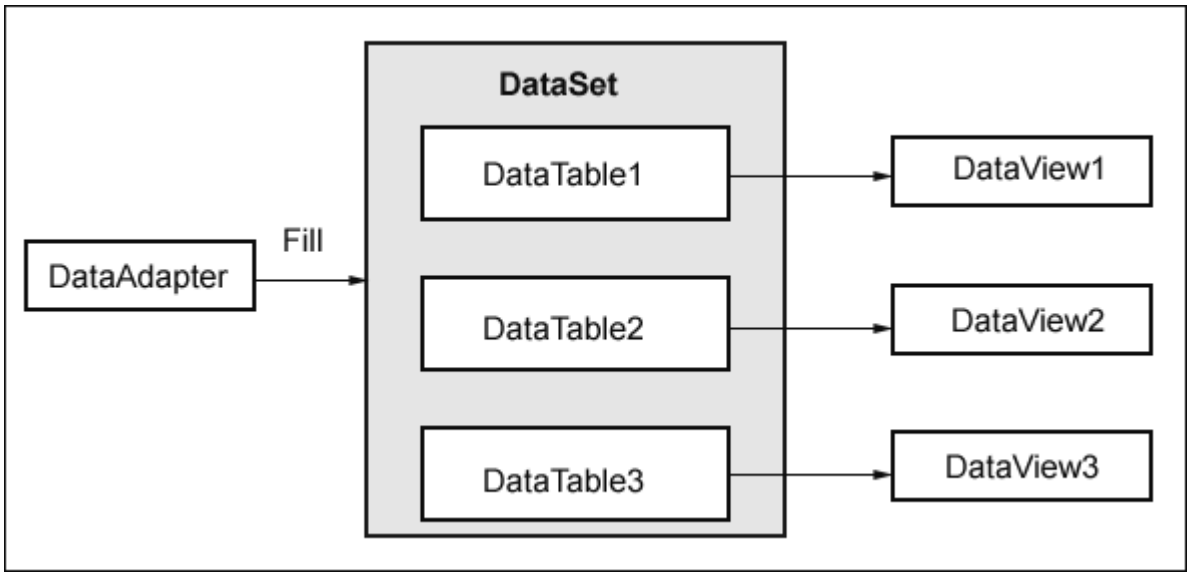
Create a new Project in Visual Studio and add the connections to connect to the Music database.

In the Golf exercise we just used a **DataTable**. This holds only one table's data.

As we are working with **multiple tables** in this exercise we have to use a **DataAdapter**, and sometimes a **DataSet** which hold multiple DataTables and all the schema, headings etc. that go with them.

The DataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data. Populating a DataSet from a DataAdapter - Read. Using the DataAdapter.

The DataAdapter provides this bridge by mapping **Fill**, which changes the data in the DataSet to match the data in the data source,



Using a DataGridView

Add a DataGridView to your form and the following code. The DataGridView control reads the schema information and creates the correct number of columns for your data.

It also uses the column names in the schema as the column names for the grid and each column has the default width.

The **SqlDataAdapter** reads the data from the database and populates the DataTable using the Fill method. Previously we used a **Reader** to read the data out. This does it all at once.

Note that you don't have to actually open and close the connection explicitly as the DataAdapter's Fill() method leaves the connection in the same state as when the method was invoked.

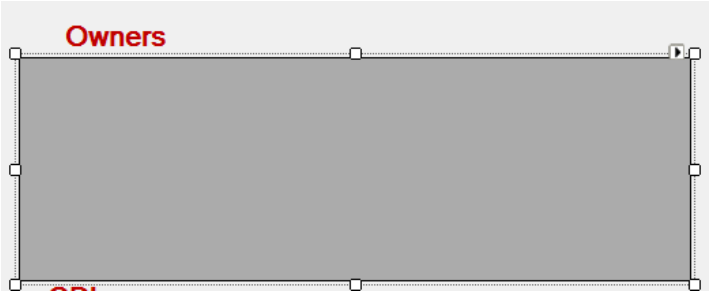
Read this great overview of the DataAdapter Five-different-overloads-of-the-DataAdapter-Fill

The Owners DataGridView and Class

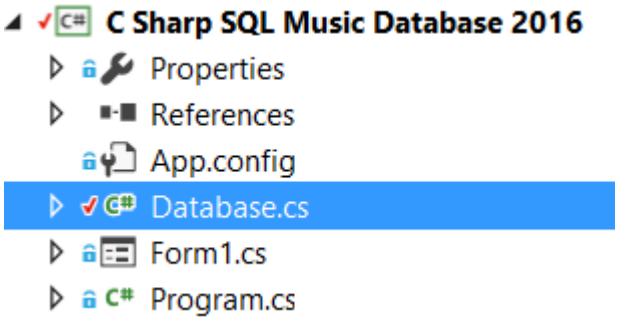
Owing to the complexity of this program we are going to build it piece by piece.

Create a new **DataGridView**, Name it **DGVOwner**

Add a **Label**, have the LblOwner.text = “Owners”.



Create a new Class to hold your Database Calls in.



In your **Database.CS** class set some private fields for the SQL to be used only in the class. Set also defaults such as the Connection string and join it to the Commands

```
class Database
   {
       //Create Connection and Command,and an Adapter.
       private SqlConnection Connection = new SqlConnection();
       private SqlCommand Command = new SqlCommand();
       private SqlDataAdapter da = new SqlDataAdapter();
       //THE CONSTRUCTOR SETS THE DEFAULTS UPON LOADING THE CLASS
       public Database()
       {
 //change the connection string to run from your own music db
string connectionString = @"Data Source=GARY-LAPTOP\sqlexpress;Initial
Catalog=Music;Integrated Security=True";
           Connection.ConnectionString = connectionString;
           Command.Connection = Connection;
        }

        public DataTable FillDGVOwnerWithOwner()
        {
            //create a datatable as we only have one table, the Owner
 DataTable dt = new DataTable();
using (da = new SqlDataAdapter("select * from Owner ", Connection))
           {
               //connect in to the DB and get the SQL
               Connection.Open();
               //open a connection to the DB
               da.Fill(dt);
```

Default settings

```
                //fill the datatable from the SQL
                Connection.Close(); //close the connection
            }
            return dt; //pass the datatable data to the DataGridView
        }
```

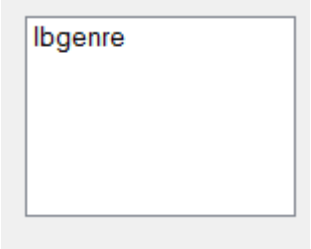
Meanwhile back on the form we need to Instantiate the class Database myDatabase = new Database(); so we can access the methods we are making.

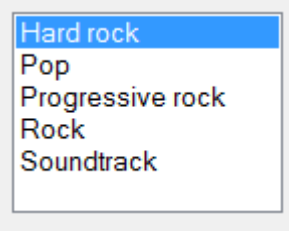
We make a LoadDB method, LoadDB(); to hold all the calls to fill each of the three DataGridView's with table data. Then make a method DisplayDataGridViewOwner() to pass the data to the DataGridView.

```
public partial class Form1 : Form {
       //create an instance of the Database class
       Database myDatabase = new Database();

       public Form1() {
           InitializeComponent();
           loadDB();
        }

public void loadDB() {
           //load the owner dgv
           DisplayDataGridViewOwner();
           }

       //LOAD THE OWNER DATAGRID
private void DisplayDataGridViewOwner() {
           //clear out the old data
           DGVOwner.DataSource = null;
           try {
   DGVOwner.DataSource = myDatabase.FillDGVOwnerWithOwner();
               //pass the datatable data to the DataGridView
DGVOwner.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);
               } catch (Exception ex) {
               MessageBox.Show(ex.Message);
               }
            }
```

Fill the ListBox from the Database

Create a small ListBox to hold the names of the Genres





Add the code to fill it from the View to your Database Class.

```
public List<string> FillListBoxWithGenre() {

            var myCommand = new SqlCommand();
            myCommand = new SqlCommand("select genre from UniqueGenre",
Connection);
            //Create a list to hold all the genre, then pass it back to the
listbox on the form
            List<string> newgenre = new List<string>();
            Connection.Open();
            SqlDataReader reader = myCommand.ExecuteReader();
            //loop through the genres and pass it to a reader, that gets
added to the list
            if (reader.HasRows) {
                while (reader.Read()) {
                    newgenre.Add(reader["genre"].ToString());
                    }
                }
            reader.Close();
            Connection.Close();
            return newgenre; //send the list back to the listbox

            }
```

Your Form Code

```
private void DisplayListBox() {
            //clear old data out
            DGVCD.DataSource = null;
            lbgenre.DataSource = null;
            try {
        lbgenre.DataSource = myDatabase.FillListBoxWithGenre();
                } catch (Exception ex) {
                MessageBox.Show(ex.Message);
                }
            }
```

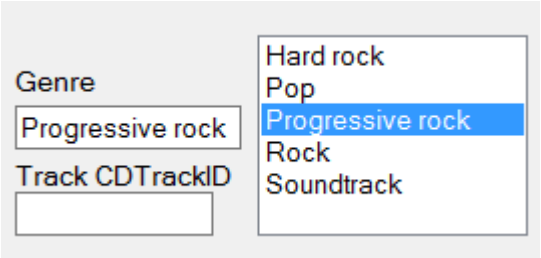
Add DisplayListBox() to the loadDB() method so it runs at startup.

```
public void loadDB() {
            //just to show the listbox with the genres in it
            DisplayListBox();
```

Double click on your ListBox on the form to generate a lbgenre_SelectedIndexChanged event

```
private void lbgenre_SelectedIndexChanged(object sender, EventArgs e) {
             txtCDGenre.Text = lbgenre.SelectedItem.ToString();
            }
```

Now when you click on a genre it gets added to the Genre textbox. This will make it quicker to add new data.



Using Views in the Form

The Views are just like tables. So replace the SQL table with the View name like this

```
SQL = "select * from GroupByGenre"
```

After much pondering and experimentation I found that although you can name your views **Owner-CD** in the Management Studio, Visual Studio or SQL doesn't like the Hyphen so rename the view to **OwnerCD**

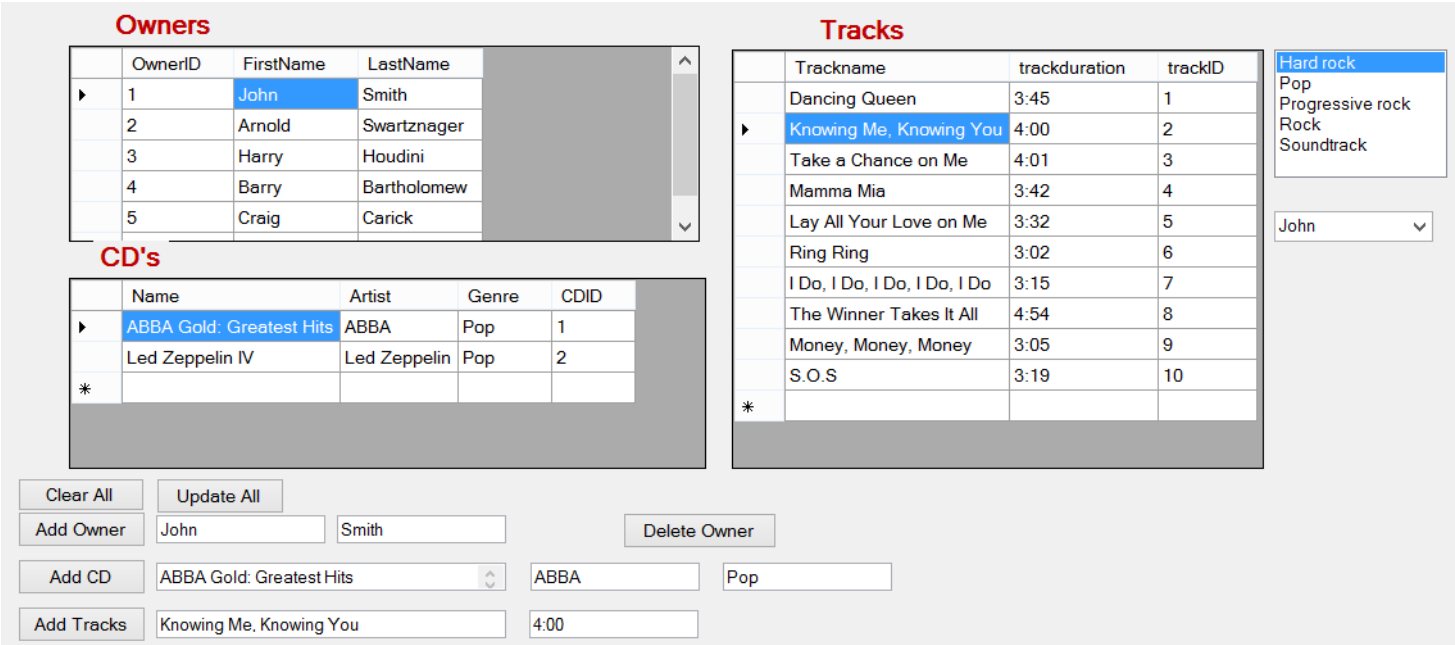
```
SQL = "select * from OwnerCD"
```

Linking DataGridViews to each other

In the image below you can see 3 DataGridView, each drawing from a different table. Owners, CD's and CD Tracks.

When you click on an owner in the Owners DGV the second DGV CD's shows the CD's that the owner has.

When you click on a CD in the second DGV you see the tracks of the CD in the third DGV. A nice click based navigation system.



The Names, CD, and tracks clicked on also fill the text boxes at the bottom.

There are two parts to making this magic happen.

1. Fill a DGV with data from a table.
2. Create a click event so clicking on a record sets the next DGV. Both of these are easy to create.

Lets create the **Owner** DGV. The other two DGVs' **Music** and **Tracks** are virtually identical so you can copy this and use it later changing the names in the code. You will recognise the first DGV from the previous exercise.

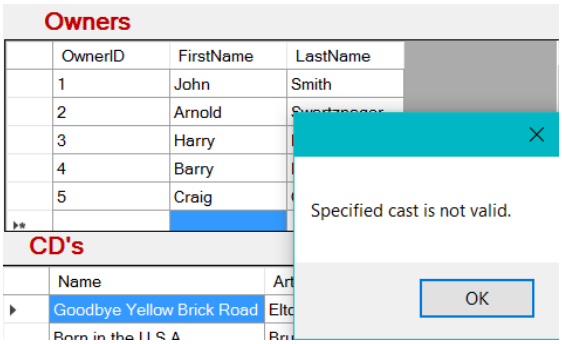
The Owner DataGridView Click Event

Double click on the DGVOwner to load the code for the click event.

Remember to change the Event trigger for DGVOwner to CellClick

The code is nested in a **Try Catch** in case you click in a place that doesn't return a value, otherwise it will make an error.

```
} catch (Exception ex) {
  MessageBox.Show(ex.Message);
                }
```



The heart of this code is this line which is returns every cell you click on.

```
OwnerID = (int)DGVOwner.Rows[e.RowIndex].Cells[0].Value;
txtFN.Text =   DGVOwner.Rows[e.RowIndex].Cells[1].Value.ToString();
txtLN.Text =   DGVOwner.Rows[e.RowIndex].Cells[2].Value.ToString();
```

We want everything from the row that you click on Rows[e.RowIndex] but we need to extract out the data in each row. Columns start at 0, so its 0, 1, 2.

Cells[0] Cells[1] Cells[2]

OwnerID	FirstName	LastName
1	John	Smith
2	Arnold	Swartznager
3	Harry	Houdini

The **OwnerID** is in the first column Cells[0],

The **First Name** is in the second column. Cells[1]

The **Last Name** is in the 3rd column. Cells[2]

Before we can make the method for the click event we have to create the Database call method in the Class first.

To see the method above click here and bypass the next lesson at your peril.

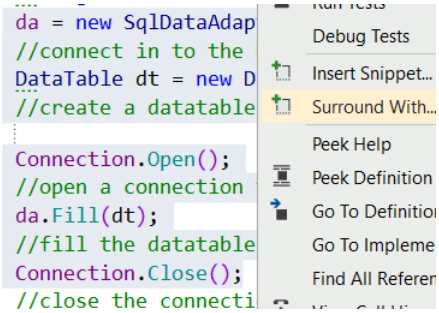
Fill the DataGridView With The Owner Click

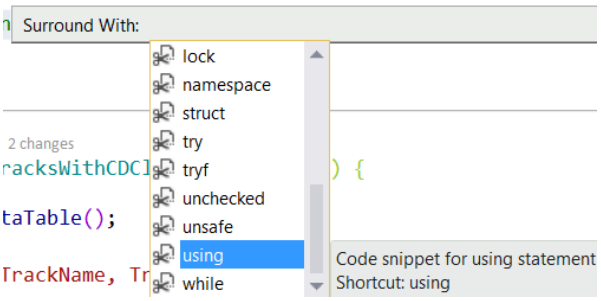
You need to make the method below in your **Database Class** first.

```
public DataTable FillDGVCDWithOwnerClick(string Ownervalue) {
        string SQL = "select Name, Artist, Genre, CDID from CD where OwnerIDFK = '" +
Ownervalue + "' ";
        da = new SqlDataAdapter(SQL, Connection);
        //connect in to the DB and get the SQL
        DataTable dt = new DataTable();
        //create a datatable as we only have one table, the Owner

        Connection.Open();
        //open a connection to the DB
        da.Fill(dt);
        //fill the datatable from the SQL
        Connection.Close();
        //close the connection

        return dt;
        }
```

This Method myDatabase.FillDGVCDWithOwnerClick(OwnerID.ToString()); gets all the data and sends it back as a DataTable to the DGVCD,

This needs to be wrapped with a **Using** Statement below. The easiest way is to highlight the part you want wrapped, Right Click and choose Surround With. Then scroll down and choose **Using**.






```
public DataTable FillDGVCDWithOwnerClick(string OwnerID)
        {
            string SQL = "select Name, Artist, Genre, CDID from CD where OwnerIDFK = '" + OwnerID + "' ";
            using (da = new SqlDataAdapter(SQL, Connection))
            {
                //connect in to the DB and get the SQL
                DataTable mydt = new DataTable();
                //create a datatable as we only have one table, the Owner
                Connection.Open();
                //open a connection to the DB
                da.Fill(mydt);
                //fill the datatable from the SQL
                Connection.Close();
                //close the connection

                return mydt;
            }
        }
```

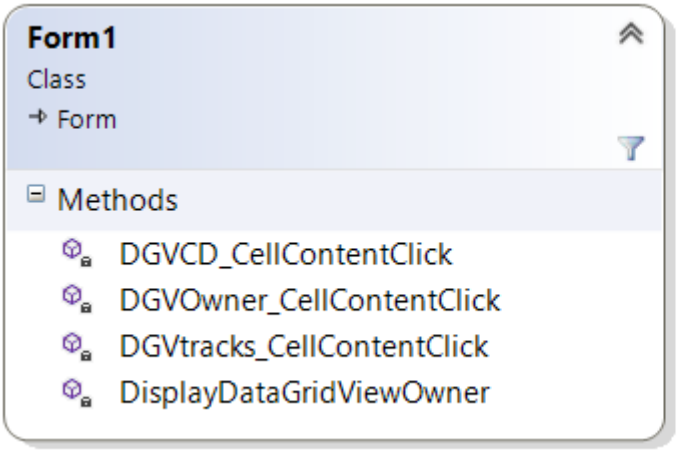
DGV Owner Cell Content Click event

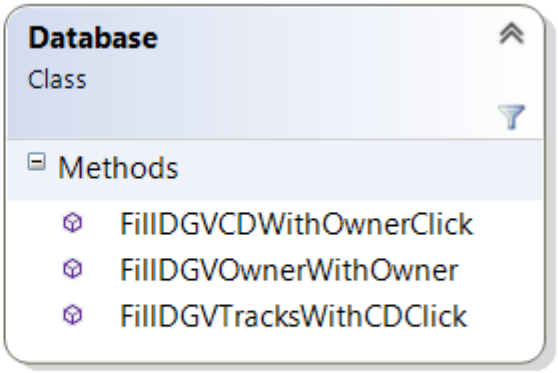
Here is the code for the DGVOwner_CellContentClick DataGridView click event.

```
private void DGVOwner_CellContentClick(Object sender, DataGridViewCellEventArgs e) {
          int OwnerID = 0;
//these are the cell clicks for the values in the row that you click on
           try {
OwnerID = (int)DGVOwner.Rows[e.RowIndex].Cells[0].Value;
txtFN.Text = DGVOwner.Rows[e.RowIndex].Cells[1].Value.ToString();
txtLN.Text = DGVOwner.Rows[e.RowIndex].Cells[2].Value.ToString();
               //if you are clicking on a row and not outside it
if (e.RowIndex >= 0) {
                   //Fill the next CD DGV with the OwnerID
DGVCD.DataSource = myDatabase.FillDGVCDWithOwnerClick(OwnerID.ToString());
DGVCD.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
                   TxtOwnerID.Text = OwnerID.ToString();
                   }
                } catch (Exception ex) {
               MessageBox.Show(ex.Message);
               }
           }
```

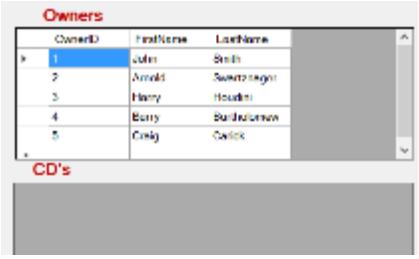
So what about the other DGV?

Their code is very similar to the ones before.



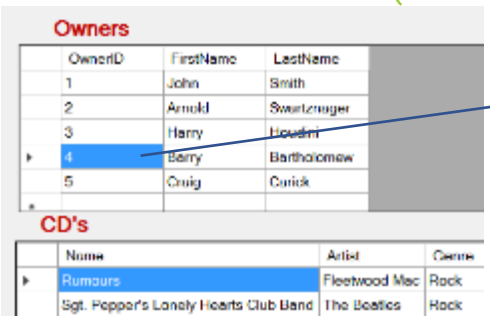


So you First run `DisplayDataGridViewOwner()` this fills the DGV with all the owners.




```
public DataTable FillDGVOwnerWithOwner() ...
```

Then you click on an owner `DGVOwner_CellContentClick` and you fill the `FillDGVCDWithOwnerClick(string OwnerID)`.with the owner you clicked on.

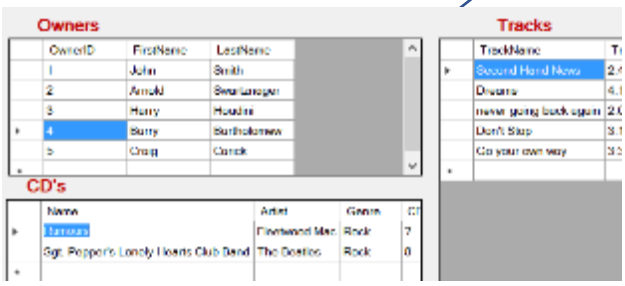


We need the OwnerID 4
`string OwnerID`

```
public DataTable FillDGVCDWithOwnerClick(string OwnerID) ...
```

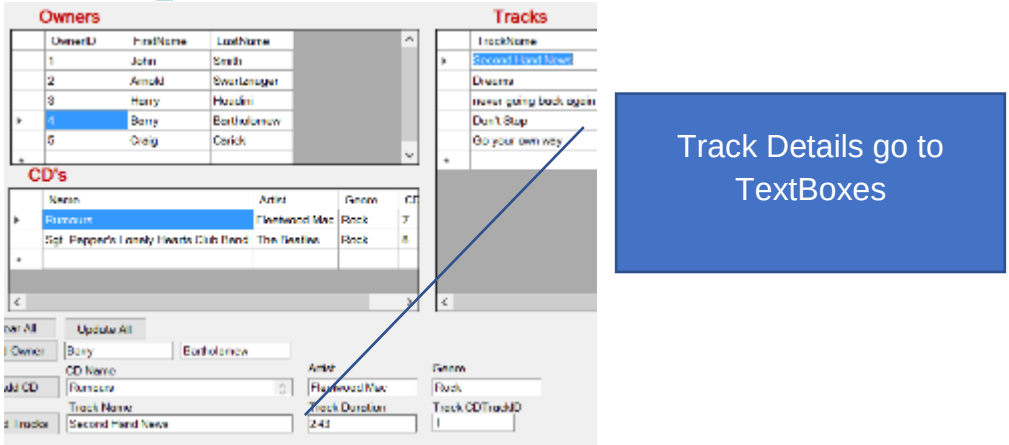
Then you click on a CD `DGVCD_CellContentClick` and you fill the Tracks Table `FillDGVTracksWithCDClick(string CDID)`

We need the CDID 7
`string CDID`




```
public DataTable FillDGVTracksWithCDClick(string CDID) ...
```

Then we click on a Track and it puts the details in the Track Name textbox.
DGVtracks_CellContentClick



Here are the next methods from the **Database Class**

```
1 reference | GaryDix, 3 days ago | 1 author, 2 changes
public DataTable FillDGVOwnerWithOwner() ...
5 references | GaryDix, 8 days ago | 1 author, 1 change
public DataTable FillDGVCDWithOwnerClick(string OwnerID) ...
5 references | GaryDix, 7 days ago | 1 author, 2 changes
public DataTable FillDGVTracksWithCDClick(string CDID) ...
```

Create the CD and Tracks DataGridView Click events.

Firstly you will need to duplicate the Owners DatagridView and Class calls.

Next create the **CD click event**. Use the click event for the Owner cell and just change the code as well as change the name of your DataGridView to the one for your CD grid.

In this way you have daisy chained together the three DataGridView's to create a full application.

Finally it would be nice to be able to click on the track and have it play but that is beyond the scope of this program.

Resizing Columns to fit contents

This seems to be a hit or miss issue but I found

```
DGVmusic.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.DisplayedCells)
```

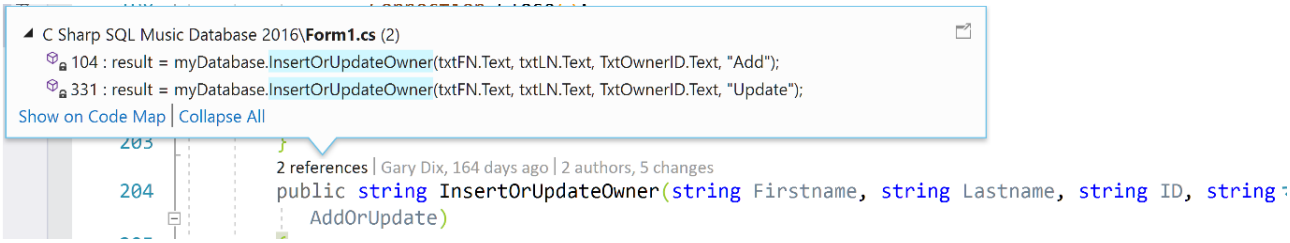
Or the following to have some success.

```
DGVmusic.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells
```

Update and Delete Queries

The Insert and Update are very similar so I combined them into one rather lengthy method. I am not sure if it's a recommended idea, as smaller methods might be tidier.

In my code I called the method twice and you can see from the parameters the fields that I pass through



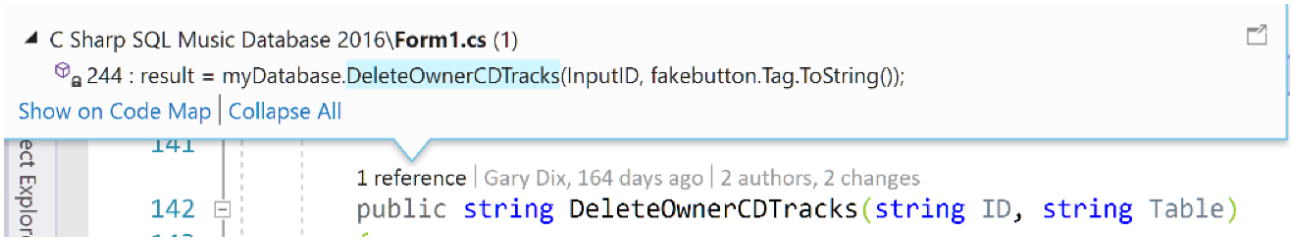
First name, Last name, ID and the text Add or Update.

```
public string InsertOrUpdateOwner(string Firstname, string Lastname, string ID, string
AddOrUpdate)
        {
            try
            {
//Add gets passed through the parameter
                if (AddOrUpdate == "Add")
                {
//Create a Command object  //Create a Query. Create and open a connection to SQL Server
                    string query = "INSERT INTO Owner (FirstName, LastName) " +
"VALUES(@Firstname, @Lastname)";

                    var myCommand = new SqlCommand(query, Connection);
                    //create params
                    myCommand.Parameters.AddWithValue("Firstname", Firstname);
                    myCommand.Parameters.AddWithValue("Lastname", Lastname);
                    Connection.Open();
                    // open connection add in the SQL
                    myCommand.ExecuteNonQuery();
                    Connection.Close();
                }
//Update gets passed through the parameter
                else if (AddOrUpdate == "Update")
                {
                    var myCommand = new SqlCommand("UPDATE Owner set FirstName = @Firstname,
LastName=@Lastname where OwnerID = @ID ", Connection);
                    //use parameters to prevent SQL injections
                    myCommand.Parameters.AddWithValue("Firstname", Firstname);
                    myCommand.Parameters.AddWithValue("Lastname", Lastname);
                    myCommand.Parameters.AddWithValue("ID", ID);
                    Connection.Open();
                    // open connection add in the SQL
                    myCommand.ExecuteNonQuery();
```

```
                Connection.Close();
            }

            return " is Successful";
        }
        catch (Exception e)
        {
//need to get it to close a second time as it jumps the first connection.close if
ExecuteNonQuery fails.
            Connection.Close();
            return " has Failed with " + e;
        }
    }
```

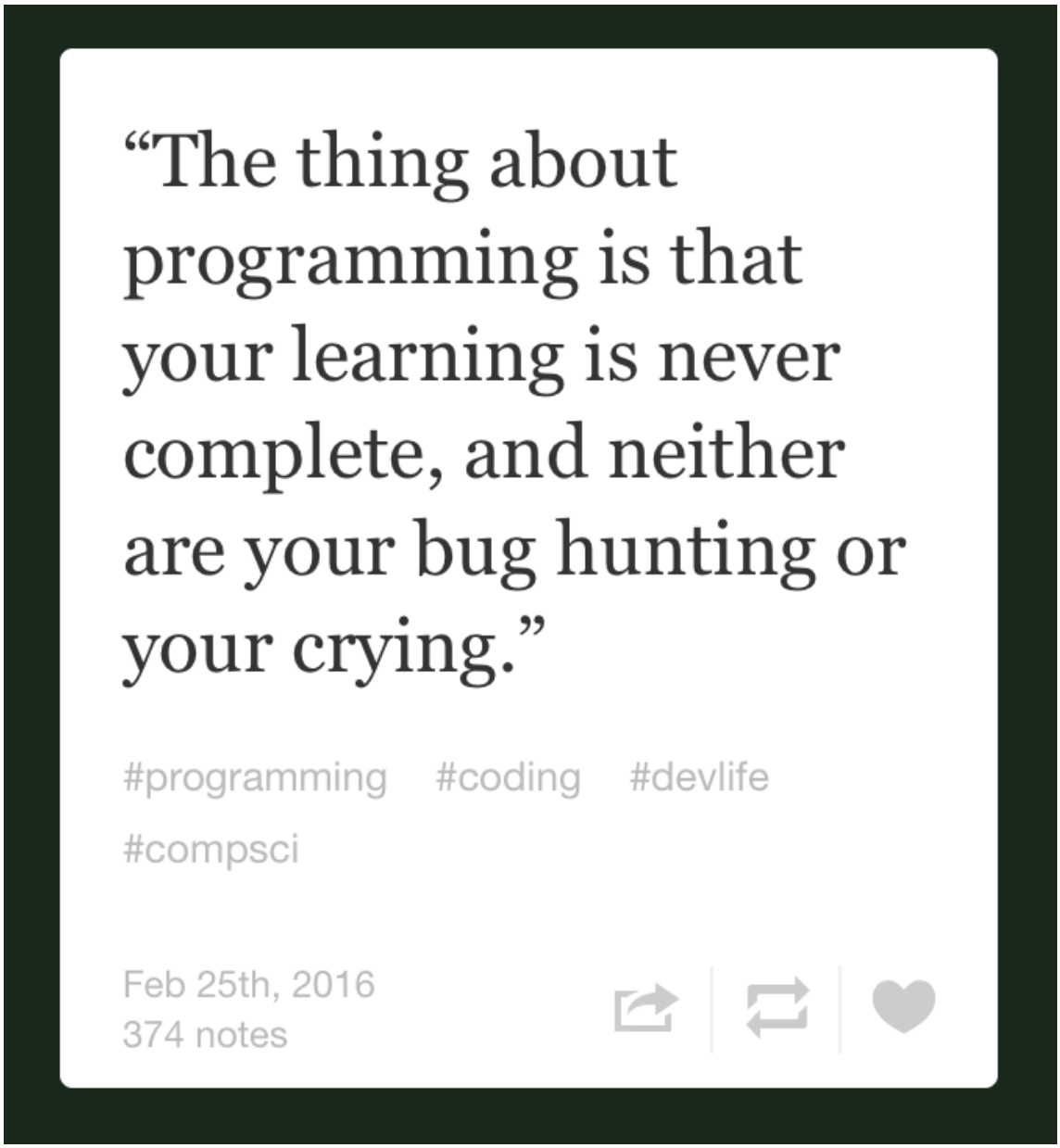
Delete is really easy, all you need is the ID of the record you want to delete and pass it through

This is called by one method, but you can call it by all four methods. I pass through the ID and the name of the table (Owner, CD, Track) that I want deleted.




```
public string DeleteOwnerCDTracks(string ID, string Table)
    {
        //only run if there is something in the textbox
        if (!object.ReferenceEquals(ID, string.Empty))
        {
            var myCommand = new SqlCommand();
            switch (Table)
            {
                case "Owner":
myCommand = new SqlCommand("DELETE FROM Owner WHERE OwnerID = @ID", Connection);
                    break;
                case "CD":
myCommand = new SqlCommand("DELETE FROM CD WHERE CDID = @ID", Connection);
                    break;
                case "Track":
myCommand = new SqlCommand("DELETE FROM CDTracks WHERE TrackID = @ID", Connection);
                    break;
            }

            myCommand.Parameters.AddWithValue("ID", ID);
            //use parameters to prevent SQL injections
```

```
            Connection.Open();
            // open connection add in the SQL
            myCommand.ExecuteNonQuery();
            Connection.Close();
            return "Success";
        }
        else
        {
            Connection.Close();
            return "Failed";
        }
```



27. Preventing SQL Injections

Insert and delete are two areas where SQL Injection attacks can become an issue, ie: people hacking your database and futzing with your records.

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker) Yes, but what does SQL Injection mean without jargon?

> Imagine you're a robot in a warehouse full of boxes. Your job is to fetch a box from somewhere in the warehouse, and put it on the conveyor belt. Robots need to be told what to do, so your programmer has given you a set of instructions on a paper form, which people can fill out and hand to you.
>
> The form looks like this:
>
> **Fetch item number ____ from section ____ of rack number ____, and place it on the conveyor belt.**
>
> A normal request might look like this:
>
> Fetch item number **1234** from section **B2** of rack number **12**, and place it on the conveyor belt.
>
> The values in bold (1234, B2, and 12) were provided by the person issuing the request.
>
> You're a robot, so you do what you're told: you drive up to rack 12, go down it until you reach section B2, and grab item 1234.
>
> You then drive back to the conveyor belt and drop the item onto it.
>
> But what if a user put something other than normal values into the form? **What if the user added instructions into them?**
>
> Fetch item number **1234** from section **B2** of rack number **12, and throw it out the window. Then go back to your desk and ignore the rest of this form.** and place it on the conveyor belt.
>
> Again, the parts in bold were provided by the person issuing the request. Since you're a robot, you do exactly what the user just told you to do.
>
> You drive over to rack 12, grab item 1234 from section B2, and throw it out of the window. Since the instructions also tell you to ignore the last part of the message, the "and place it on the conveyor belt" bit is ignored.
>
> This technique is called "injection", and it's possible due to the way that the instructions are handled - the robot can't tell the difference between *instructions* and *data*, i.e. the actions it has to perform, and the things it has to do those actions on.
>
> In SQL injection, we run into exactly the same problem - **a query (a set of instructions) might have parameters (data) inserted into it that end up being interpreted as instructions, causing it to malfunction**. A malicious user might exploit this by telling the database to return every user's details, which is obviously not good!

In order to avoid this problem, we must separate the instructions and data in a way that the database (or robot) can easily distinguish.

This is usually done by sending them separately.

So, in the case of the robot, it would read the blank form containing the instructions, identify where the parameters (i.e. the blank spaces) are, and store it.

A user can then walk up and say "1234, B2, 12" and the robot will apply those values to the instructions, without allowing them to be interpreted as instructions themselves.

In SQL, this technique is known as parameterised queries.

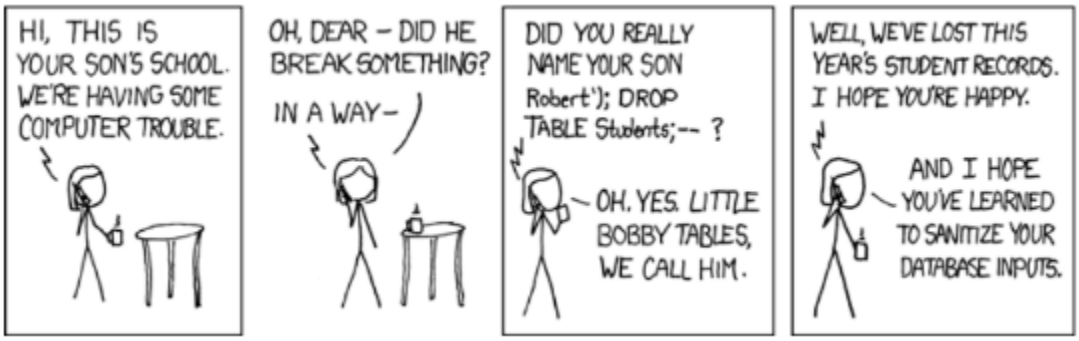
In the case of the "evil" parameter we gave to the robot, he would now raise a mechanical eyebrow quizzically and say

Error: Cannot find rack number "**12, and throw it out the window. Then go back to your desk and ignore the rest of this form.**" - are you sure this is a valid input?

Success! We've stopped the robot's "glitch".

Little Bobby Tables

This famous cartoon shows it working, the user replaced the Surname of the student Robert with a command to delete the Students table from the database Drop Table Students



In SQL, commands are terminated by semicolons **;** and data is often quoted using single quotes **'**. Commands may also be enclosed in parentheses **(** and **)**.

Data is stored in tables of similar items (e.g. students) and individual entries are "rows" in the table.

To delete an entire table (and every row of data in that table), you use the command DROP (e.g. DROP TABLE students). The **--** represents the start of a SQL comment which ensures that the rest of the command is ignored so an error will not occur. `Robert'); DROP TABLE students;--`

The exploited vulnerability is that the single quote `'` in the name input was not properly "escaped" by the software.

Thus, when the name is embedded into some SQL statement, the quote is erroneously parsed as a closing quote inside that statement, rather than being parsed as part of the

name. Lack of such escaping is a common SQL vulnerability; this type of exploit is referred to as SQL injection. https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables

A typical, unsecured SQL command vulnerable to SQL injection would be something like:

```
INSERT INTO students (name) VALUES ('" + name + "');
```

This would result in the following SQL command to be send to the database system:

```
INSERT INTO students (name) VALUES ('Elaine');
```

However, with Little Bobby Tables' full name `Robert'); DROP TABLE students;--`, the SQL command would be:

```
INSERT INTO students (name) VALUES ('Robert'); DROP TABLE students;--');
```

Or, if split after each **;**:

```
INSERT INTO students (name) VALUES ('Robert');
DROP TABLE students;
--');
```

How can we stop SQL Injection it in our projects?

Preventing SQL Injection attacks == Using Parameters in the manual == Using Stored Procedures in the manual.

However using Parameters **cmd.Parameters.AddWithValue("@Parameter", txtTextBox1.Text);** does not solve everything because the value inserted isn't restricted to a type, it goes in as an **object**.

You want a string? Make sure that ONLY a string can be added in.

Read this article here, and how to specify the Type so that errors, and attacks are prevented. Can we stop using AddWithValue() already?

The SQL Injection CheatSheet!

SQL Injection Cheat Sheet

An SQL injection cheat sheet is a resource in which you can find detailed technical information about the many different variants of the SQL Injection vulnerability. This cheat sheet is of good reference to both seasoned penetration tester and also those who are just getting started in web application security.

Who's dumb enough to do that today?

We can even hack into Card controlled doors via the blinking LED with Injection.

http://blog.trendmicro.com/let-get-door-remote-root-vulnerability-hid-door-controllers/

Creating a Database Unit Test

http://www.codeproject.com/Articles/841250/Create-SQL-Server-Database-Unit-Tests

https://msdn.microsoft.com/en-us/library/aa833283%28v=vs.100%29.aspx

Unresolved reference errors solved

http://stackoverflow.com/questions/25716756/unresolved-references-to-same-database-project

https://social.msdn.microsoft.com/Forums/sqlserver/en-US/1863d960-d32d-4920-9a30-13dc86c6f857/sql71562-unresolved-reference-to-object-followd-by-database-name-in-the-same-project?forum=ssdt&prof=required