(T22)比較 OptimisticConcurrency、PessimisticConcurrency。討論 Rowversion 解決 ChangeConflictException
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c
==========
(T22)比較 OptimisticConcurrency、PessimisticConcurrency。討論 Rowversion 解決 ChangeConflictException
==========
0. Summary
---
1. optimistic concurrency control V.S. pessimistic concurrency control
---
2. Web Form Application - Linq Query
2.1. TSQL
2.2. Set up SQL Authentication
---
3. Create Web Application
3.1.Web.config
3.2. Linq to SQL
3.2.1. Add Connection
3.2.2. Sample.dbml
3.3. WebForm1.aspx
3.3.1. WebForm1.aspx
3.3.2. WebForm1.aspx.cs
3.4. Run WebForm1.aspx
3.5. Run WebForm1.aspx with update conflicts
3.5.1. WebForm1.aspx with KeepCurrentValues
3.5.2. WebForm1.aspx with KeepChanges
3.5.3. WebForm1.aspx with OverwriteCurrentValues
---
4. UpdateCheck Property
4.1. TSQL (UpdateCheck Property)
4.2. WebForm1.aspx (UpdateCheck Property)
4.3. SQL Profiler (UpdateCheck Property)
4.4. DBML (UpdateCheck Property)
4.5. WebForm1.aspx (UpdateCheck Property)
4.6. SQL Profiler (UpdateCheck Property)
4.7. DBML (UpdateCheck Property)
---
5. Rowversion and Timestamp
5.1. TSQL (Rowversion and Timestamp)
5.2. WebForm1.aspx (Rowversion and Timestamp)
5.3. SQL Profiler (Rowversion and Timestamp)
5.4. Add Rowversion or Timestamp (Rowversion and Timestamp)
5.5. DBML (Rowversion and Timestamp)
5.6. WebForm1.aspx (Rowversion and Timestamp)
5.7. SQL Profiler (Rowversion and Timestamp)
==========

0. Summary

* Linq to SQL Concurrency, TSql Concurrency, C# Thread Async-Await 一直都是一個門檻，如果 UserA update 一個 data，然後 UserB update 同一個 data 在同一個時間，那該怎辦?

這邊介紹了 Linq to Sql Concurrency 要怎麼處理 ChangeConflictException。
* 一開始介紹了，optimistic(樂觀的) concurrency control V.S. pessimistic(悲觀的) concurrency control，後來介紹 3 種 ChangeConflictException 的處理方式，KeepCurrentValues V.S. KeepChanges V.S. OverwriteCurrentValues，然後接下來開始討論效能 Performance，介紹了 UpdateCheck property，然後為了要更好的 Performance，介紹了 Rowversion 的用法。如果你以前沒有 fully understand 3 種 ChangeConflictException 的處理方式，這個 Video 會對你非常有幫助。

1.
optimistic concurrency control(樂觀並行控制) V.S. pessimistic concurrency control(悲觀並行控制)
1.1.
Pessimistic concurrency control
Using rows lock to prevent other user from modifying the same data at the same time.
When lock owner lock the rows,
no one else can access the rows
until the lock owner release the lock.
Lock always has performance issue,
so Pessimistic concurrency is no good for performance.
1.2.
Optimistic concurrency control
It does not use rows lock.
If 2 users tried to update the same data at the same time,
userA's changes will be committed
and userB's changes will be discarded.
In addition, an exception will be thrown to notify the userB.
By default, Linq to Sql uses optimistic concurrency to handle concurrent updates.

2.
RefreshMode enum in optimistic concurrency control
has 3 different options to handle **ChangeConflictException**.
KeepCurrentValues V.S. **KeepChanges** V.S. **OverwriteCurrentValues**

2.1.
KeepCurrentValues
KeepCurrentValues means keep all the new values from the current user.
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
<u>KeepCurrentValues means keep all the new values from the current user.</u>
dbContextA is used by **current user**
and tries to update the **Column1**.
In the meantime, **dbContextB** is used by **another user**
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextA** will be saved into Database.
The **old** value of **Column2** from **dbContextA** will be saved into Database.

2.2.
KeepChanges
KeepChanges means keep all the changes from all the users.
If any conflict, then keeps the new values from the current user.
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepChanges);
dbContextA is used by **current user**

and tries to update the **Column1**.
In the meantime, **dbContextB** is used by **other user**
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextA** will be saved into Database.
The **new** value of **Column2** from **dbContextB** will be saved into Database.

2.3.
OverwriteCurrentValues
OverwriteCurrentValues means discard all the changes from the current user.
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.OverwriteCurrentValues);
<u>OverwriteCurrentValues means discard all the changes from the current user.</u>
dbContextA is used by **current user**
and tries to update the **Column1**.
In the meantime, **dbContextB** is used by **another user**
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextB** will be saved into Database.
The **new** value of **Column2** from **dbContextB** will be saved into Database.
Because OverwriteCurrentValues means discard all the changes from **dbContextA**.

3.
When **ChangeConflictException** happens,
we can access the following values.
3.1.
//memberChangeConflict.Member.Name
This will show you the property Name which contains change conflict data.
The property name is normally same as Column Name from the database.
3.2.
//memberChangeConflict.CurrentValue
The will show you the current value of the property which contains change conflict data.
This is the new value which updated by the current user.
3.3.
//memberChangeConflict.OriginalValue
The will show you the original value of the property which contains change conflict data.
This is the old value which has not been updated from the current user yet.
3.4.
//memberChangeConflict.DatabaseValue
The will show you the current value of the corresponding column in the database table.

4.
UpdateCheck property
UpdateCheck property can be set to
one of the 3 values of the UpdateCheck enum
which is in System.Data.Linq.Mapping namespace.
4.1.
Always
By default, "Always" use this column for conflict detection
4.2.
Never
"Never" use this column for conflict detection
4.3.

WhenChanged
Use this column only when the member has been changed by the application

==========

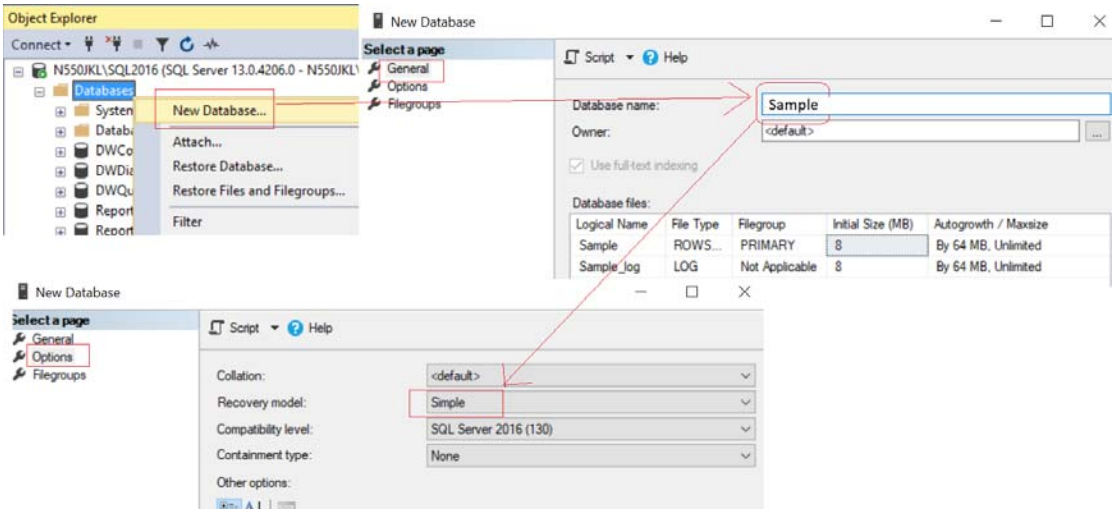
1. optimistic concurrency control V.S. pessimistic concurrency control

1.
optimistic(樂觀的) concurrency control V.S. pessimistic(悲觀的) concurrency control
1.1.
Pessimistic concurrency control
Using rows lock to prevent other users from modifying the same data at the same time.
When lock owner lock the rows,
no one else can access the rows
until the lock owner releases the lock.
Lock always has a performance issue,
so Pessimistic concurrency is no good for performance.
1.2.
Optimistic concurrency control
It does not use rows lock.
If 2 users tried to update the same data at the same time,
userA's changes will be committed
and userB's changes will be discarded.
In addition, an exception will be thrown to notify the userB.
By default, Linq to Sql uses optimistic concurrency to handle concurrent updates.

2. Web Form Application - Linq Query

2.1. TSQL

Database --> Right Click --> New Database -->
Database Name : Sample
Options --> Recovery Model : Simple



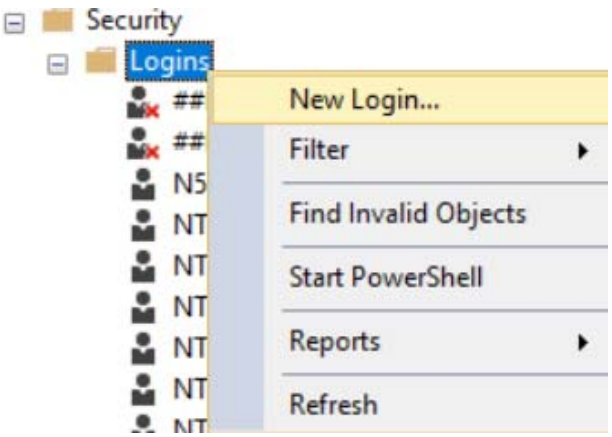
--Create a Sample DataBase and Run the following TSQL

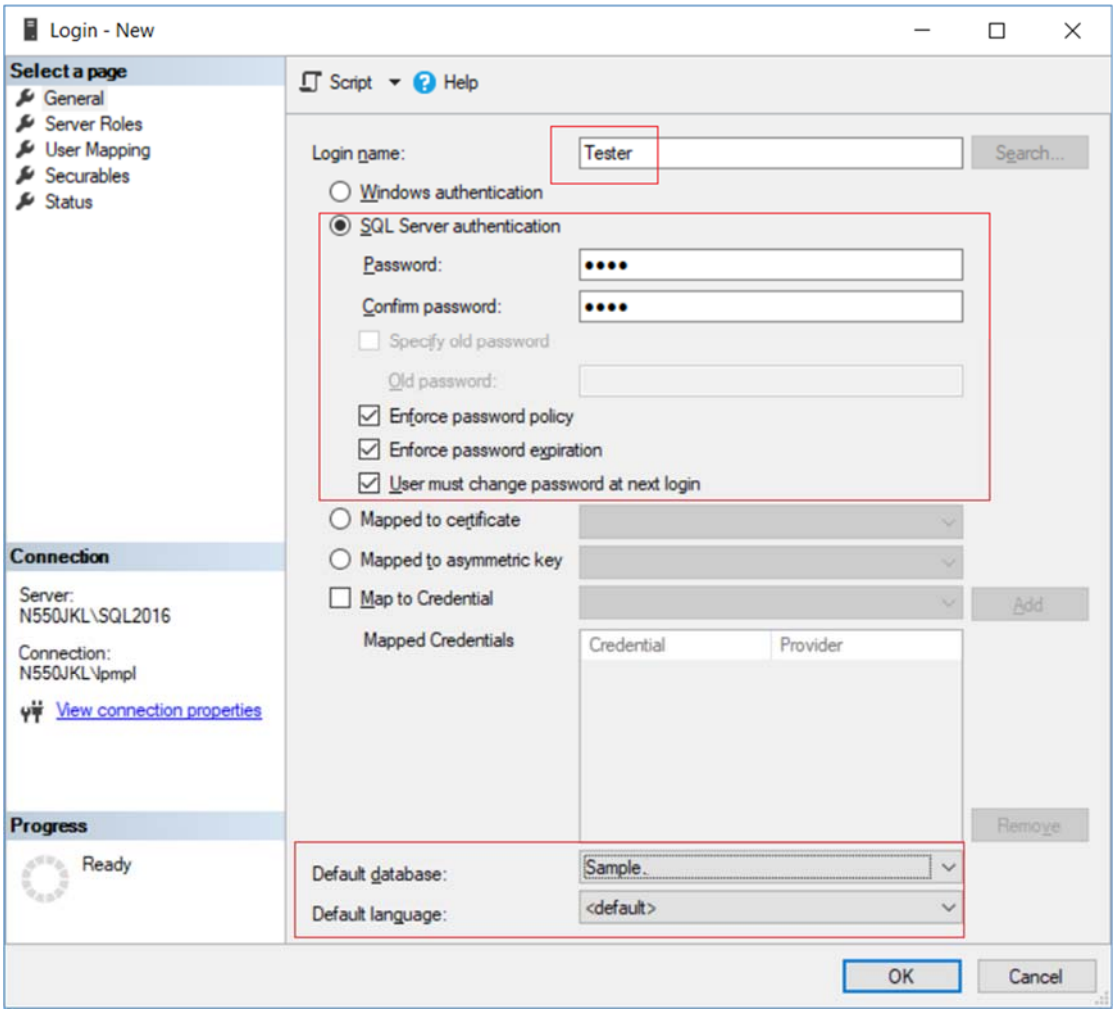
```
--1 -------------------------------------------------------
--Drop Table if it exists.
--IF OBJECT_ID('Gamer') IS NOT NULL
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE      TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
    (
      Id INT PRIMARY KEY
             IDENTITY ,
      Name NVARCHAR(50) ,
      Score INT ,
    );
GO -- Run the previous command and begins new batch
--2 -------------------------------------------------------
INSERT  INTO Gamer
VALUES  ( 'Name1 ABC', 5000 );
GO -- Run the previous command and begins new batch
```

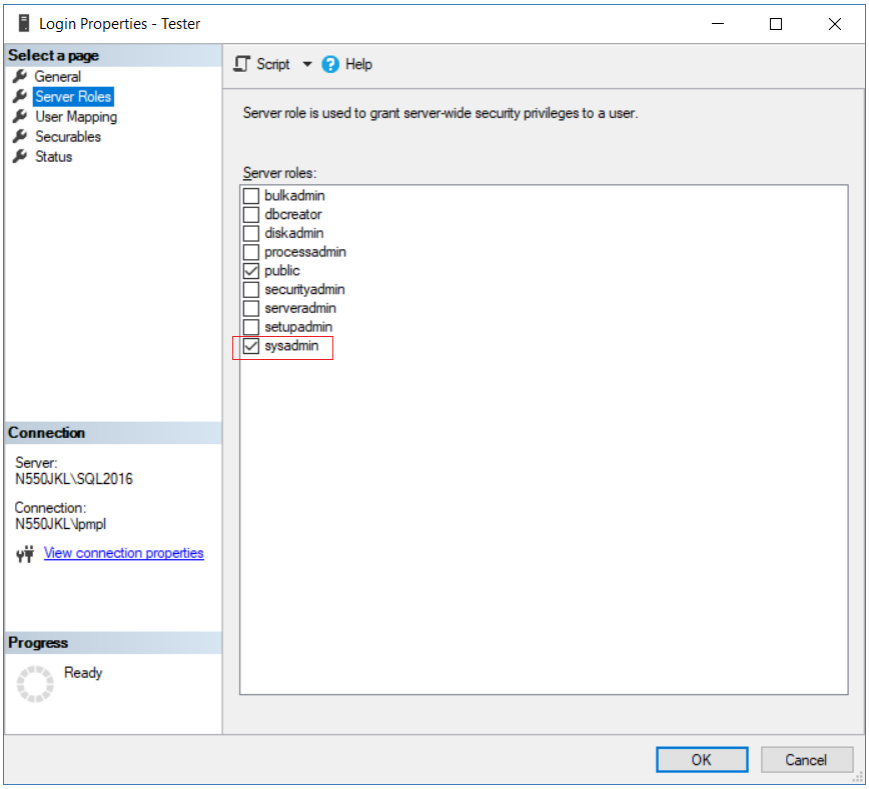
2.2. Set up SQL Authentication

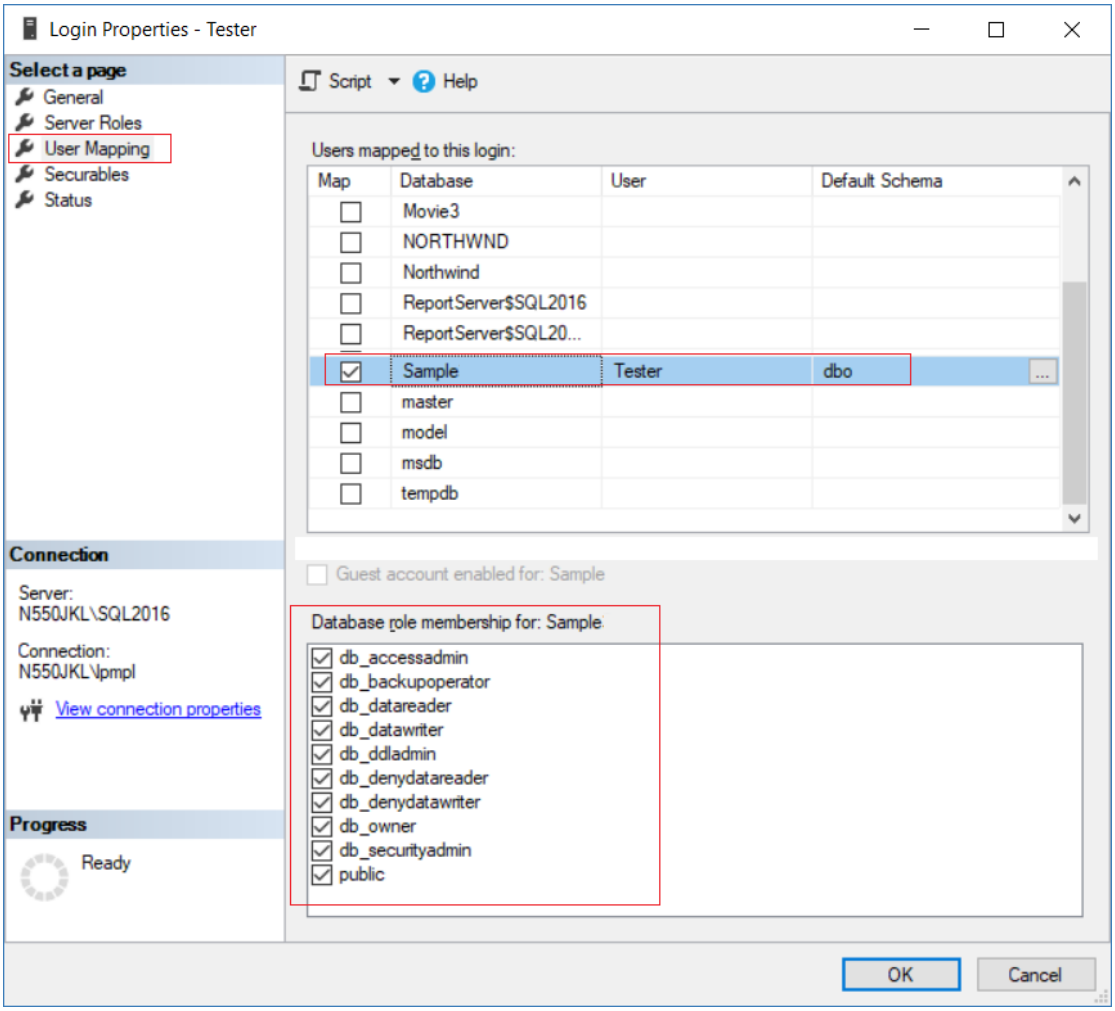
In SQL server
Object Explorer --> Security --> Logins --> New Logins
-->
General Tab
Login Name :
Tester
Password:
1234

Default Database:
Sample
-->
Server Roles Tab
Select
sysadmin
-->
User Mapping Tab
Select Sample
Select every Roles.





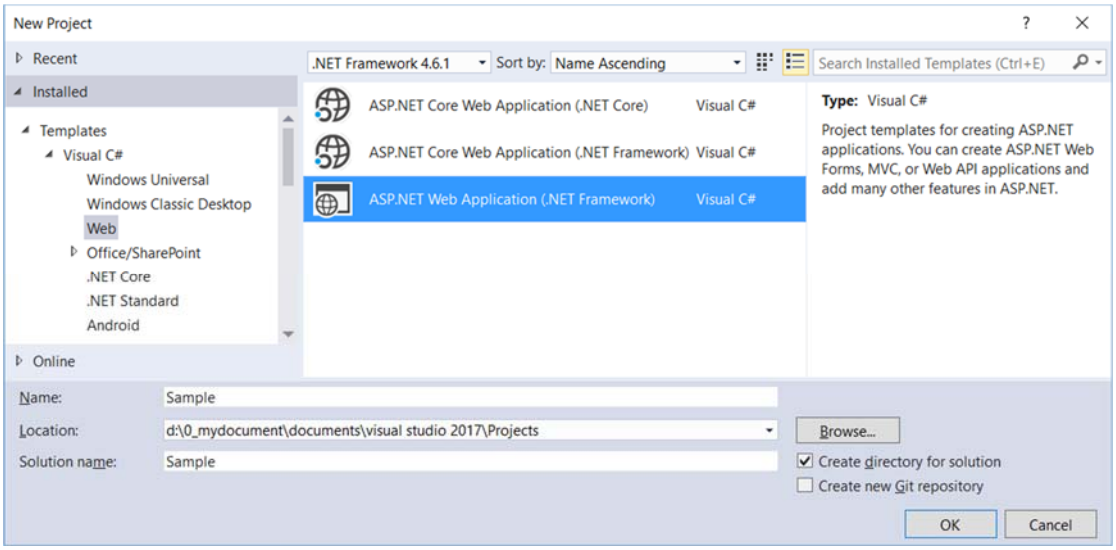


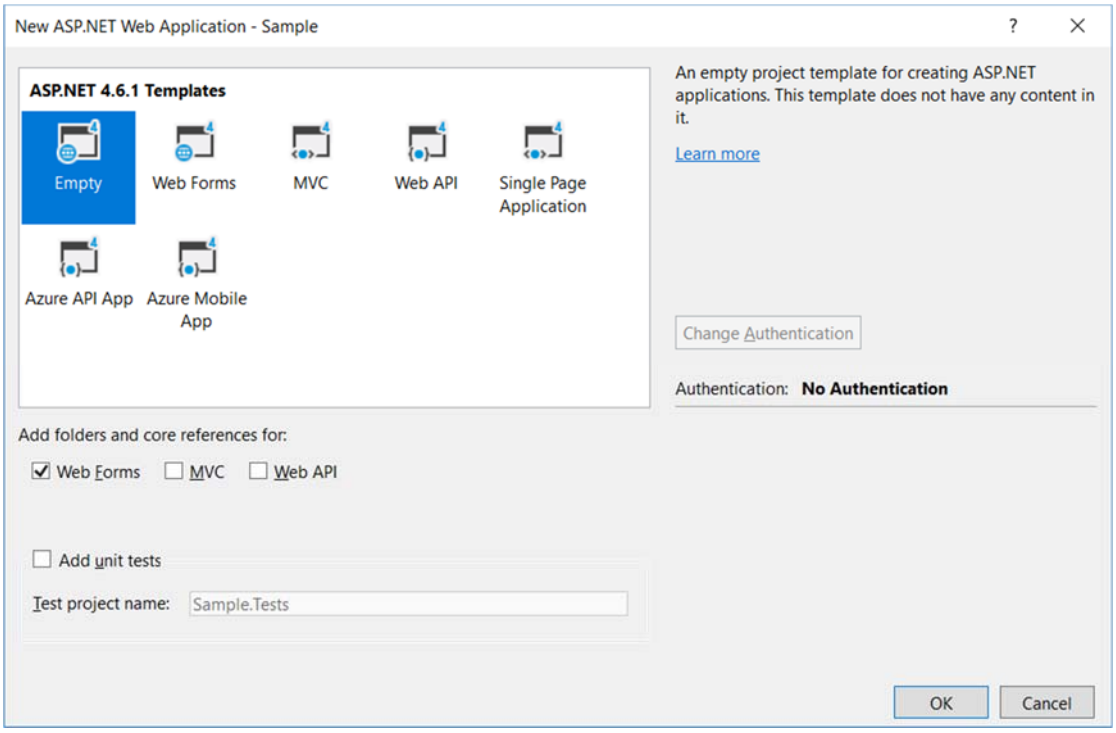


3. Create Web Application

Open Visual Studio, I am currently using VS2017
If you don't have it, you may follow the instruction here to download.
http://ithandyguytutorial.blogspot.com/2017/10/ch00install-visual-studio-2017-offline.html

New Project --> Web -->ASP.NET**Web Application (.Net Framework)**
-->
Name:
Sample
--> **Empty** --> Select "**Web Forms**" --> OK



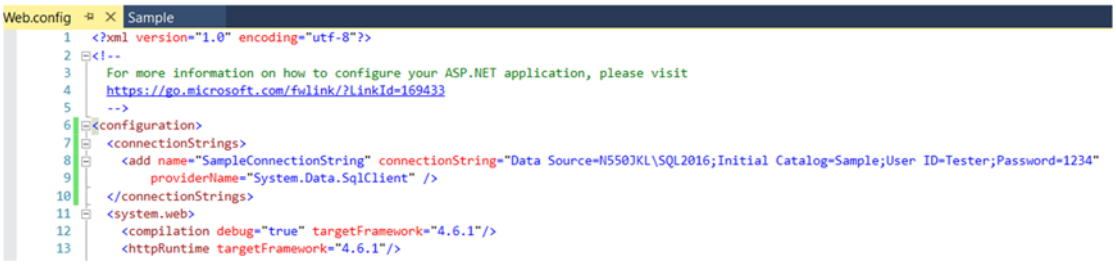


3.1.Web.config

Add connection String

If you use Linq to Sql, you don't have to set this connection string.
I personally already get used to set it on my own.

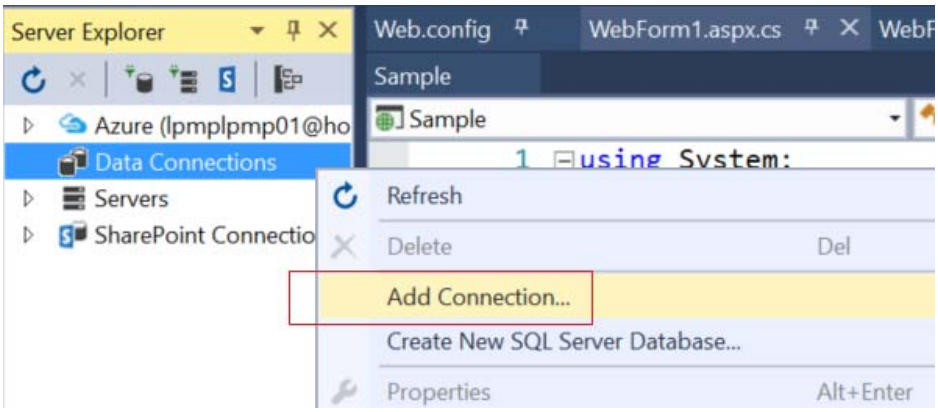
```
<configuration>
  <connectionStrings>
    <add name="SampleConnectionString" connectionString="Data Source=N550JKL\SQL2016;Initial
Catalog=Sample;User ID=Tester;Password=1234"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
```

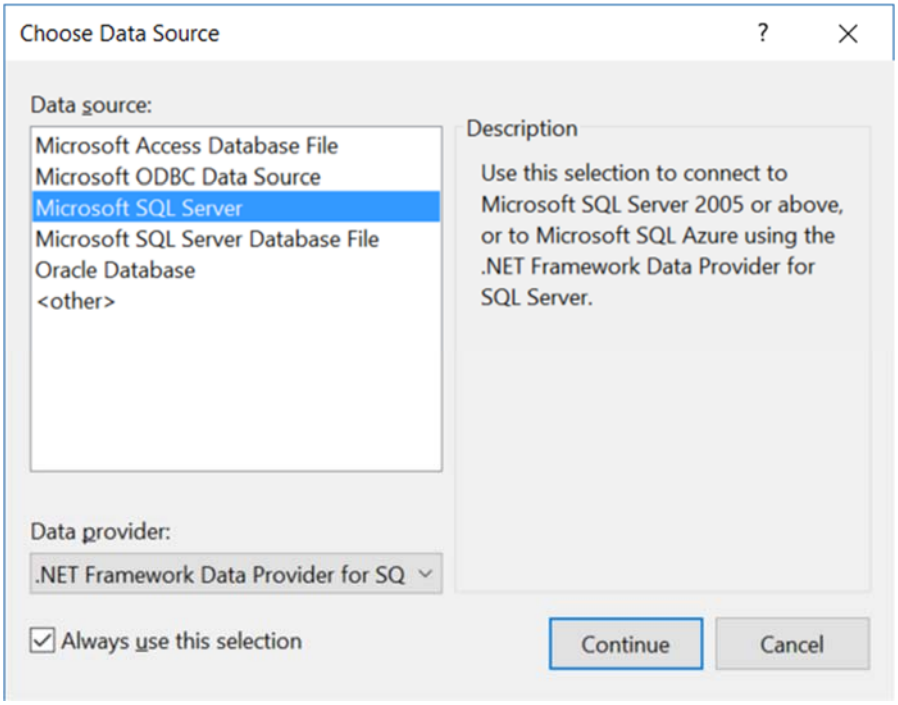


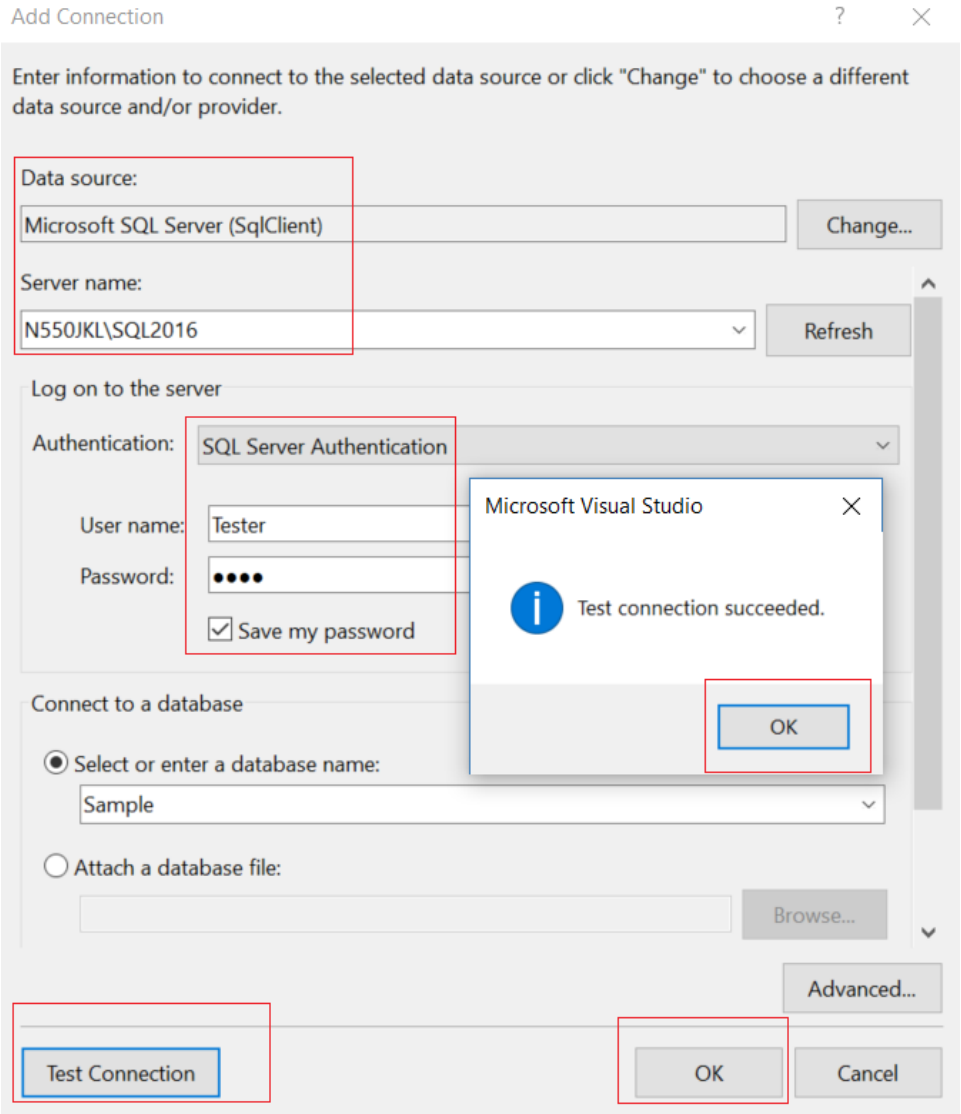
3.2. Linq to SQL

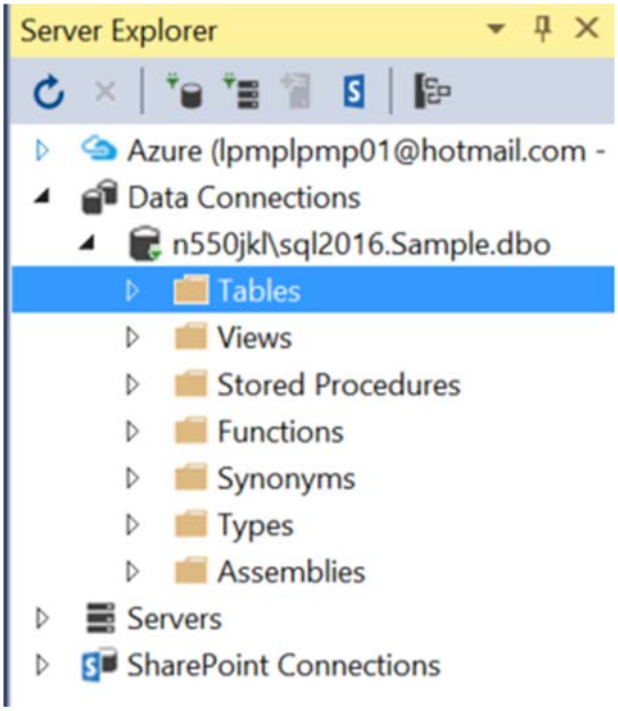
3.2.1. Add Connection

Server Explorer --> Data Connections --> Right click --> Add Connection...
--> Microsoft SQL server -->
Enter your server and database details ....




Choose Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.
Data provider:
.NET Framework Data Provider for SQ
Always use this selection
Continue
Cancel


Add Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
Server name:
N550JKL\SQL2016
Refresh
Log on to the server
Authentication:
SQL Server Authentication
User name:
Tester
Password:
Save my password
Microsoft Visual Studio
Test connection succeeded.
OK
Connect to a database
Select or enter a database name:
Sample
Attach a database file:
Browse...
Advanced...
Test Connection
OK
Cancel

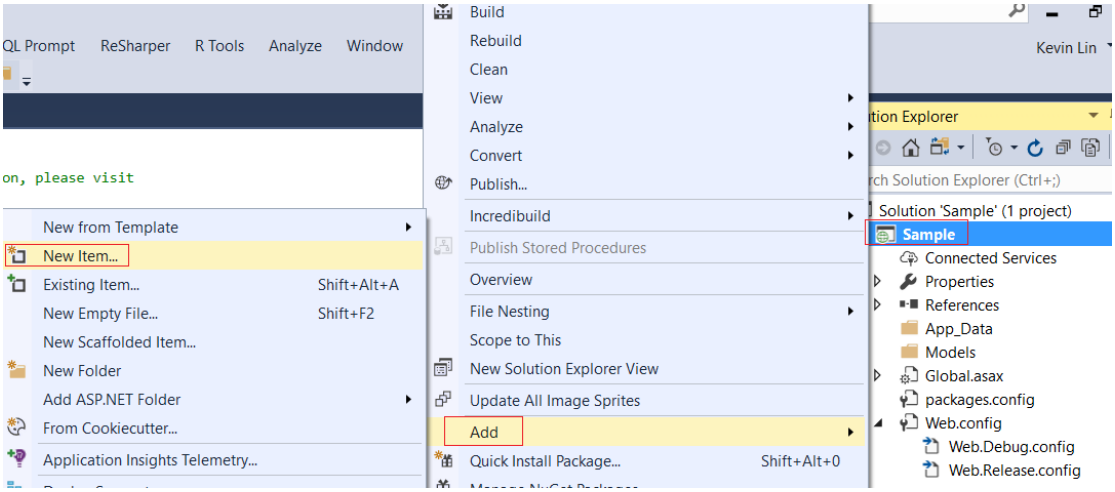


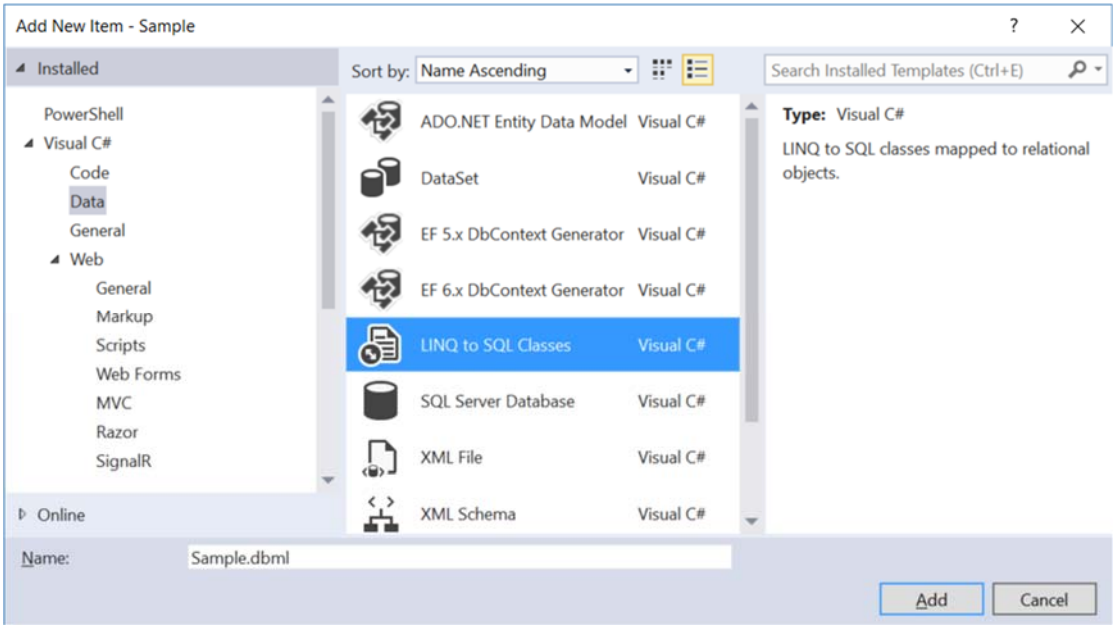
3.2.2. Sample.dbml

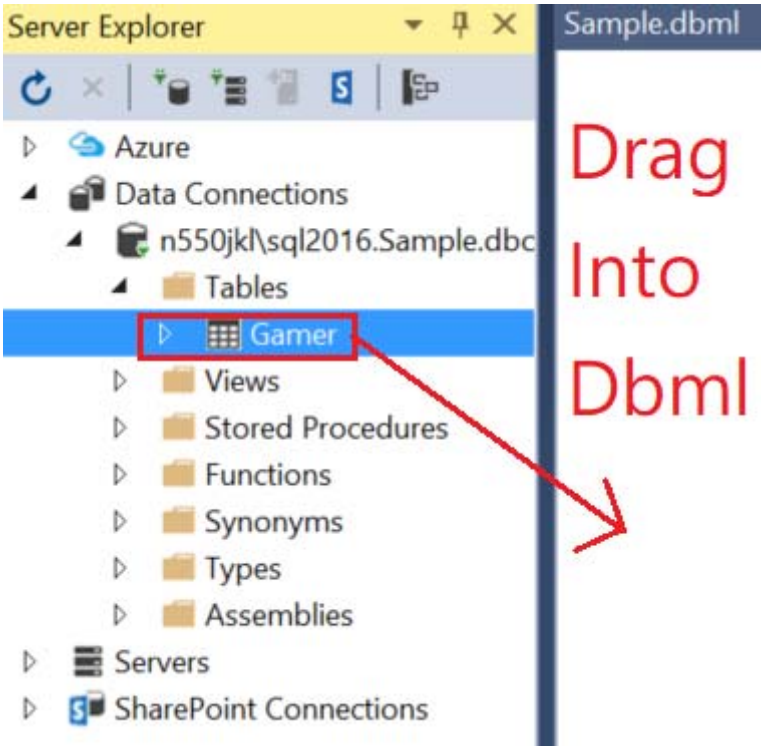
ProjectName --> Right Click --> Add --> New Item...
--> Linq to SQL classes -->
Name : **Sample.dbml**

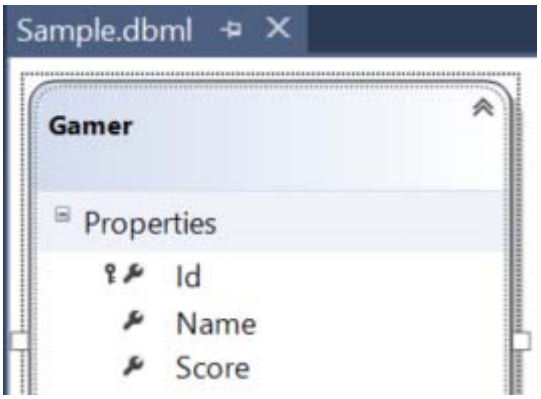
I name it as "Sample.dbml",
because I know this is for connection to "Sample" Database.

-->
Drag Table from Server Explorer into DBML

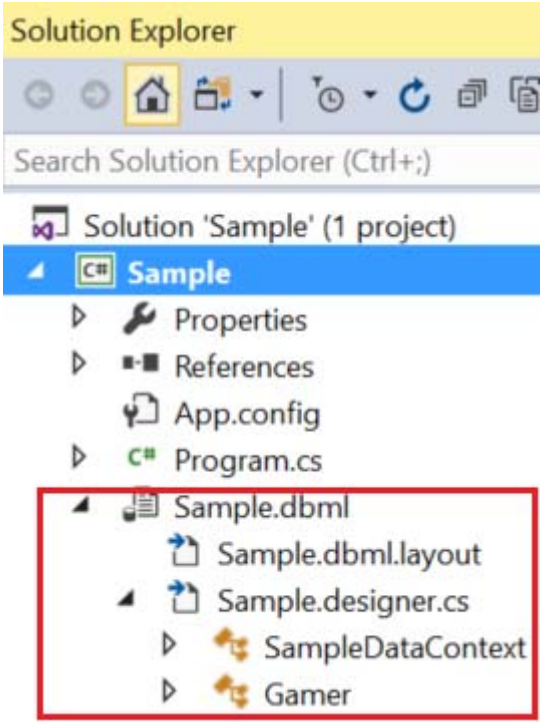








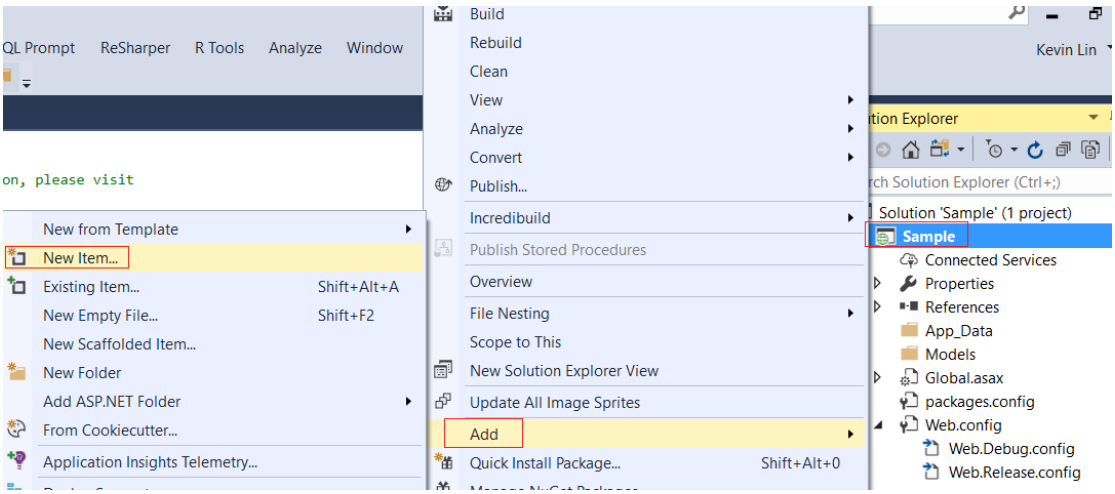
Save the dbml, it will generate the following files.
The DataContext context is the entry point to the database.

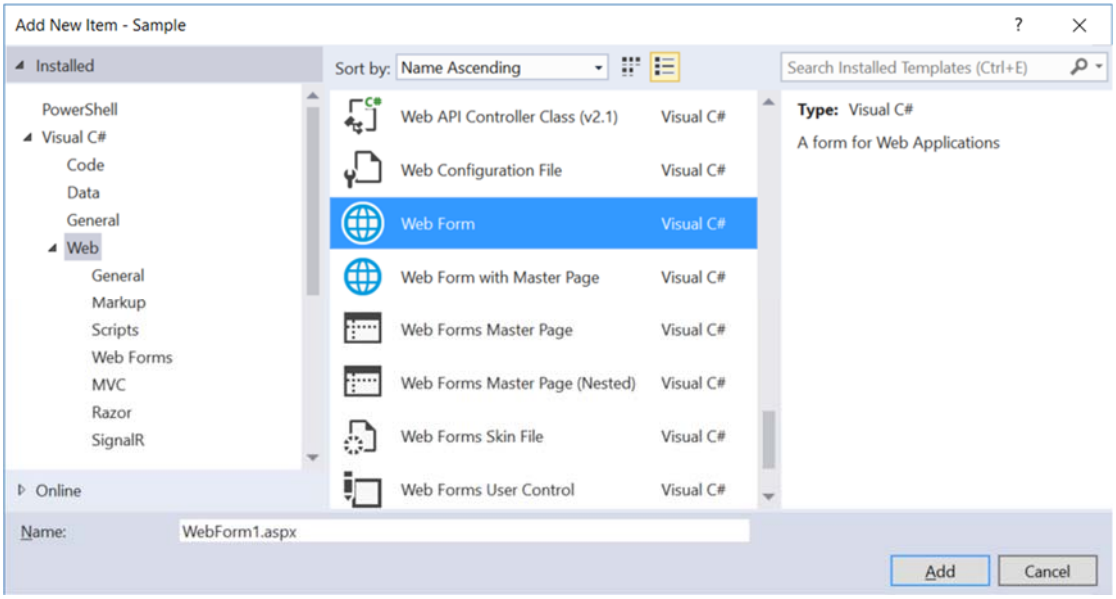


3.3. WebForm1.aspx

3.3.1. WebForm1.aspx

ProjectName --> Right Click --> Add --> New Item...
-->
WebForm
Name :
WebForm1.aspx






```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Sample.WebForm1" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                <tr>
                    <td>
                        <b>Id</b>
                    </td>
                    <td>
                        <asp:Label ID="lblId" runat="server"></asp:Label>
                    </td>
                </tr>
                <tr>
                    <td>
                        <b>Name</b>
                    </td>
                    <td>
                        <asp:Label ID="lblName" runat="server"></asp:Label>
                    </td>
                </tr>
                <tr>
                    <td>
                        <b>Score</b>
                    </td>
                    <td>
                        <asp:Label ID="lblScore" runat="server"></asp:Label>
                    </td>
                </tr>
            </table>
            <br />
            <asp:Button ID="btnAdd1000KeepCurrentValues" runat="server"
```

```
                Text="Score+1000KeepCurrentValues"
                OnClick="btnAdd1000KeepCurrentValues_Click" />
            <asp:Button ID="btnAdd1000KeepChanges" runat="server"
                Text="Score+1000KeepChanges"
                OnClick="btnAdd1000KeepChanges_Click" />
            <asp:Button ID="btnAdd1000OverwriteCurrentValues" runat="server"
                Text="Score+1000OverwriteCurrentValues"
                OnClick="btnAdd1000OverwriteCurrentValues_Click" />
            <asp:Button ID="btnDeduct500" runat="server" Text="Score-500"
                OnClick="btnDeduct500_Click" />
        </div>
    </form>
</body>
</html>
```

3.3.2. WebForm1.aspx.cs

```
using System;
using System.Data.Linq;
using System.Linq;
using System.Threading;
namespace Sample
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            //if (!IsPostBack)  means first time to load this page.
            if (!IsPostBack)
            {
                getGamerData();
            }
        }
        private void getGamerData()
        {
            using (SampleDataContext dbContext = new SampleDataContext())
            {
                Gamer gamer = dbContext.Gamers.First(g => g.Id == 1);
                lblId.Text = gamer.Id.ToString();
                lblName.Text = gamer.Name;
                lblScore.Text = gamer.Score.ToString();
            }
        }
        //1. ====================================
        //KeepCurrentValues
        //Wait N for N millisecond, then update
        //dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
        protected void btnAdd1000KeepCurrentValues_Click(object sender, EventArgs e)
        {
            // 2.1.
            // KeepCurrentValues
            // E.g.
            // //dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
            // KeepCurrentValues means keep all the new values from the current user.
```

```
            // dbContextA is used by current user
            // and tries to update the Column1.
            // In the mean time, dbContextB is used by other user
            // and tries to update the Column1, and Column2.
            // The new value of Column1 from dbContextA will be saved into Database.
            // The old value of Column2 from dbContextA will be saved into Database.
            using (SampleDataContext dbContext = new SampleDataContext())
            {
                try
                {
                    ScoreAdd1000(dbContext);
                }
                catch (ChangeConflictException ex)
                {
                    dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
                    DisplayChangeConflict(dbContext);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                }
            }
        }
        private void DisplayChangeConflict(SampleDataContext dbContext)
        {
            // 3.
            // When ChangeConflictException happens,
            // we can access the following values.
            // 3.1.
            // //memberChangeConflict.Member.Name
            // This will show you the property Name which contains change conflict data.
            // The property name is normally same as Column Name from database.
            // 3.2.
            // //memberChangeConflict.CurrentValue
            // The will show you the current value of the property which contains change conflict data.
            // This is the new value which updated from the current user.
            // 3.3.
            // //memberChangeConflict.OriginalValue
            // The will show you the original value of the property which contains change conflict data.
            // This is the old value which has not been updated from the current user yet.
            // 3.4.
            // //memberChangeConflict.DatabaseValue
            // The will show you the current value of the corresponding column in the database table.
            foreach (ObjectChangeConflict objectChangeConflict
                in dbContext.ChangeConflicts)
            {
                foreach (MemberChangeConflict memberChangeConflict
                    in objectChangeConflict.MemberConflicts)
                {
                    Response.Write($"memberChangeConflict.Member.Name=={memberChangeConflict.Member.Name}
<br/>");
                    Response.Write($"memberChangeConflict.CurrentValue=={memberChangeConflict.CurrentValu
e}<br/>");
                    Response.Write($"memberChangeConflict.OriginalValue=={memberChangeConflict.OriginalVa
lue}<br/>");
                    Response.Write($"memberChangeConflict.DatabaseValue=={memberChangeConflict.DatabaseVa
lue}<br/>");
                }
```

```
        }
        dbContext.SubmitChanges();
        getGamerData();
    }


private void ScoreAdd1000(SampleDataContext dbContext)
    {
        Gamer gamer = dbContext.Gamers.First(g => g.Id == 1);
        gamer.Score += 1000;
        Thread.Sleep(2000); // sleep for N millisecond.
        dbContext.SubmitChanges();
        getGamerData();
    }
    //2. ====================================
    //KeepChanges
    //Wait N for N millisecond, then update
    //dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepChanges);
    protected void btnAdd1000KeepChanges_Click(object sender, EventArgs e)
    {
        //2.2.
        // KeepChanges
        // E.g.
        // //dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepChanges);
        // KeepChanges means keep all the changes from all the users.
        // If any conflict, then keep the new values from the current user.
        // dbContextA is used by current user
        // and tries to update the Column1.
        // In the mean time, dbContextB is used by other user
        // and tries to update the Column1, and Column2.
        // The new value of Column1 from dbContextA will be saved into Database.
        // The new value of Column2 from dbContextB will be saved into Database.
        using (SampleDataContext dbContext = new SampleDataContext())
        {
            try
            {
                ScoreAdd1000(dbContext);
            }
            catch (ChangeConflictException ex)
            {
                dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepChanges);
                DisplayChangeConflict(dbContext);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }


//3. ====================================
    //OverwriteCurrentValues
    //Wait N for N millisecond, then update
    //dbContext.ChangeConflicts.ResolveAll(RefreshMode.OverwriteCurrentValues);
    protected void btnAdd1000OverwriteCurrentValues_Click(object sender, EventArgs e)
    {
        //2.3.
```

```
            // OverwriteCurrentValues
            // E.g.
            // //dbContext.ChangeConflicts.ResolveAll(RefreshMode.OverwriteCurrentValues);
            // OverwriteCurrentValues means discard all the changes from the current user.
            // dbContextA is used by current user
            // and tries to update the Column1.
            // In the mean time, dbContextB is used by other user
            // and tries to update the Column1, and Column2.
            // The new value of Column1 from dbContextB will be saved into Database.
            // The new value of Column2 from dbContextB will be saved into Database.
            // Because OverwriteCurrentValues means discard all the changes from dbContextA.
            using (SampleDataContext dbContext = new SampleDataContext())
            {
                try
                {
                    ScoreAdd1000(dbContext);
                }
                catch (ChangeConflictException ex)
                {
                    dbContext.ChangeConflicts.ResolveAll(RefreshMode.OverwriteCurrentValues);
                    DisplayChangeConflict(dbContext);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                }
            }
        }
        // 4. Update straight away. ====================================
        protected void btnDeduct500_Click(object sender, EventArgs e)
        {
            using (SampleDataContext dbContext = new SampleDataContext())
            {
                Gamer gamer = dbContext.Gamers.First(g => g.Id == 1);
                gamer.Name += "X";
                gamer.Score -= 500;
                dbContext.SubmitChanges();
                getGamerData();
            }
        }
    }
}
```

3.4. Run WebForm1.aspx

Id 1

Name Name1 ABC

Score 5000

Score+1000KeepCurrentValues | Score+1000KeepChanges

Score+1000OverwriteCurrentValues | Score-500

Id 1
Name Name1 ABC
Score 6000

Score+1000KeepCurrentValues | Score+1000KeepChanges
Score+1000OverwriteCurrentValues | Score-500

Id 1
Name Name1 ABC
Score 7000

Score+1000KeepCurrentValues | Score+1000KeepChanges
Score+1000OverwriteCurrentValues | Score-500

Id 1
Name Name1 ABC
Score 8000

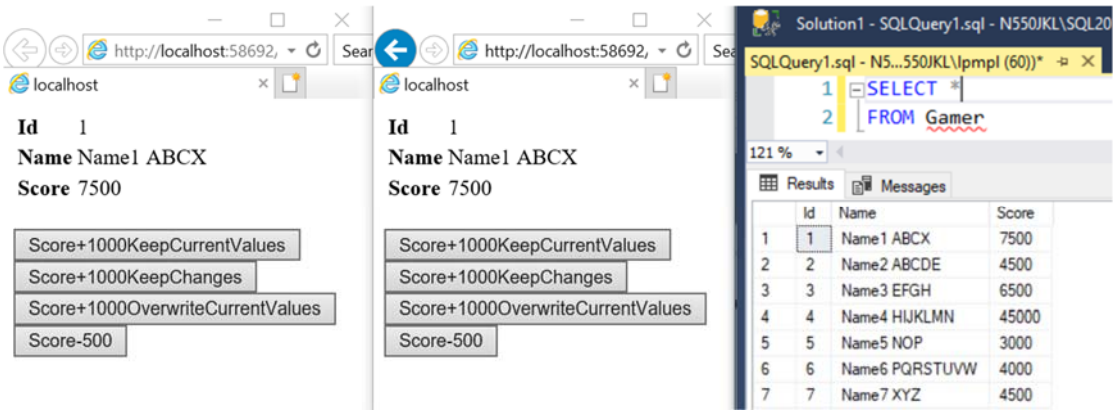
Score+1000KeepCurrentValues | Score+1000KeepChanges
Score+1000OverwriteCurrentValues | Score-500

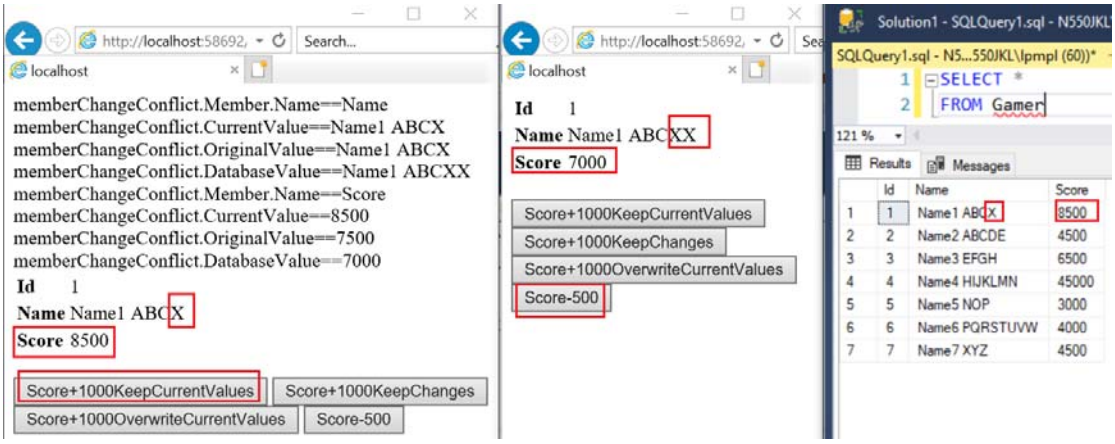
Id 1
Name Name1 ABCX
Score 7500

Score+1000KeepCurrentValues | Score+1000KeepChanges
Score+1000OverwriteCurrentValues | Score-500

3.5. Run WebForm1.aspx with update conflicts

3.5.1. WebForm1.aspx with KeepCurrentValues

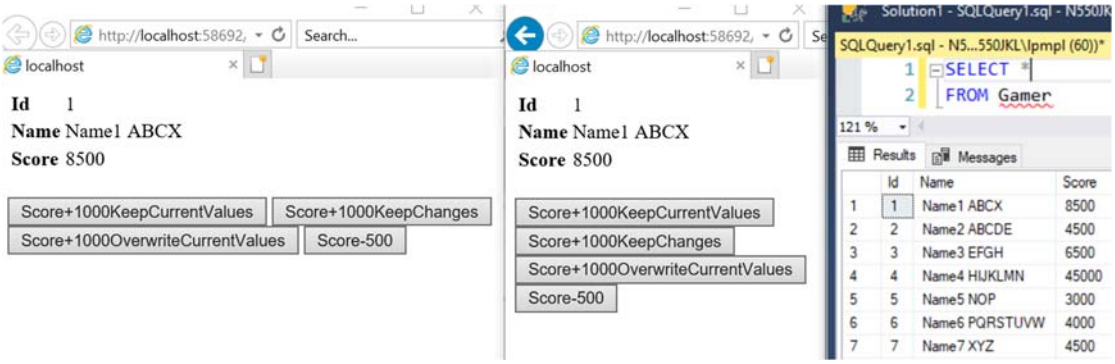


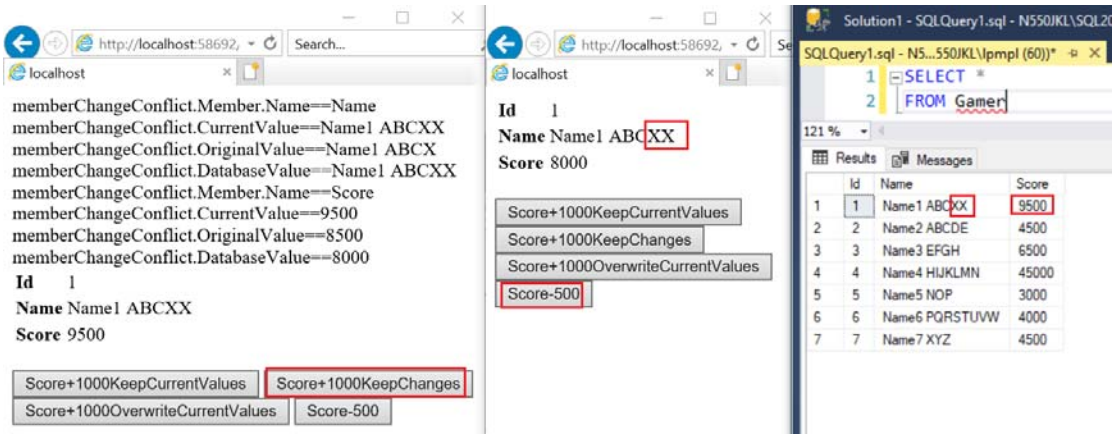


2.
RefreshMode enum in optimistic concurrency control
has 3 different options to handle **ChangeConflictException**.
KeepCurrentValues V.S. **KeepChanges** V.S. **OverwriteCurrentValues**

2.1.
KeepCurrentValues
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
<u>KeepCurrentValues means keep all the new values from the current user.</u>
dbContextA is used by current user
and tries to update the **Column1**.
In the meantime, **dbContextB** is used by **another user**
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextA** will be saved into Database.
The **old** value of **Column2** from **dbContextA** will be saved into Database.

3.5.2. WebForm1.aspx with KeepChanges

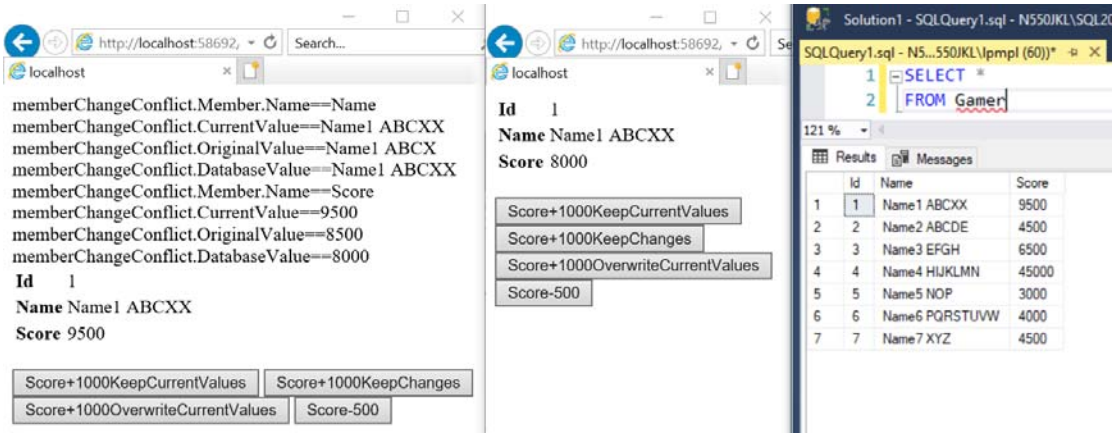


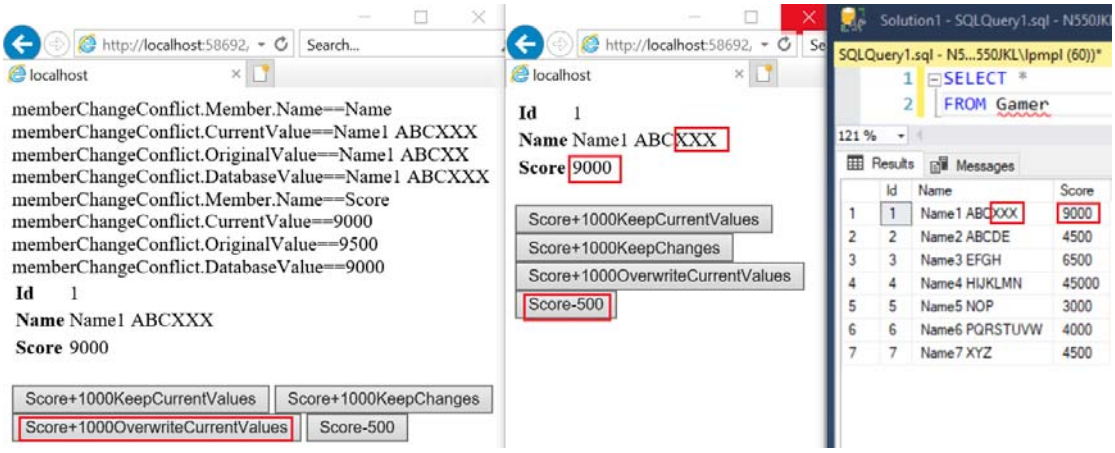


2.
RefreshMode enum in optimistic concurrency control
has 3 different options to handle **ChangeConflictException**.
KeepCurrentValues V.S. **KeepChanges** V.S. **OverwriteCurrentValues**

2.2.
KeepChanges
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.KeepChanges);
<u>KeepChanges means keep all the changes from all the users.</u>
<u>If any conflict, then keeps the new values from the current user.</u>
dbContextA is used by the **current user**
and tries to update the **Column1**.
In the meantime, **dbContextB** is used by **another user**
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextA** will be saved into Database.
The **new** value of **Column2** from **dbContextB** will be saved into Database.

3.5.3. WebForm1.aspx with OverwriteCurrentValues





2.
RefreshMode enum in optimistic concurrency control
has 3 different options to handle **ChangeConflictException**.
KeepCurrentValues V.S. **KeepChanges** V.S. **OverwriteCurrentValues**

2.3.
OverwriteCurrentValues
E.g.
//dbContext.ChangeConflicts.ResolveAll(RefreshMode.OverwriteCurrentValues);
<u>OverwriteCurrentValues means discard all the changes from the current user.</u>
dbContextA is used by current user
and tries to update the **Column1**.
In the mean time, **dbContextB** is used by other user
and tries to update the **Column1**, and **Column2**.
The **new** value of **Column1** from **dbContextB** will be saved into Database.
The **new** value of **Column2** from **dbContextB** will be saved into Database.
Because OverwriteCurrentValues means discard all the changes from **dbContextA**.

4. UpdateCheck Property

4.1. TSQL (UpdateCheck Property)

Run the following Tsql to Sample Database again to clean up the data.

--Create a Sample DataBase and Run the following TSQL

```
--1 -------------------------------------------------------
--Drop Table if it exists.
--IF OBJECT_ID('Gamer') IS NOT NULL
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE      TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
    (
      Id INT PRIMARY KEY
             IDENTITY ,
     Name NVARCHAR(50) ,
      Score INT ,
    );
GO -- Run the previous command and begins new batch
--2 -------------------------------------------------------
INSERT  INTO Gamer
VALUES  ( 'Name1 ABC', 5000 );
GO -- Run the previous command and begins new batch
```

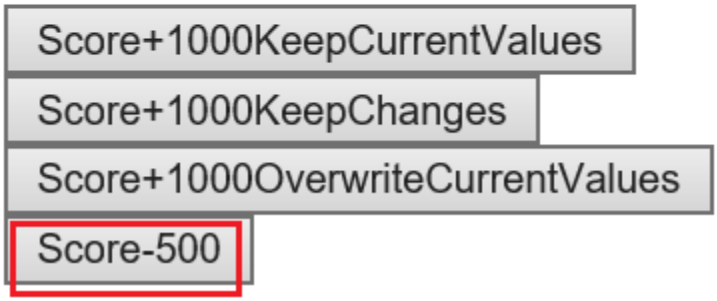
4.2. WebForm1.aspx (UpdateCheck Property)

Turn on the Sql Profiler and run WebForm1.aspx

Id 1

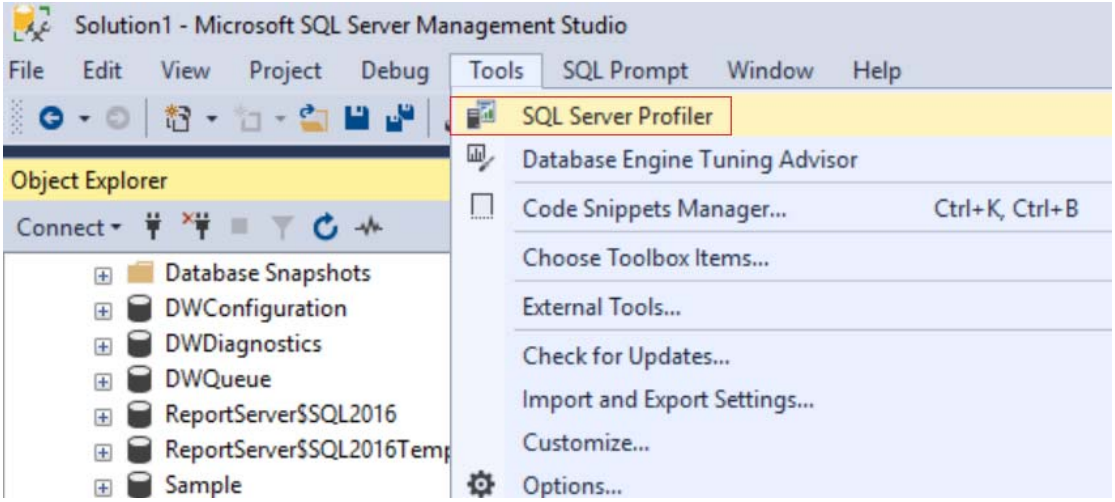
Name Name1 ABC

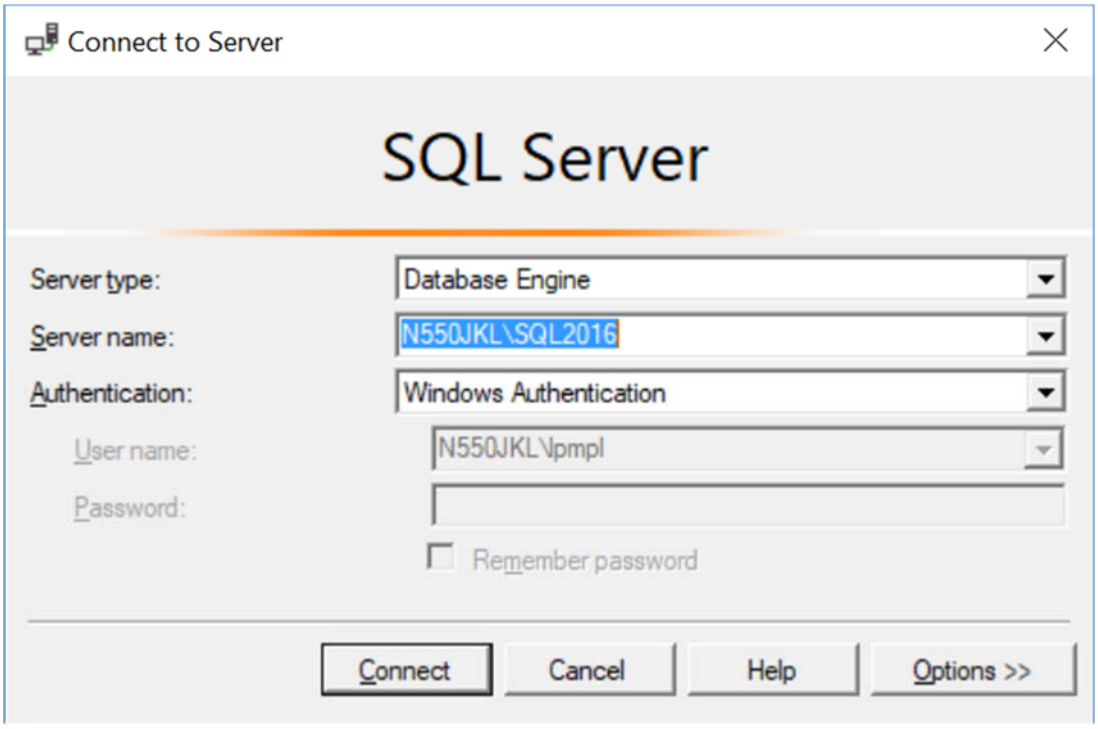
Score 5000

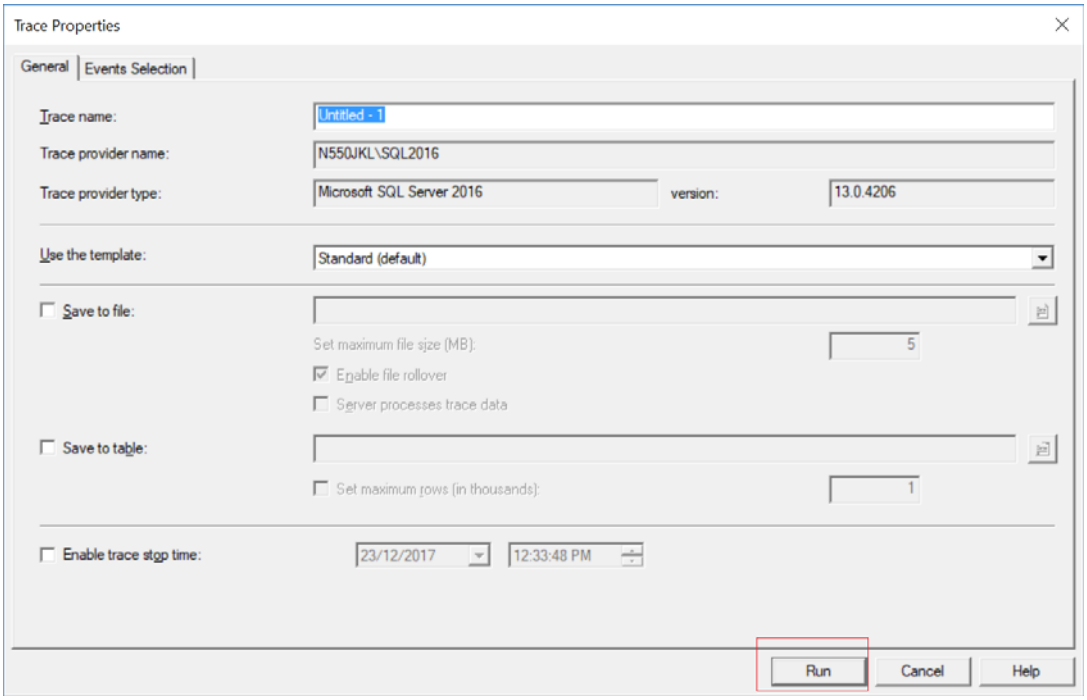


4.3. SQL Profiler (UpdateCheck Property)

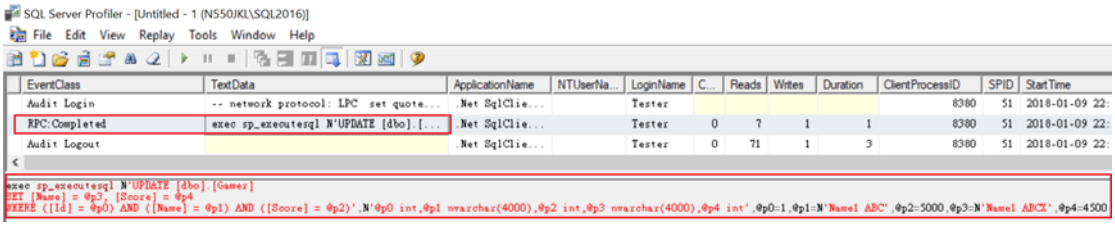
Tools --> SQL Server Profiler


Solution1 - Microsoft SQL Server Management Studio
File
Edit
View
Project
Debug
Tools
SQL Prompt
Window
Help
SQL Server Profiler
Database Engine Tuning Advisor
Code Snippets Manager...
Ctrl+K, Ctrl+B
Choose Toolbox Items...
External Tools...
Check for Updates...
Import and Export Settings...
Customize...
Options...
Object Explorer
Connect
Database Snapshots
DWConfiguration
DWDiagnostics
DWQueue
ReportServer$SQL2016
ReportServer$SQL2016Temp
Sample


Connect to Server
SQL Server
Server type:
Database Engine
Server name:
N550JKL\SQL2016
Authentication:
Windows Authentication
User name:
N550JKL\pmpl
Password:
Remember password
Connect
Cancel
Help
Options >>


Trace Properties
General
Events Selection
Trace name:
Untitled - 1
Trace provider name:
N550JKL\SQL2016
Trace provider type:
Microsoft SQL Server 2016
version:
13.0.4206
Use the template:
Standard (default)
Save to file:
Set maximum file size (MB):
5
Enable file rollover
Server processes trace data
Save to table:
Set maximum rows (in thousands):
1
Enable trace stop time:
23/12/2017
12:33:48 PM
Run
Cancel
Help

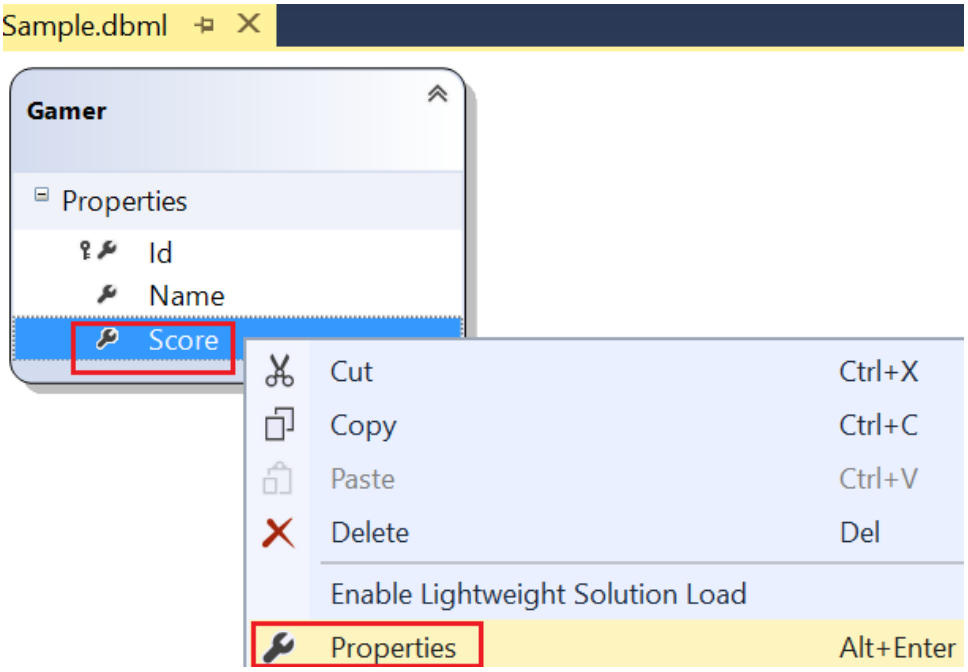
Now, go back to VS2017, and run WebForm1.aspx again
You will see Linq to SQL provider convert Linq to TSQL.

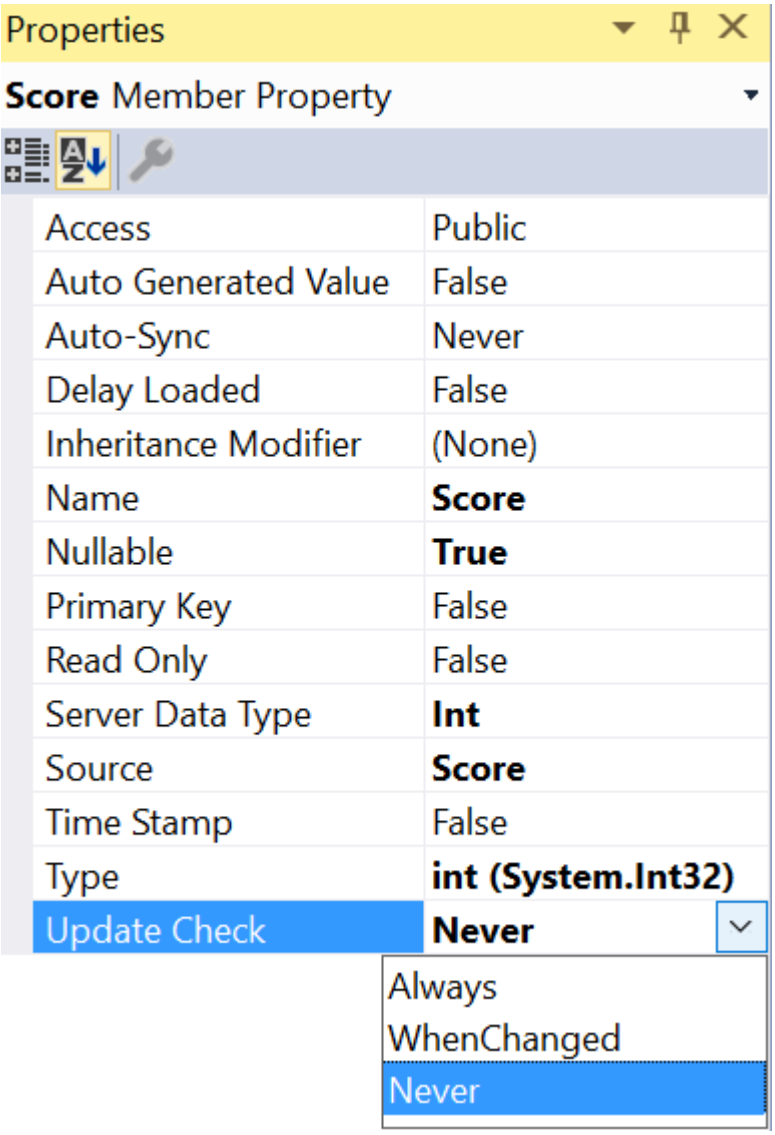



```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p3, [Score] = @p4
WHERE
    ([Id] = @p0) AND
       ([Name] = @p1) AND
       ([Score] = @p2)',
N'@p0 int,
@p1 nvarchar(4000),
@p2 int,@p3 nvarchar(4000),
@p4 int',
@p0=1,@p1=N'Name1 ABC',@p2=5000,@p3=N'Name1 ABCX',@p4=4500
```

We can see all the columns is in WHERE clause,
which means by default all the columns will be used to detect concurrency conflicts.

4.4. DBML (UpdateCheck Property)



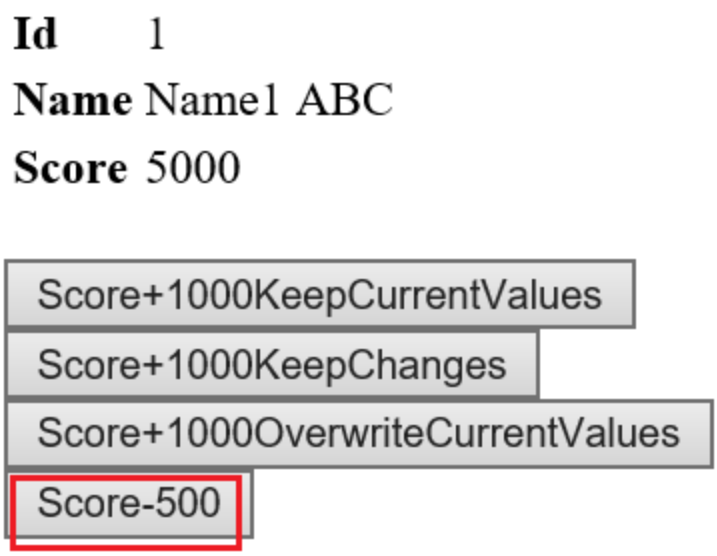


In Dbml -->
Gamer Class --> **Score** --> Right click --> Properties
--> Update Check --> Now, choose "**Never**"

4.
UpdateCheck property
UpdateCheck property can be set to
one of the 3 values of the UpdateCheck enum
which is in System.Data.Linq.Mapping namespace.
4.1.
Always
By default, "Always" use this column for conflict detection
4.2.
Never
"Never" use this column for conflict detection
4.3.
WhenChanged
Use this column only when the member has been changed by the application

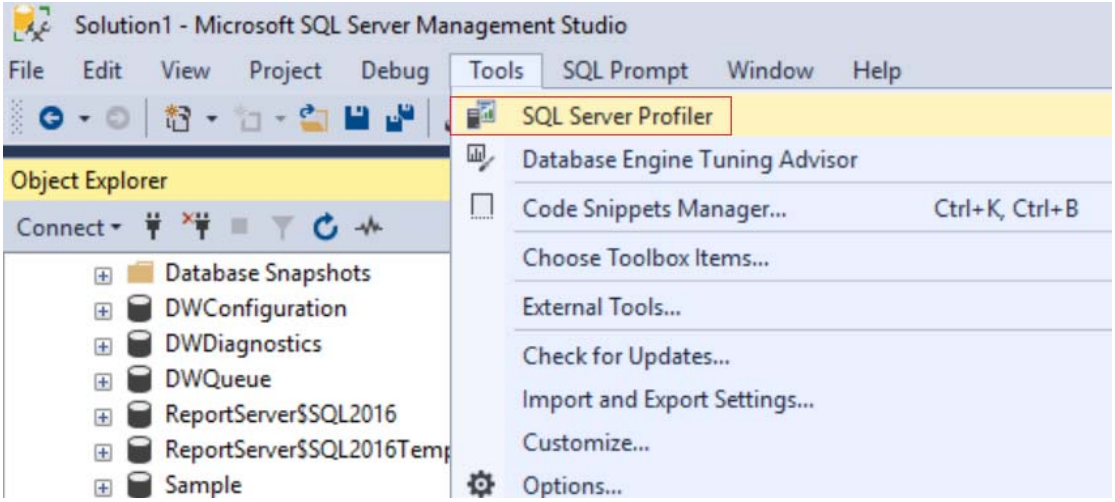
4.5. WebForm1.aspx (UpdateCheck Property)

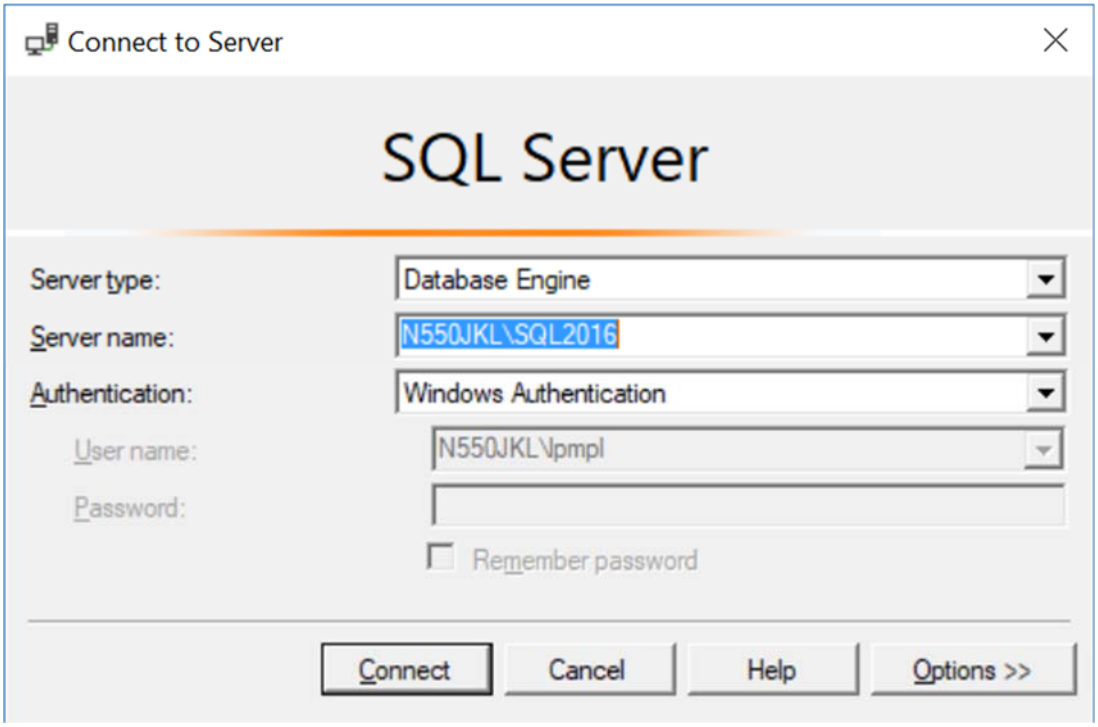
Turn on the Sql Profiler and run WebForm1.aspx

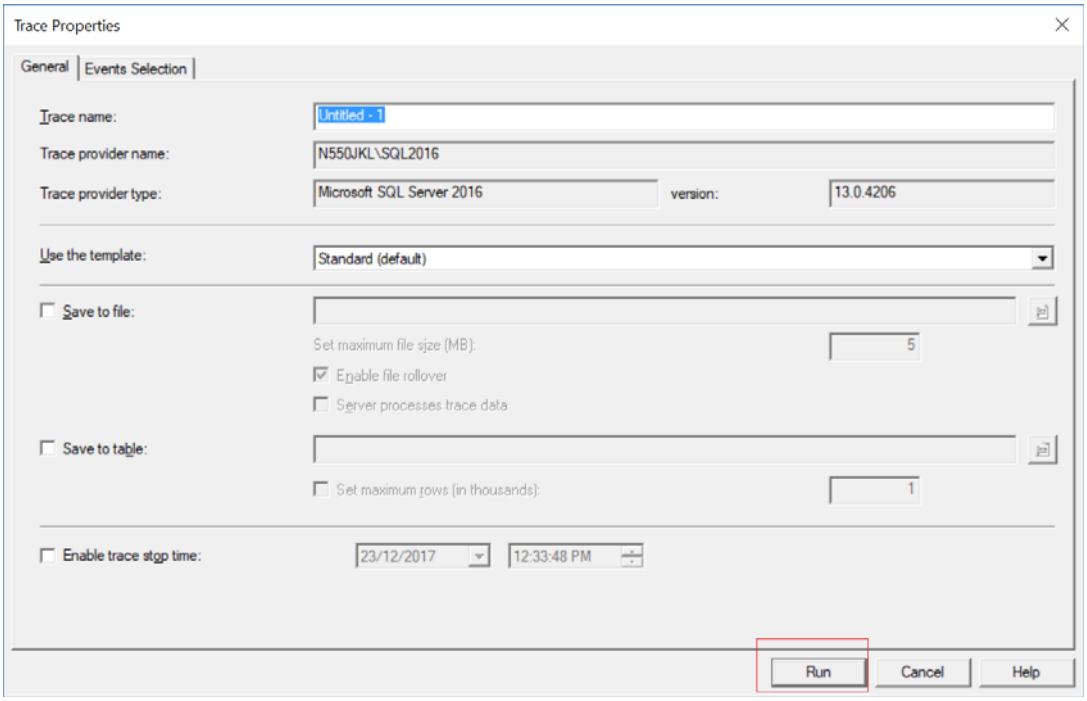


4.6. SQL Profiler (UpdateCheck Property)

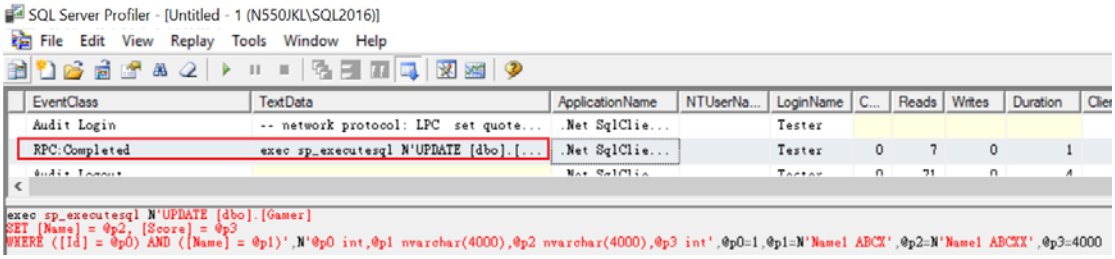
Tools --> SQL Server Profiler







Now, go back to VS2017, and run WebForm1.aspx again
You will see Linq to SQL provider convert Linq to TSQL.



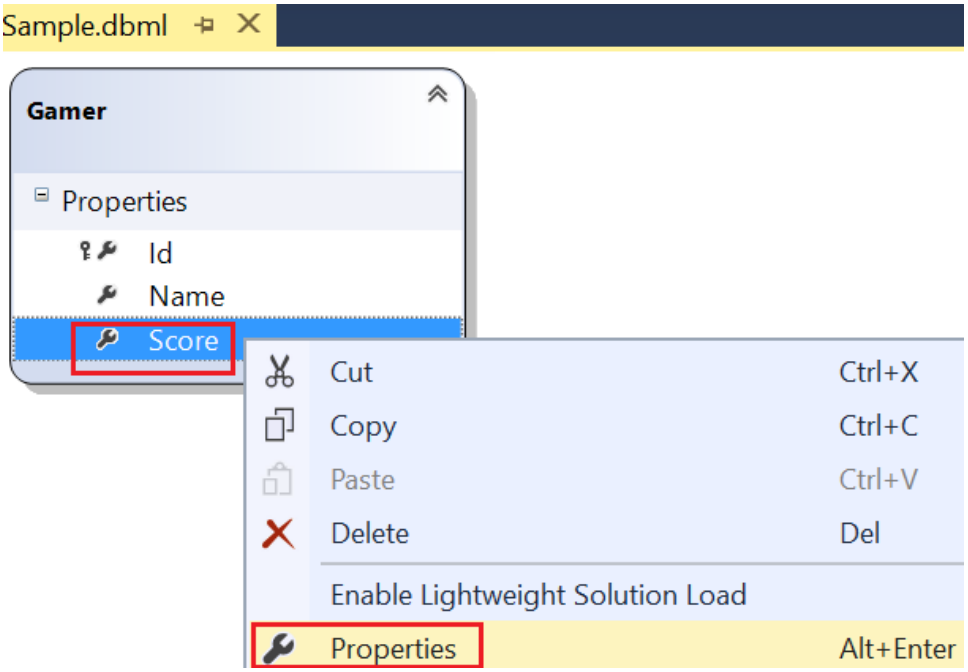

```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p2, [Score] = @p3
WHERE
```

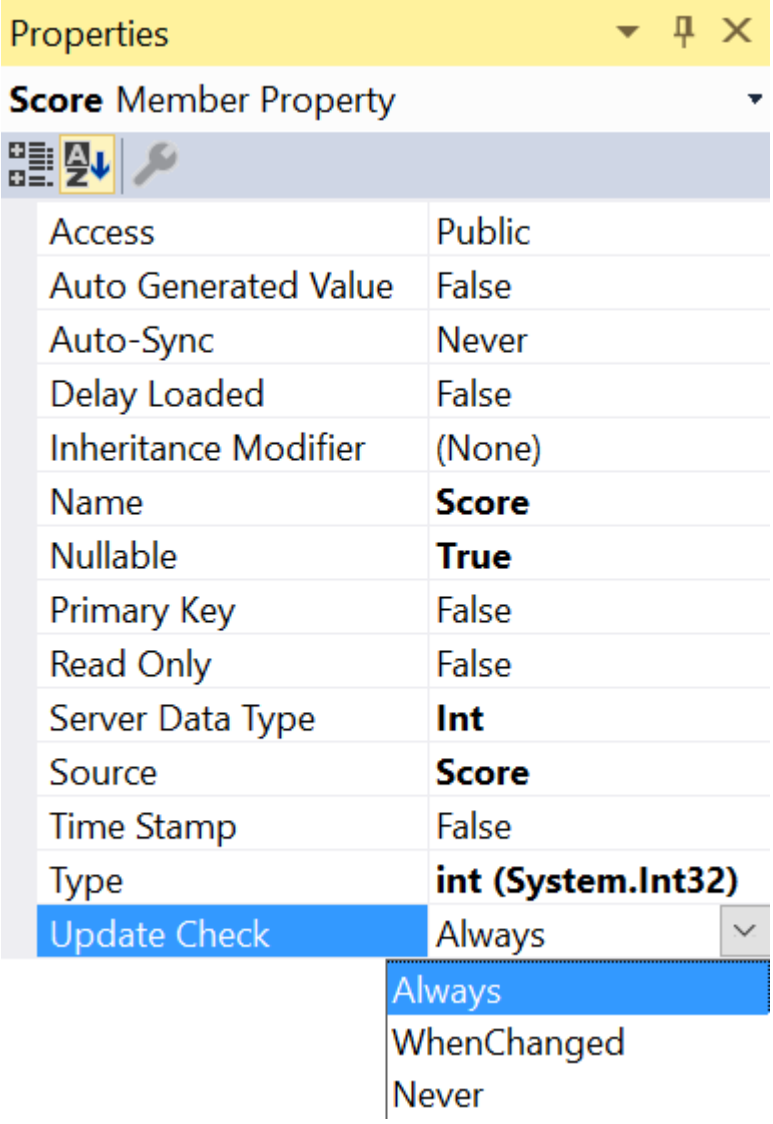
```
    ([Id] = @p0) AND
    ([Name] = @p1)',
N'@p0 int,
@p1 nvarchar(4000),
@p2 nvarchar(4000),
@p3 int',
@p0=1,@p1=N'Name1 ABCX',@p2=N'Name1 ABCXX',@p3=4000
```

By default all the columns will be used to detect concurrency conflicts.
However, we set updateCheck property of "Score" is "Never".
We can see "Score" column is not in Where clause any more,
but we can still see rest of columns is in WHERE clause.

4.7. DBML (UpdateCheck Property)

In Dbml -->
Gamer Class --> **Score** --> Right click --> Properties
--> Update Check --> Now, choose "**Always**"





5. Rowversion and Timestamp

5.1. TSQL (Rowversion and Timestamp)

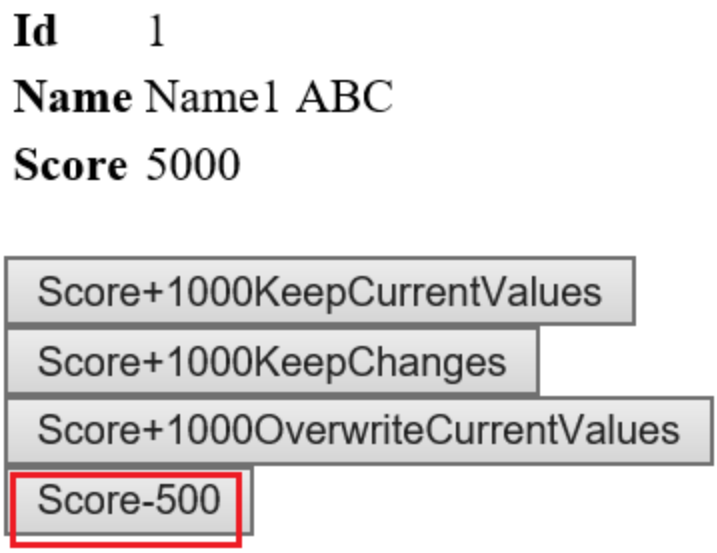
Run the following Tsql to Sample Database again to clean up the data.

```
--Create a Sample DataBase and Run the following TSQL
--1 -------------------------------------------------------
--Drop Table if it exists.
--IF OBJECT_ID('Gamer') IS NOT NULL
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE      TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
```

```sql
CREATE TABLE Gamer
    (
      Id INT PRIMARY KEY
            IDENTITY ,
     Name NVARCHAR(50) ,
      Score INT ,
    );
GO -- Run the previous command and begins new batch
--2 -------------------------------------------------------
INSERT  INTO Gamer
VALUES  ( 'Name1 ABC', 5000 );
GO -- Run the previous command and begins new batch
```

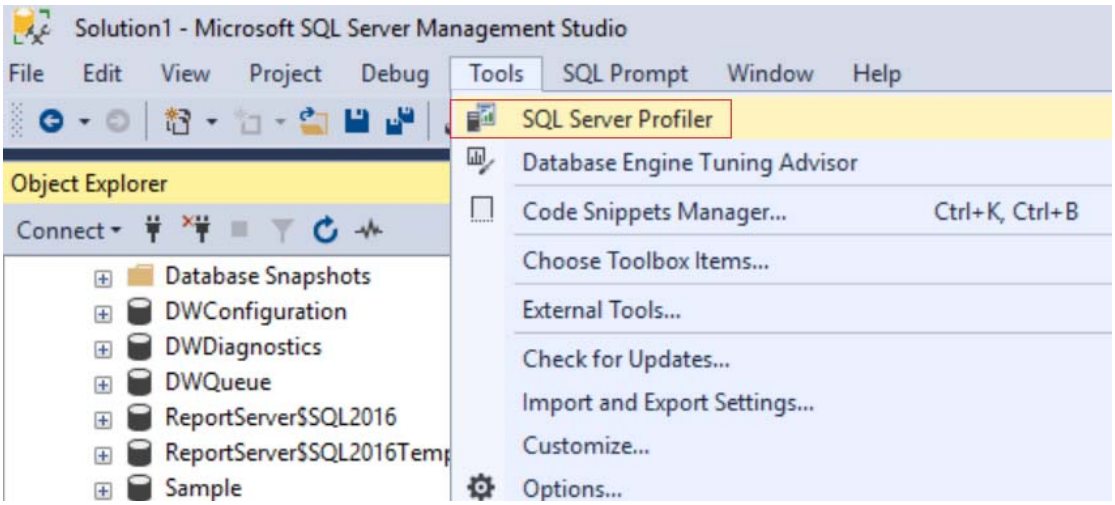
5.2. WebForm1.aspx (Rowversion and Timestamp)

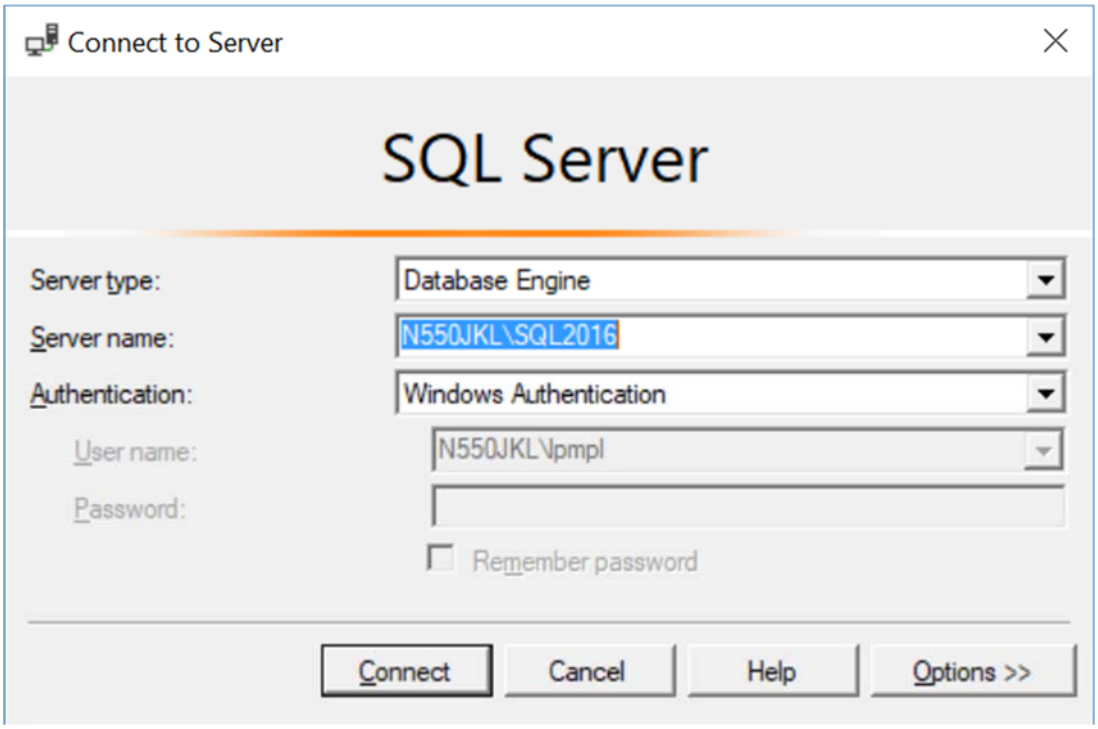
Turn on the Sql Profiler and run WebForm1.aspx

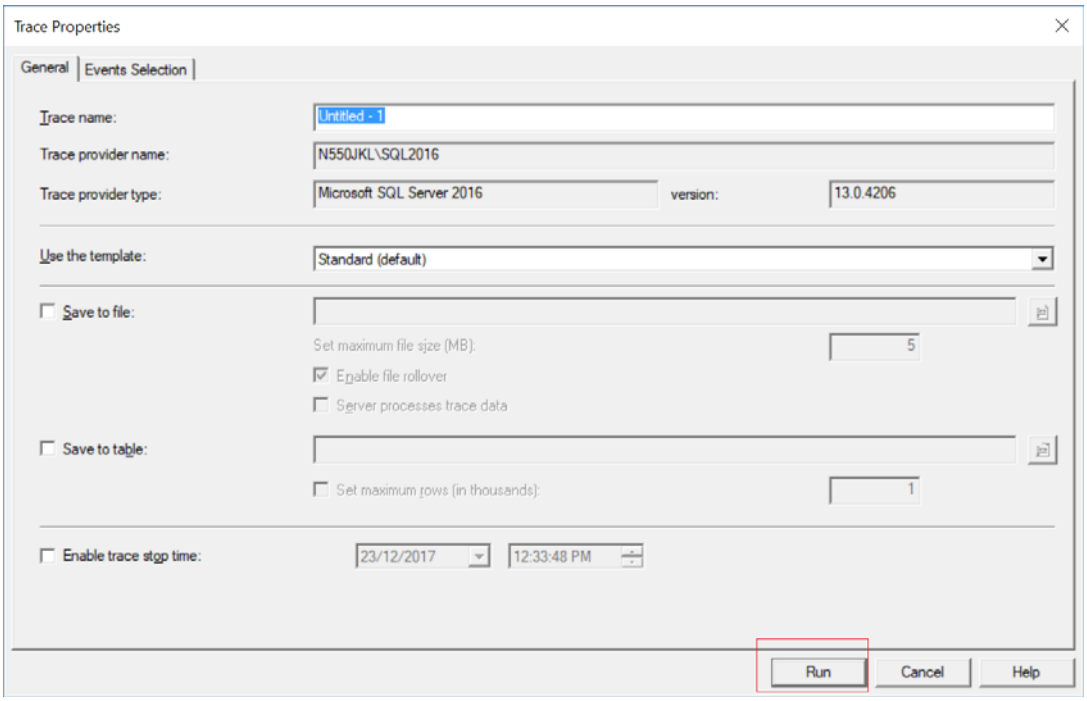


5.3. SQL Profiler (Rowversion and Timestamp)

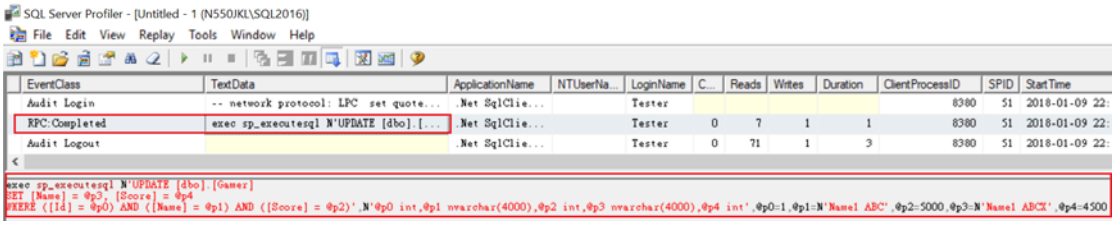
Tools --> SQL Server Profiler







Now, go back to VS2017, and run WebForm1.aspx again
You will see Linq to SQL provider convert Linq to TSQL.




```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p3, [Score] = @p4
WHERE
```

```
    ([Id] = @p0) AND
       ([Name] = @p1) AND
       ([Score] = @p2)',
N'@p0 int,
@p1 nvarchar(4000),
@p2 int,@p3 nvarchar(4000),
@p4 int',
@p0=1,@p1=N'Name1 ABC',@p2=5000,@p3=N'Name1 ABCX',@p4=4500
```

We can see all the columns is in WHERE clause,
which means by default all the columns will be used to detect concurrency conflicts.

It is fine if the table only has a few columns.
However, if the table has a lot of columns such as 30 columns,
this will cause performance problem.

Therefore, we need a "**Version**" Column
and the data type can be "**ROWVERSION**" or "**TIMESTAMP**"

5.4. Add Rowversion or Timestamp (Rowversion and Timestamp)

Run the following Tsql to Sample Databsae.
Here, we add another column called "**Version**"
and the data type can be "ROWVERSION" or "Timestamp"
Here, we use "**ROWVERSION**"

```
ALTER TABLE Gamer
ADD [Version] ROWVERSION
```

Now lets see what is going on in Gamer Table

```
SELECT  *
FROM    Gamer;
```

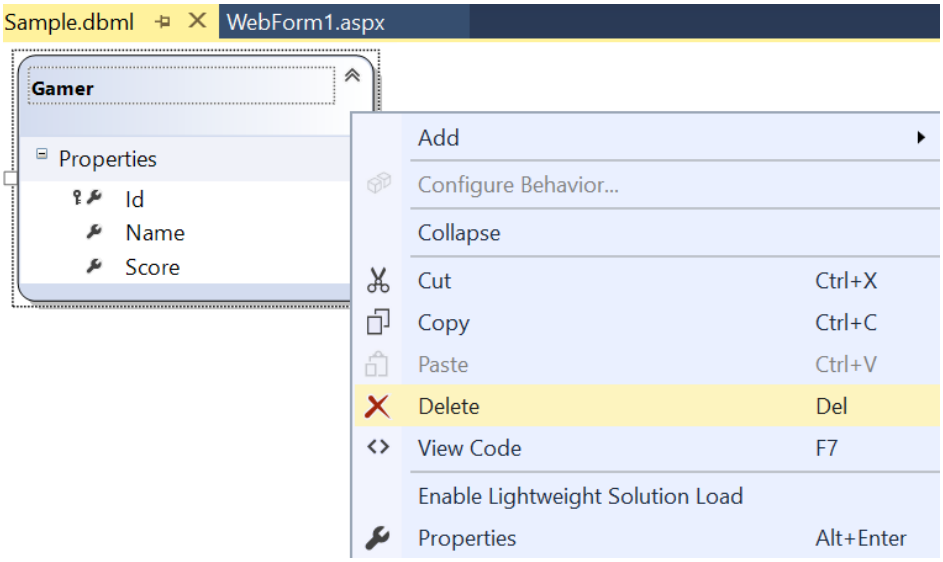
Results | Messages

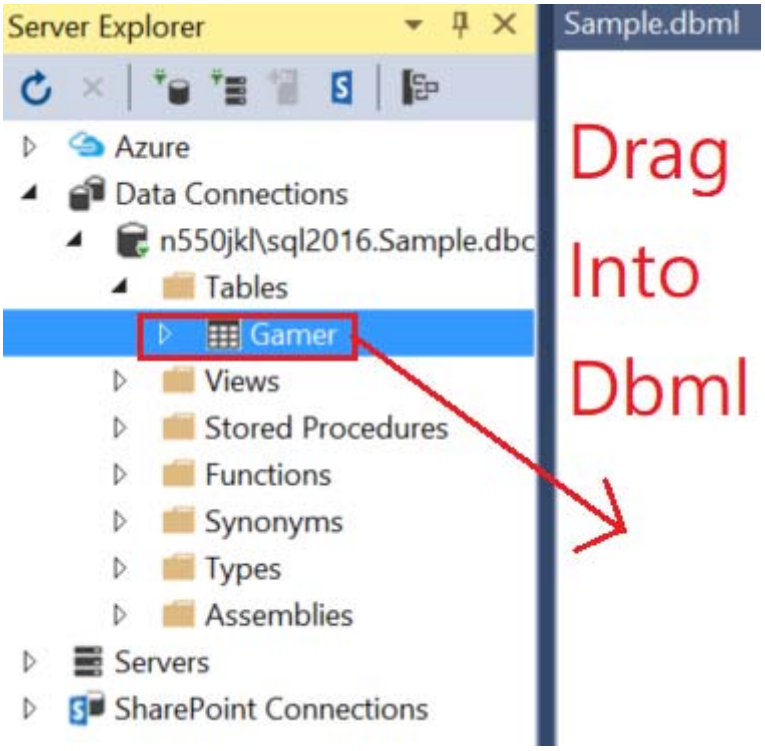
	Id	Name	Score	Version
1	1	Name1 ABCX	4500	0x00000000000007D1

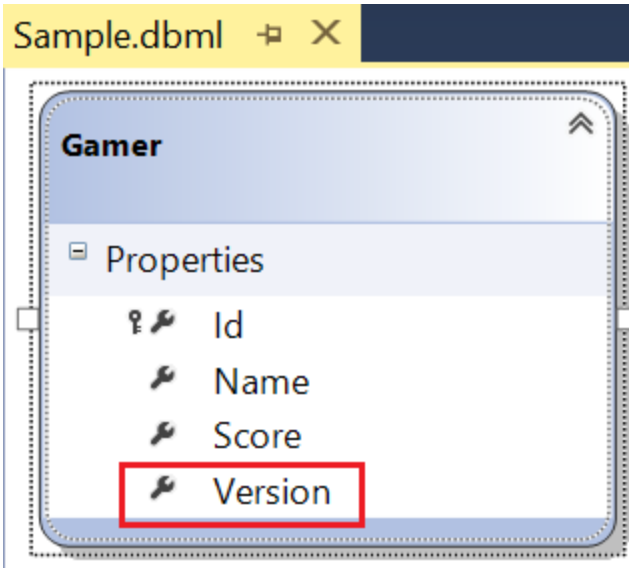
We can see the "**0x00000000000007D1**" is dynamically generated by the database.
The type is of "**Version**" is "**Rowversion**"

5.5. DBML (Rowversion and Timestamp)

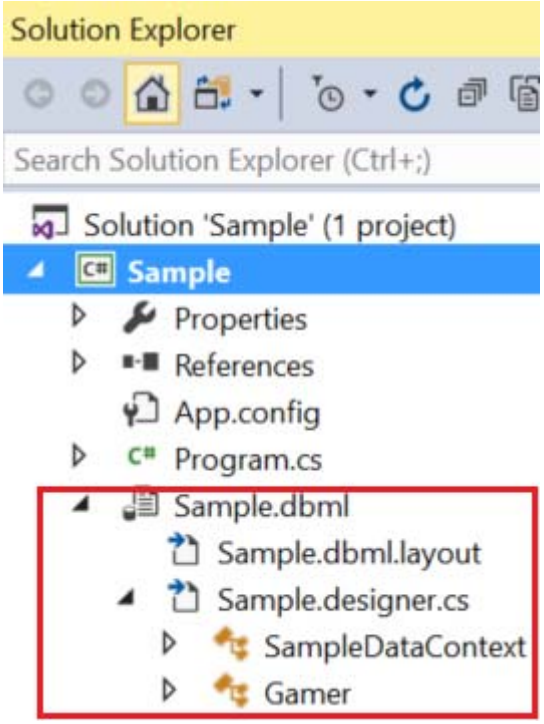
In DBML
Select "Gamer" --> Delete


Sample.dbml
WebForm1.aspx
Gamer
Properties
Id
Name
Score
Add
Configure Behavior...
Collapse
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
View Code F7
Enable Lightweight Solution Load
Properties Alt+Enter


Server Explorer
Sample.dbml
Azure
Data Connections
n550jkl\sql2016.Sample.dbc
Tables
Gamer
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Servers
SharePoint Connections
Drag
Into
Dbml


Sample.dbml
Gamer
Properties
Id
Name
Score
Version

Save the dbml, it will generate the following files.
The DataContext context is the entry point to the database.



In Sample.designer.cs
check the "Version" property in "Gamer" class
Notice that **IsVersion** and **IsDbGenerated** properties are set to **true**.
This means the value of "Version" is dynamically generated by the database.
Normally the data type of corresponding "Version" column
in Database table is "**Rowversion**" or "**TimeStamp**".

```
[global::System.Data.Linq.Mapping.ColumnAttribute(Storage="_Version", AutoSync=AutoSync.Always,
DbType="rowversion NOT NULL", CanBeNull=false, IsDbGenerated=true, IsVersion=true,
UpdateCheck=UpdateCheck.Never)]
public System.Data.Linq.Binary Version
{
        get
...
```

5.6. WebForm1.aspx (Rowversion and Timestamp)

Turn on the Sql Profiler and run WebForm1.aspx

Id 1

Name Name1 ABC

Score 5000

Score+1000KeepCurrentValues

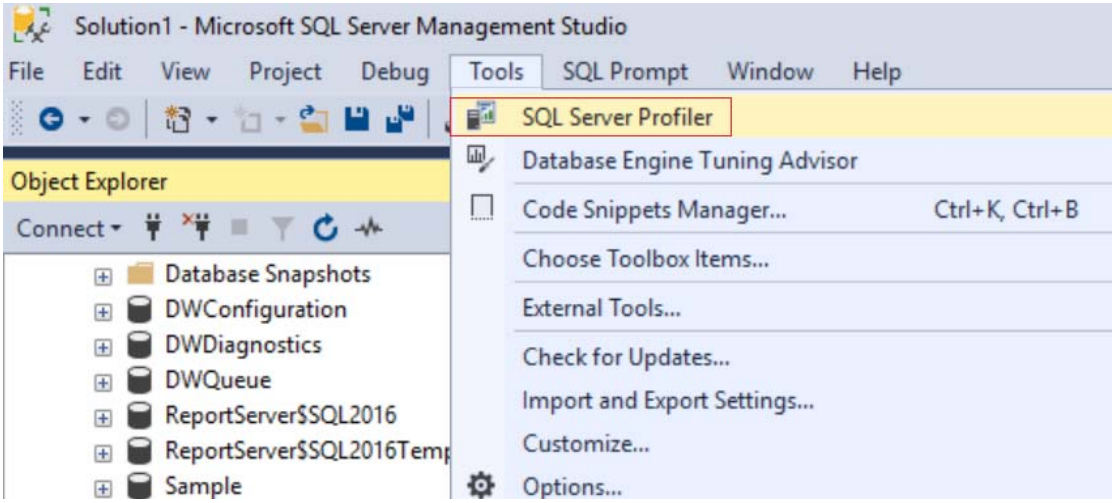
Score+1000KeepChanges

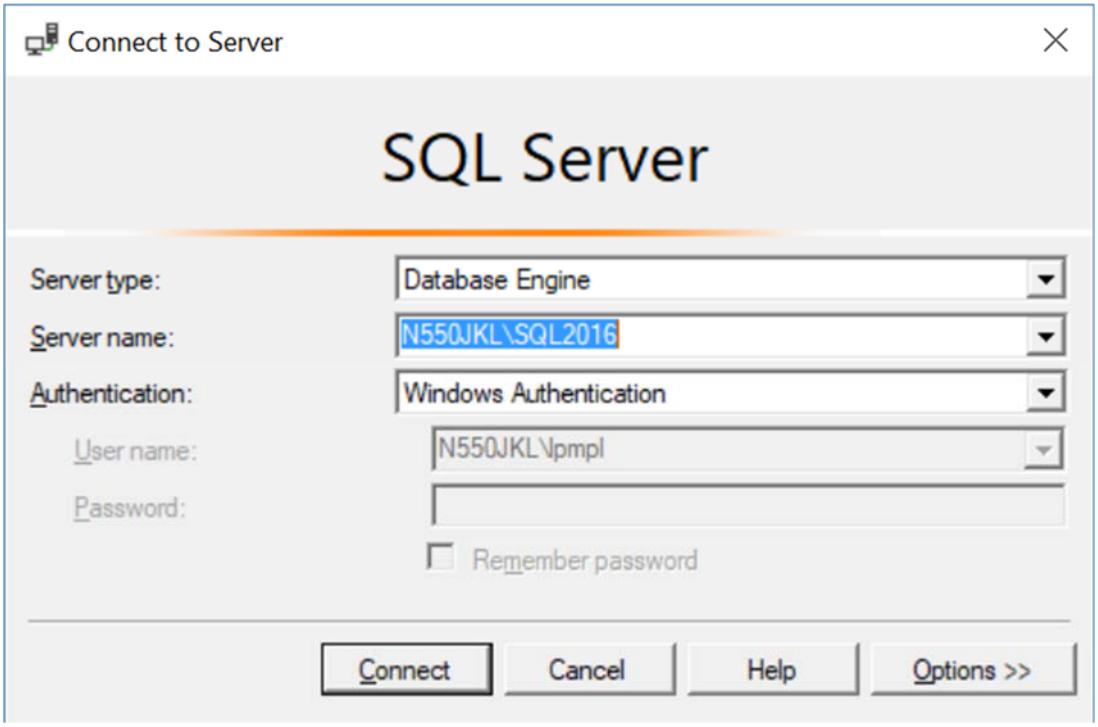
Score+1000OverwriteCurrentValues

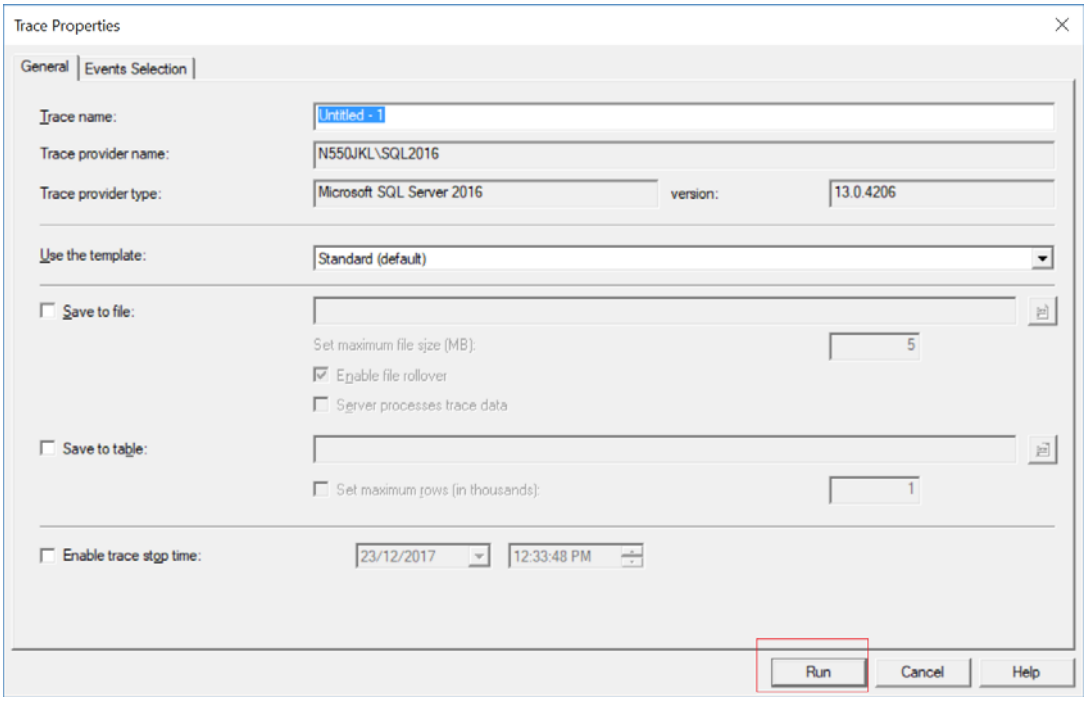
Score-500

5.7. SQL Profiler (Rowversion and Timestamp)

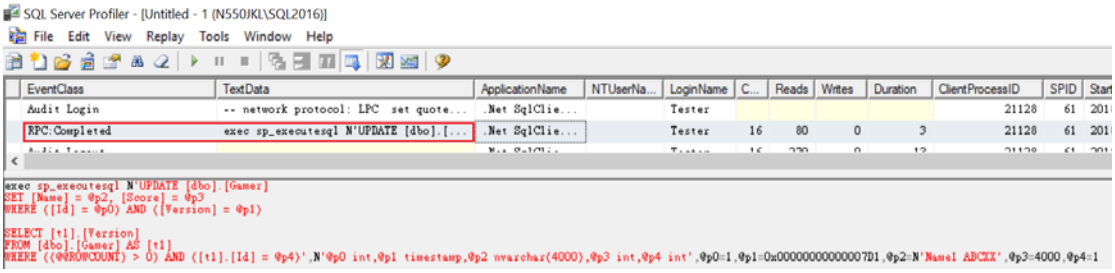
Tools --> SQL Server Profiler







Now, go back to VS2017, and run WebForm1.aspx again
You will see Linq to SQL provider convert Linq to TSQL.



The following code is the Tsql **before** we use "Version" Column with data type "RowVersion"

```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p3, [Score] = @p4
WHERE
    ([Id] = @p0) AND
       ([Name] = @p1) AND
       ([Score] = @p2)',
N'@p0 int,
@p1 nvarchar(4000),
@p2 int,@p3 nvarchar(4000),
@p4 int',
@p0=1,@p1=N'Name1 ABC',@p2=5000,@p3=N'Name1 ABCX',@p4=4500
```

We can see all the columns is in WHERE clause,
which means by default all the columns will be used to detect concurrency conflicts.

It is fine if the table only has a few columns.
However, if the table has a lot of columns such as 30 columns,
this will cause performance problem.

Therefore, we need a "**Version**" Column
and the data type can be "**ROWVERSION**" or "**TIMESTAMP**"

The following code is the Tsql **after** we use "Version" Column with data type "RowVersion"
Now the Tsql only use "Version" column with the data type "Rowversion" to detect concurrency conflicts.

```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p2, [Score] = @p3
WHERE
    ([Id] = @p0) AND
    ([Name] = @p1)',
N'@p0 int,
@p1 nvarchar(4000),
@p2 nvarchar(4000),
@p3 int',
@p0=1,@p1=N'Name1 ABCX',@p2=N'Name1 ABCXX',@p3=4000
```

```
exec sp_executesql N'
UPDATE [dbo].[Gamer]
SET [Name] = @p2, [Score] = @p3
WHERE
    ([Id] = @p0) AND
       ([Version] = @p1)
SELECT [t1].[Version]
FROM [dbo].[Gamer] AS [t1]
WHERE
    ((@@ROWCOUNT) > 0) AND
       ([t1].[Id] = @p4)',
N'@p0 int,@p1 timestamp,@p2 nvarchar(4000),@p3 int,@p4 int',
@p0=1,@p1=0x00000000000007D1,@p2=N'Name1 ABCXX',@p3=4000,@p4=1
```