(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);

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 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);

OverwriteCurrentValues means discard all the changes from the current user.

**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.

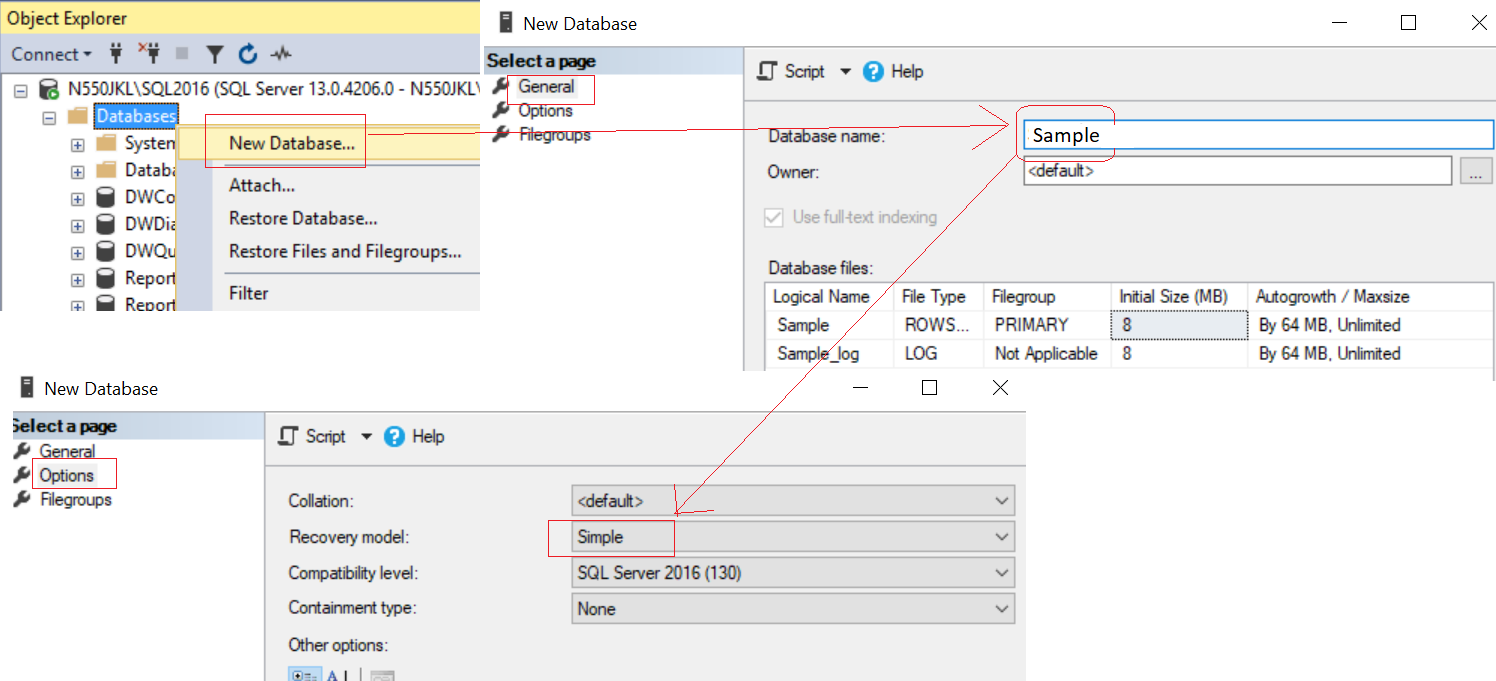
2. Web Form Application - Linq Query

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>

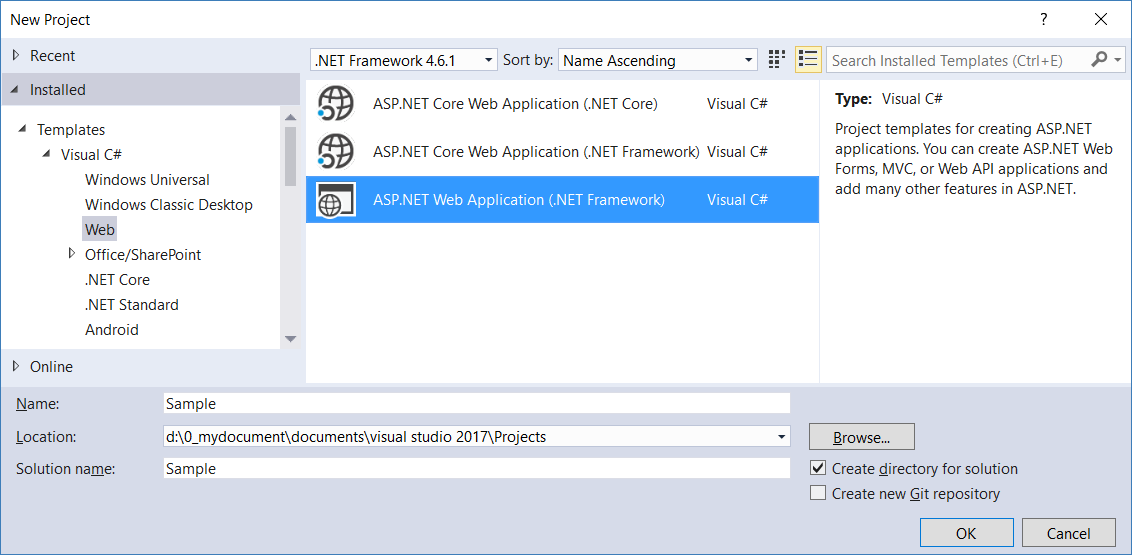
New Project --> Web -->[ASP.NET](http://asp.net/)**Web Application (.Net Framework)**

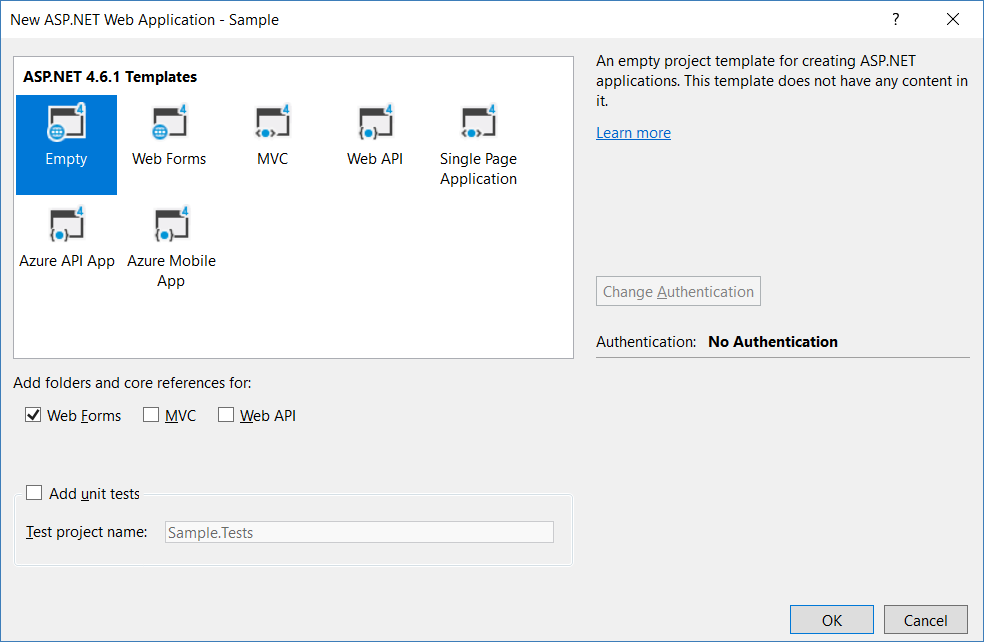
-->

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>

Graphical user interface, text, application

Description automatically generated

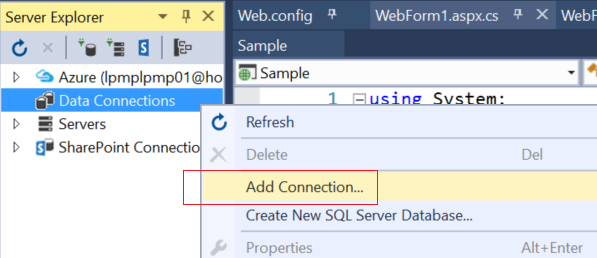
3.2. Linq to SQL

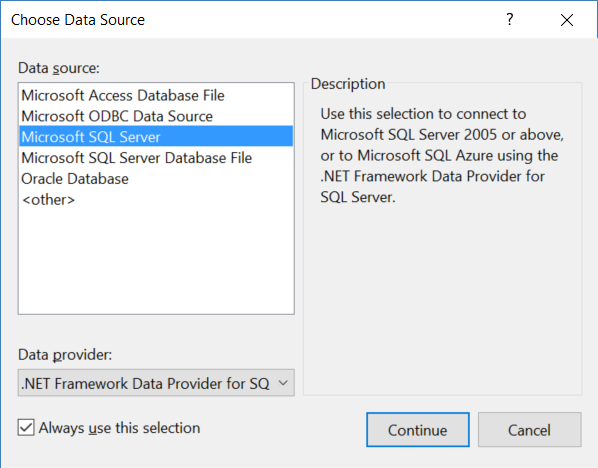
3.2.1. Add Connection

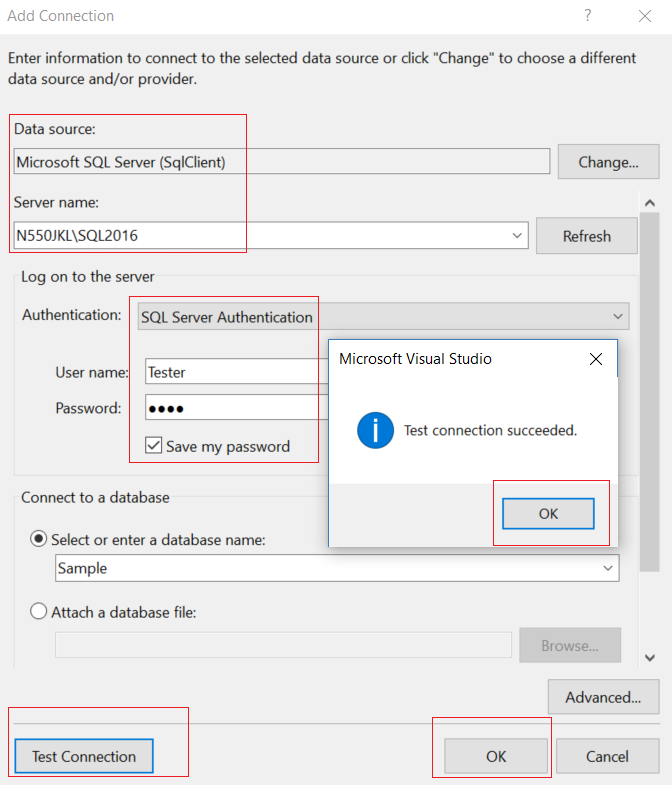
Server Explorer --> Data Connections --> Right click --> Add Connection...

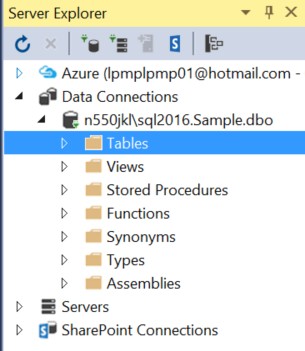
--> Microsoft SQL server -->

Enter your server and database details ....









3.2.2. Sample.dbml

ProjectName --> Right Click --> Add --> New Item...

--> Linq to SQL classes -->

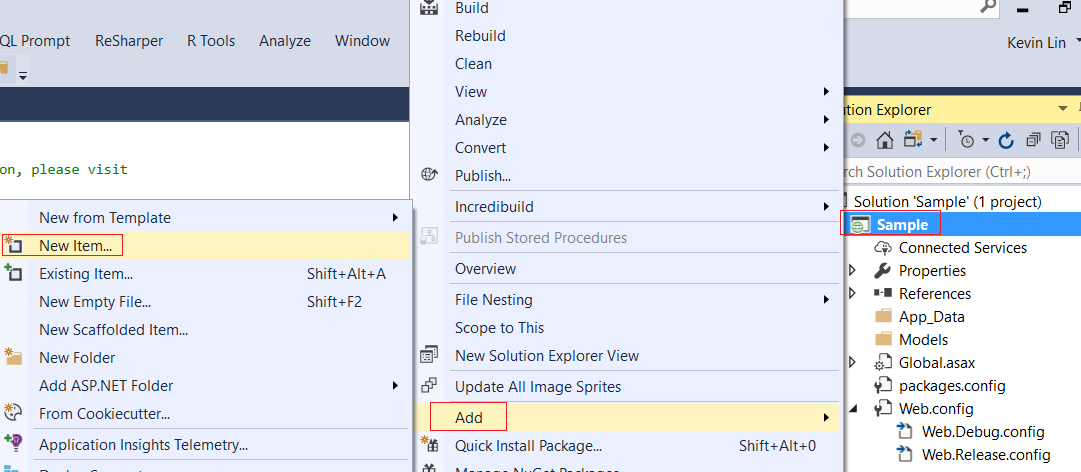
Name : **Sample.dbml**

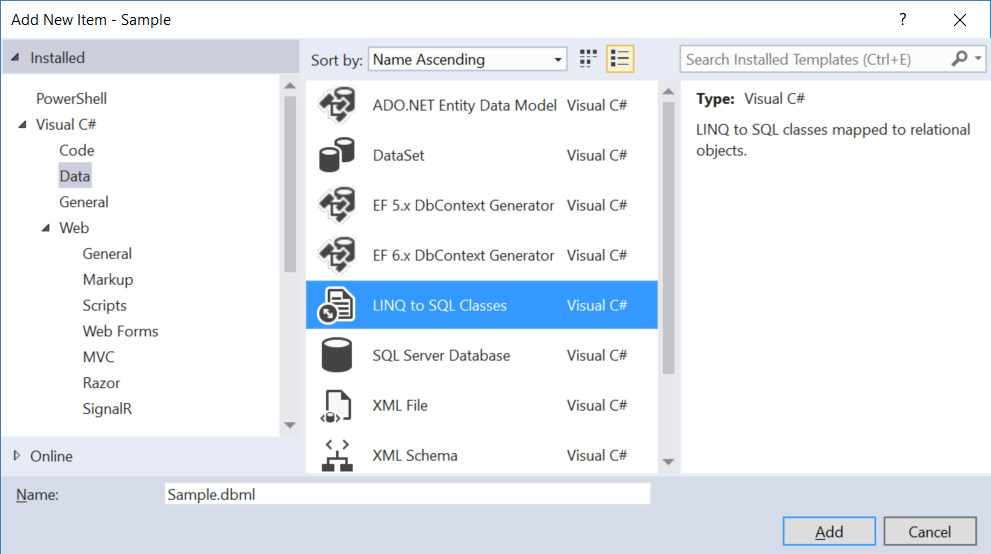
I name it as "Sample.dbml",

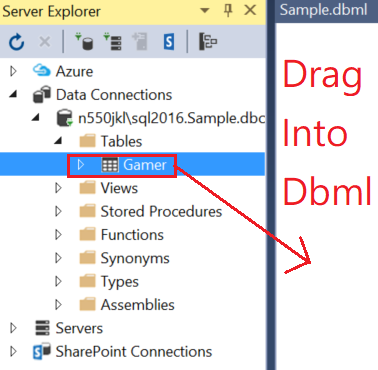
because I know this is for connection to "Sample" Database.

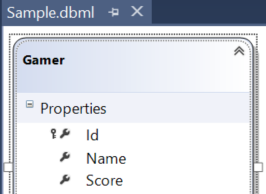
-->

Drag Table from Server Explorer into DBML



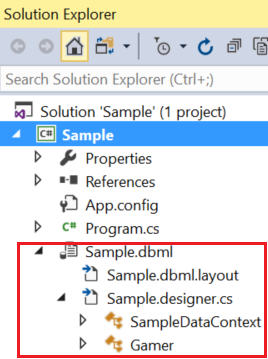






Save the dbml, it will generate the following files.

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**

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

<%@ 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.CurrentValue}<br/>");

                    Response.Write($"memberChangeConflict.OriginalValue=={memberChangeConflict.OriginalValue}<br/>");

                    Response.Write($"memberChangeConflict.DatabaseValue=={memberChangeConflict.DatabaseValue}<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();

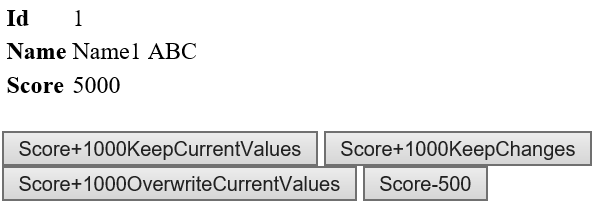
            }

        }

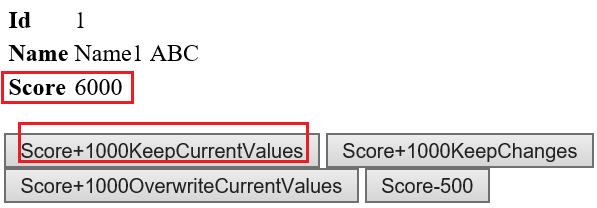
    }

}

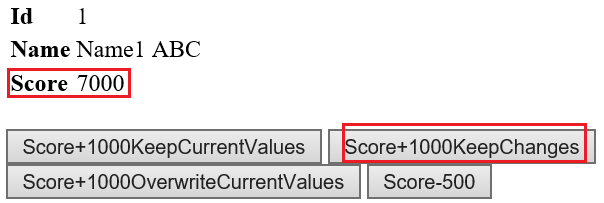
3.4. Run WebForm1.aspx



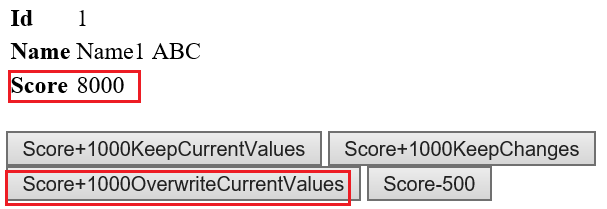
-----------------------------------



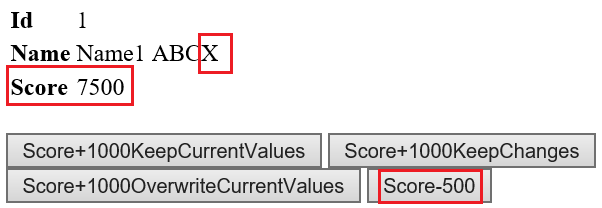
-----------------------------------



-----------------------------------

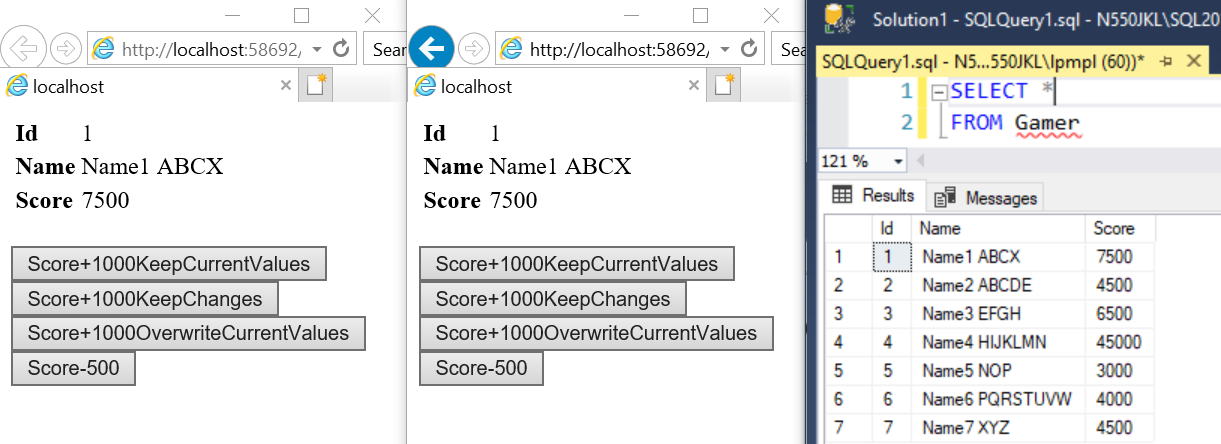


-----------------------------------

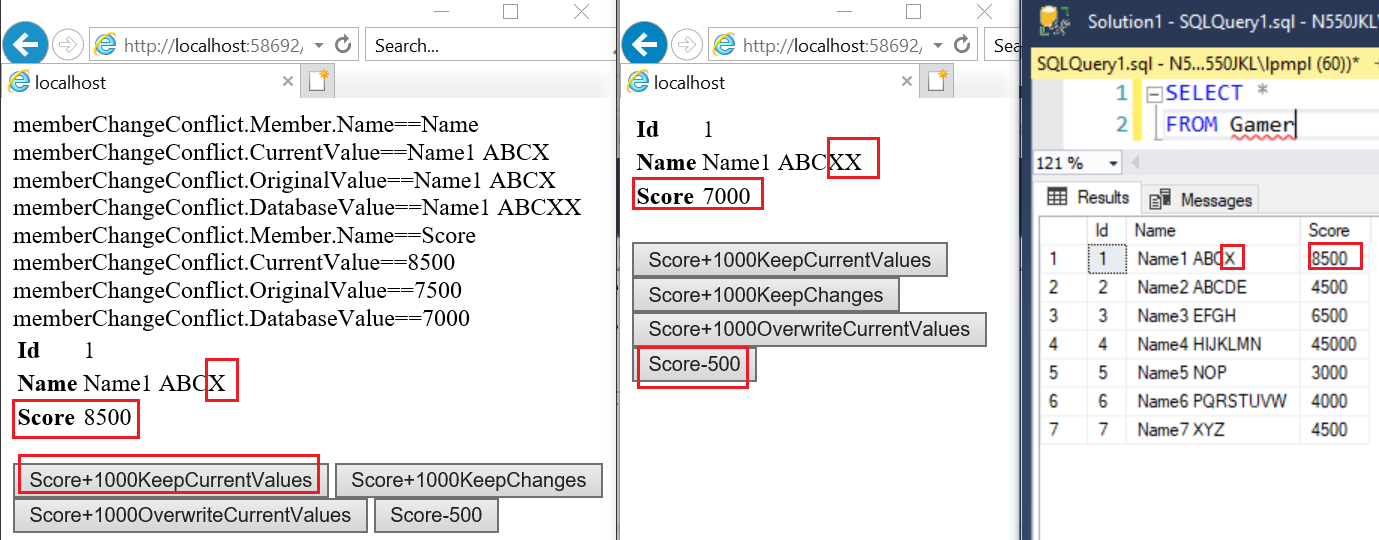


3.5. Run WebForm1.aspx with update conflicts

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);

KeepCurrentValues means keep all the new values from the current user.

**dbContextA** is used by current user

and tries to update the **Column1**.

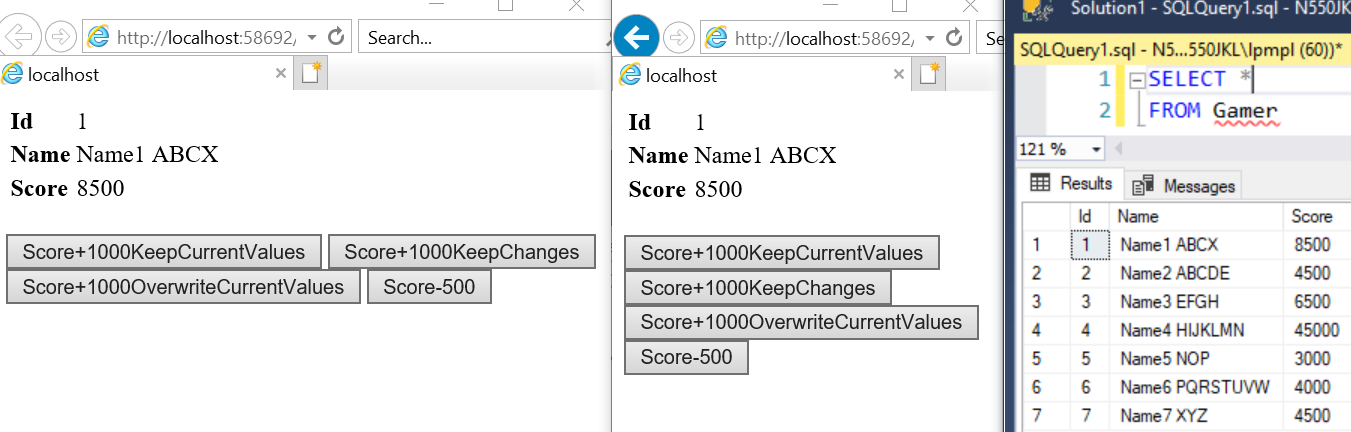
In the meantime, **dbContextB** is used by **another user**

and tries to update the **Column1**, and **Column2**.

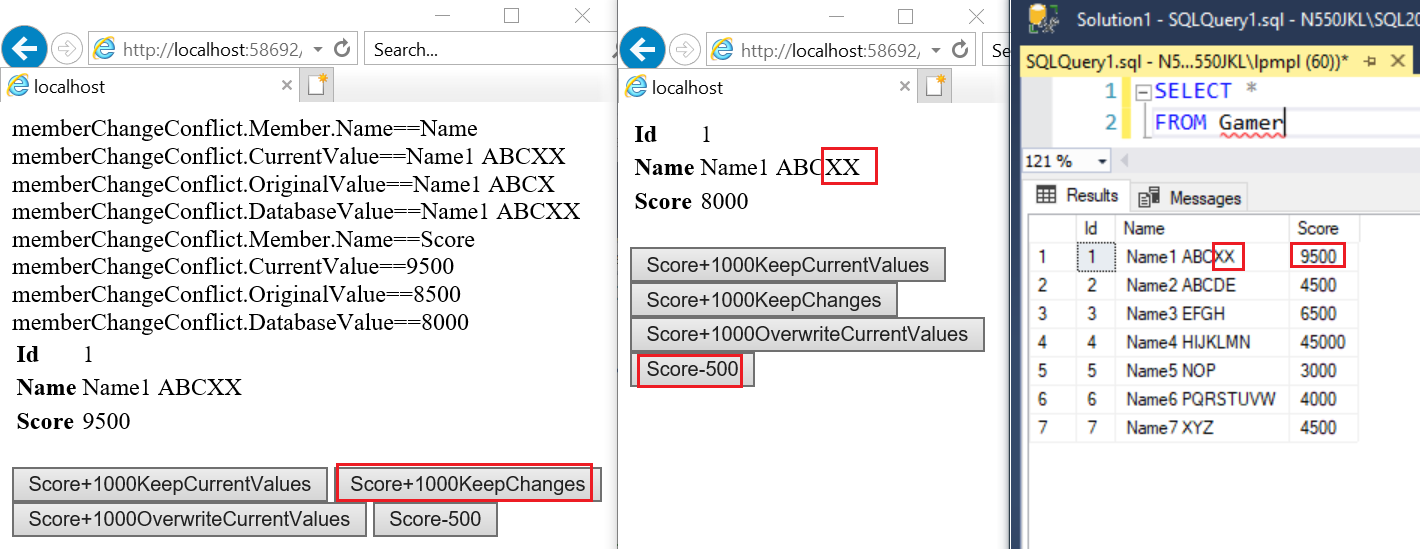
The **new** value of **Column1** from **dbContextA** will be saved into Database.

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);

KeepChanges means keep all the changes from all the users.

If any conflict, then keeps the new values from the current user.

**dbContextA** is used by the **current user**

and tries to update the **Column1**.

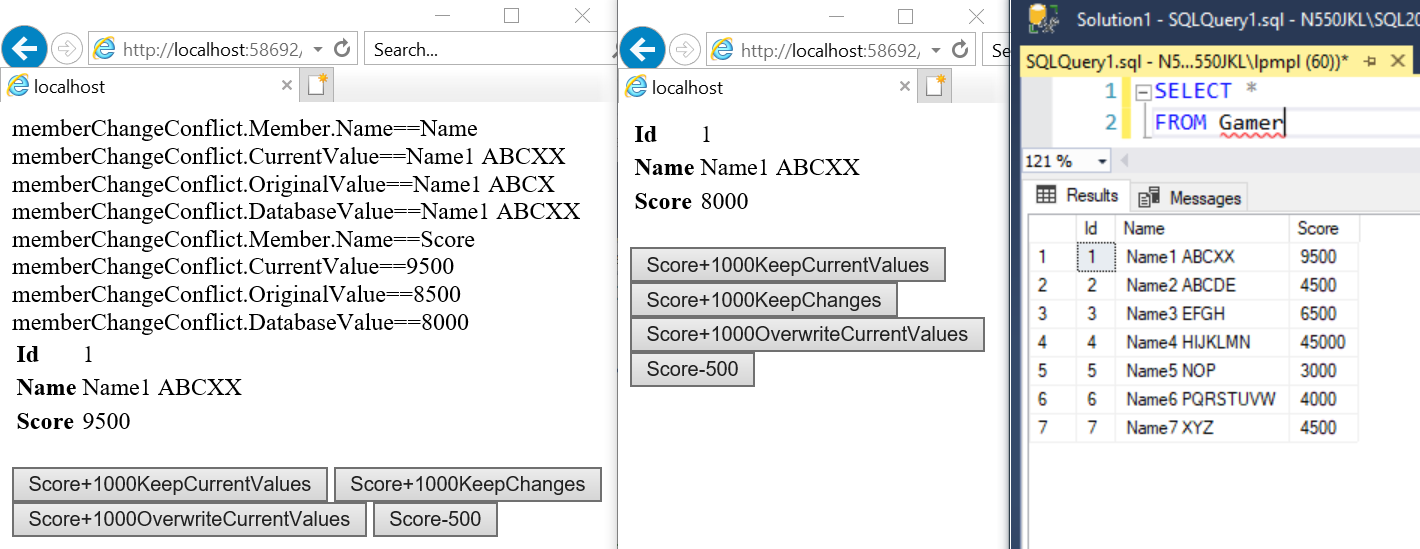
In the meantime, **dbContextB** is used by **another user**

and tries to update the **Column1**, and **Column2**.

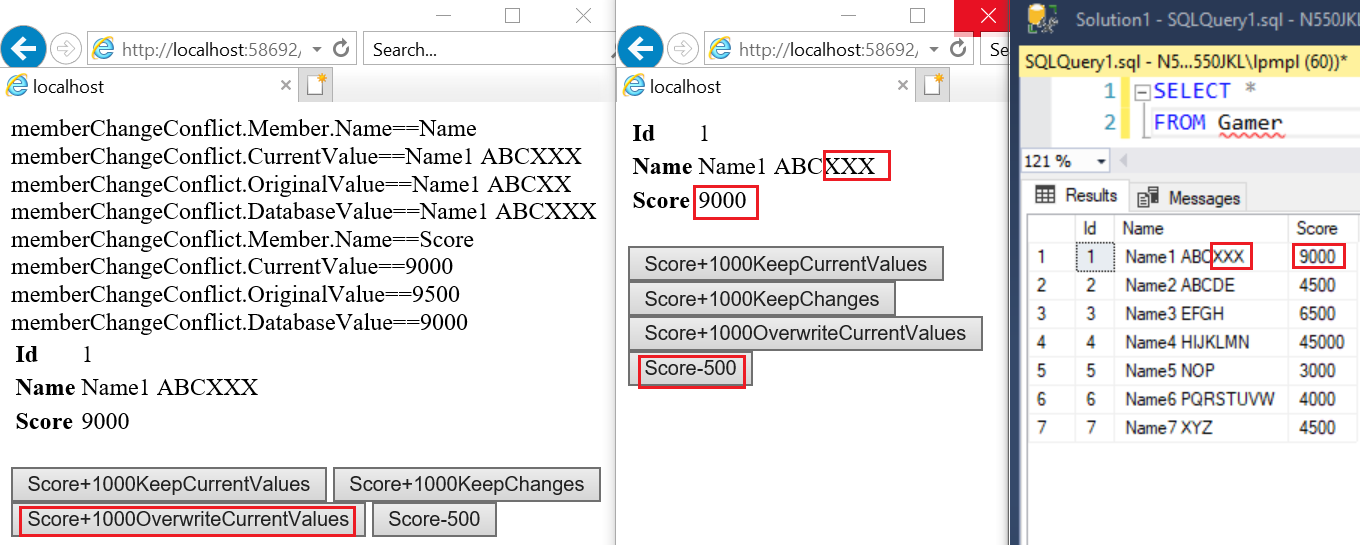
The **new** value of **Column1** from **dbContextA** will be saved into Database.

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);

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**.

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 ----------------------------------------------------------

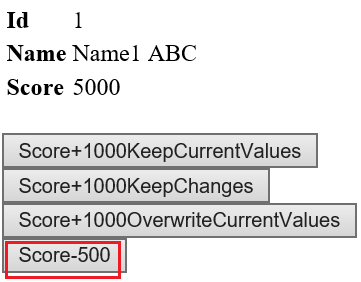
INSERT  INTO Gamer

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

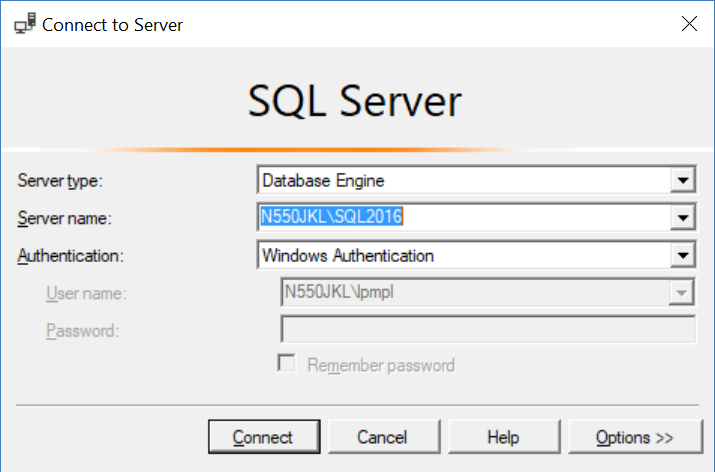


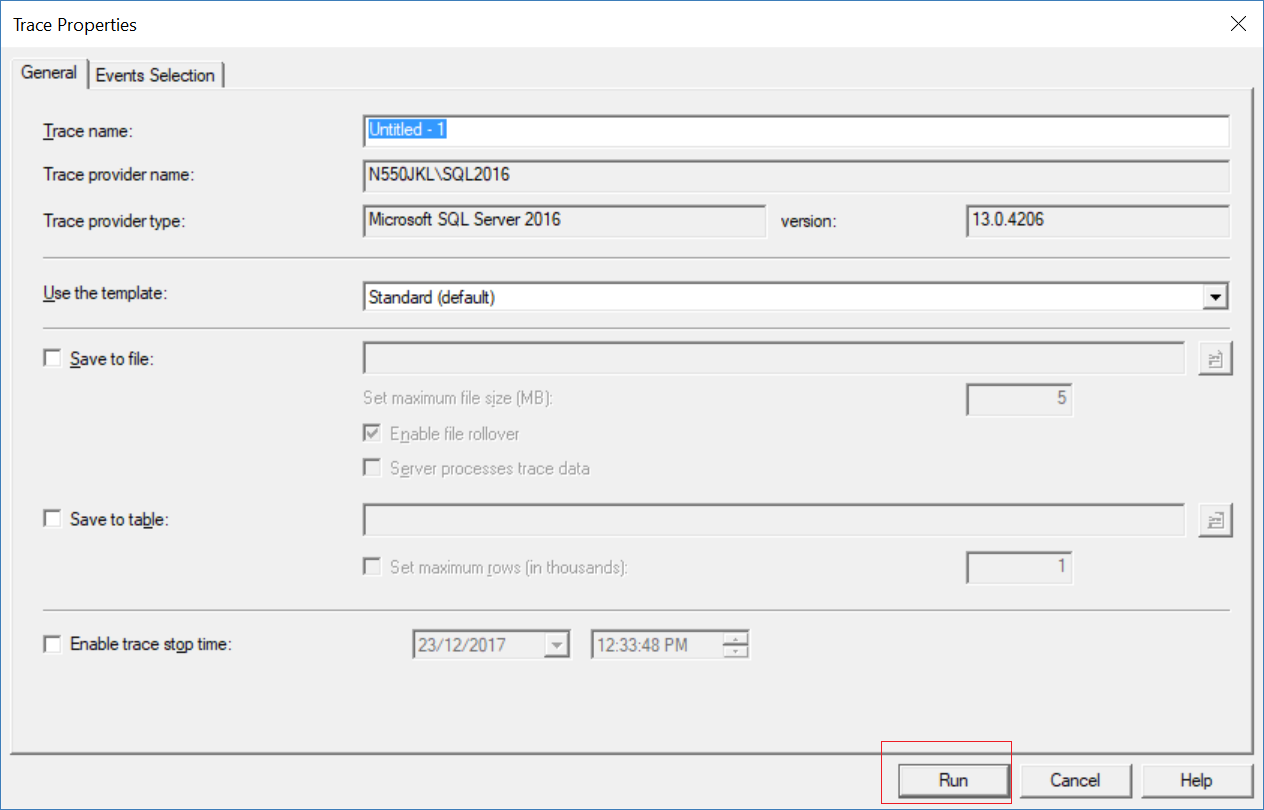
4.3. SQL Profiler (UpdateCheck Property)

Tools --> SQL Server Profiler

Graphical user interface, text, application

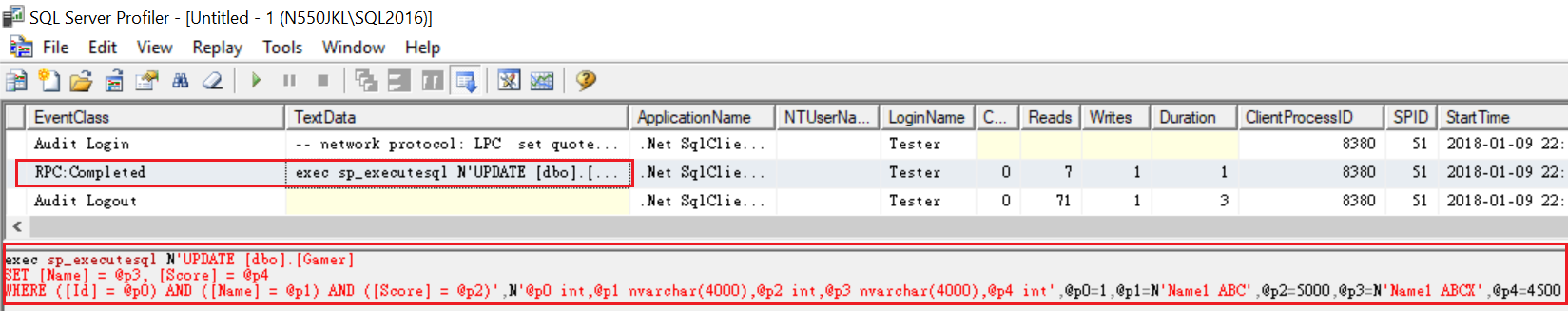
Description automatically generated





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),

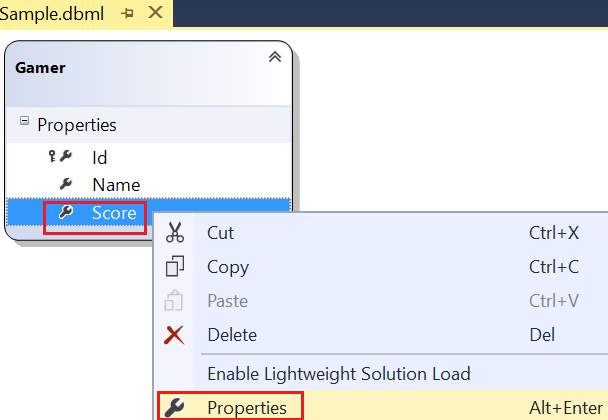
@p4 int',

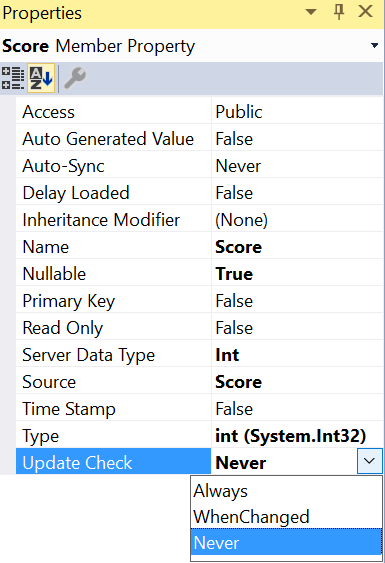
@p0=1,@p1=N'Name1 ABC',@p2=5000,@p3=N'Name1 ABCX',@p4=4500

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

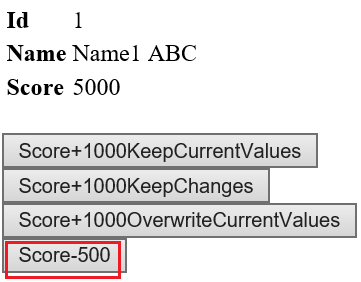
4.3.

WhenChanged

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

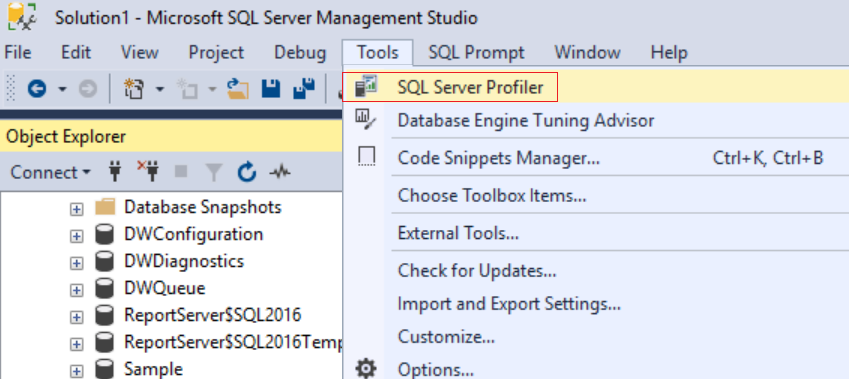
4.5. WebForm1.aspx (UpdateCheck Property)

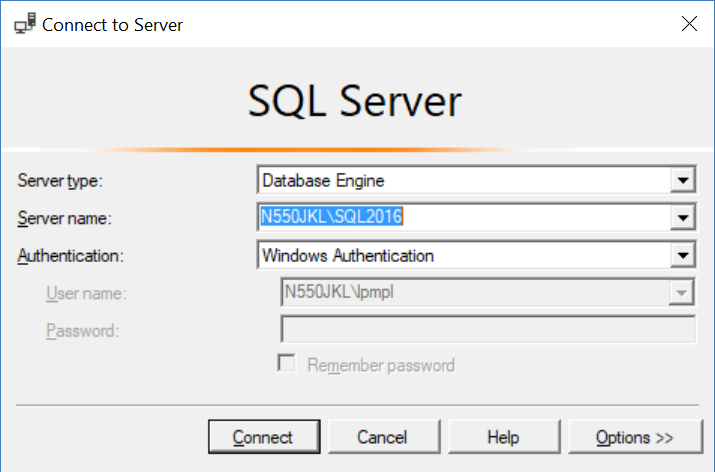
Turn on the Sql Profiler and run WebForm1.aspx

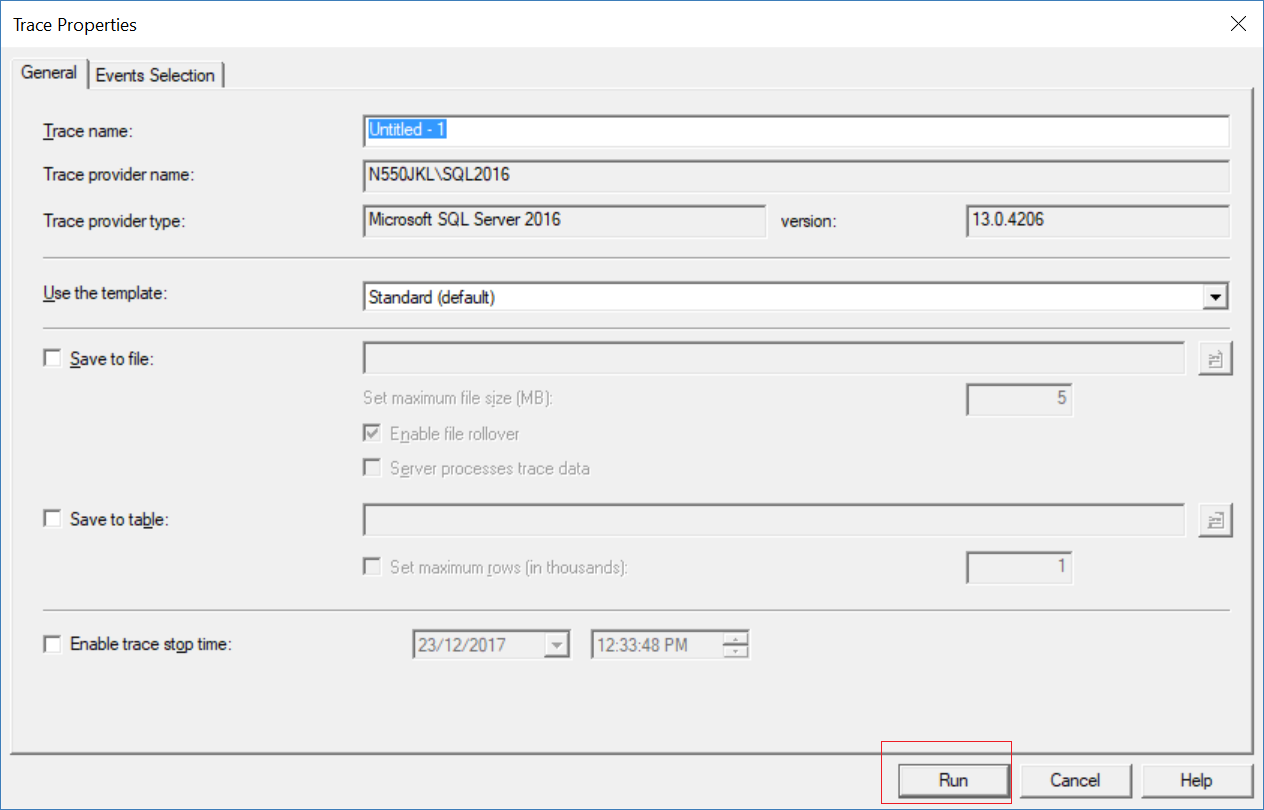


4.6. SQL Profiler (UpdateCheck Property)

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.

Graphical user interface, application

Description automatically generated

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**"

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

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

CREATE TABLE Gamer

    (

      Id INT PRIMARY KEY

             IDENTITY ,

      Name NVARCHAR(50) ,

      Score INT ,

    );

GO -- Run the previous command and begins new batch

--2 ----------------------------------------------------------

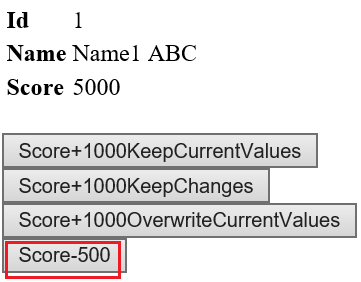
INSERT  INTO Gamer

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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

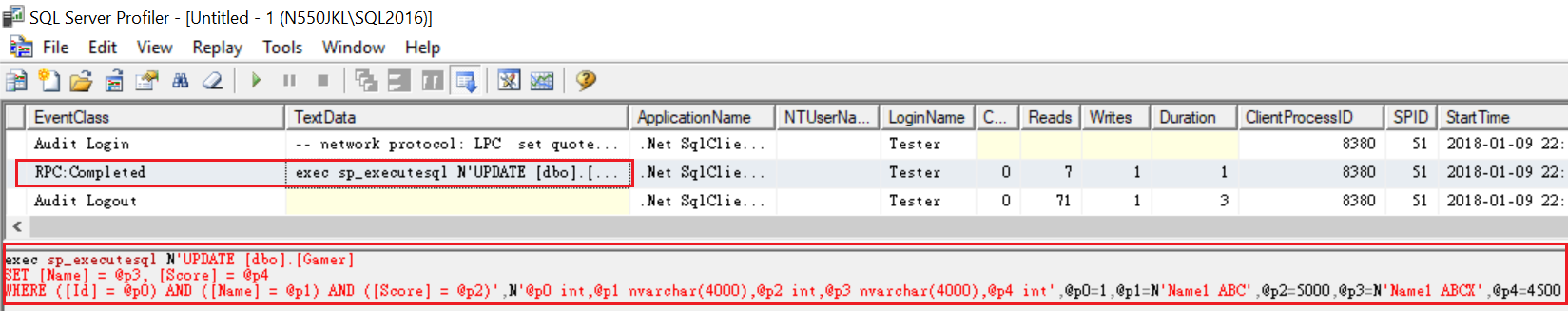
Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

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**"

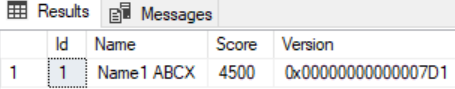
ALTER TABLE Gamer

ADD [Version] ROWVERSION

Now lets see what is going on in Gamer Table

SELECT  \*

FROM    Gamer;



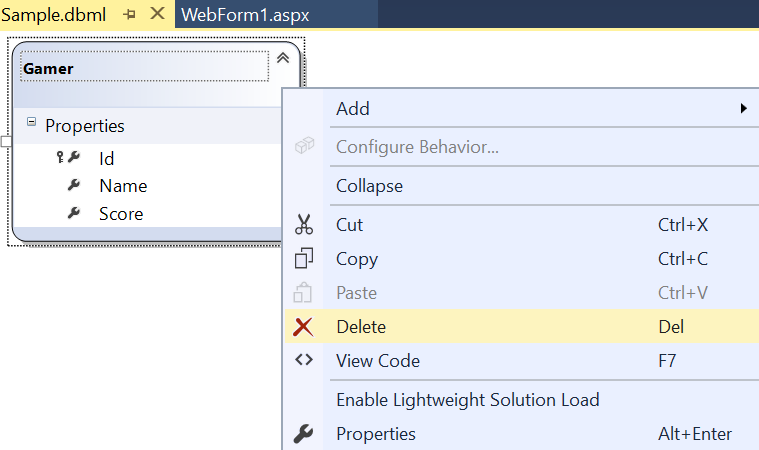
We can see the "**0x00000000000007D1**" is dynamically generated by the database.

The type is of "**Version**" is "**Rowversion**"

5.5. DBML (Rowversion and Timestamp)

In DBML

Select "Gamer" --> Delete



Graphical user interface, application

Description automatically generated

Graphical user interface, application, table

Description automatically generated

Save the dbml, it will generate the following files.

The DataContext context is the entry point to the database.

Graphical user interface, text, application

Description automatically generated

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)

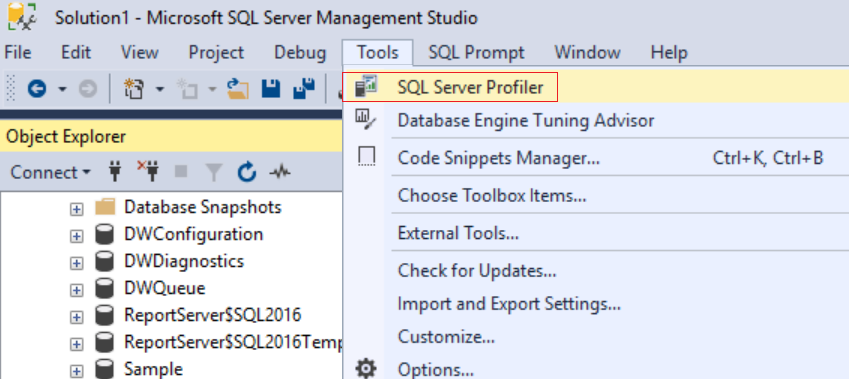
Turn on the Sql Profiler and run WebForm1.aspx

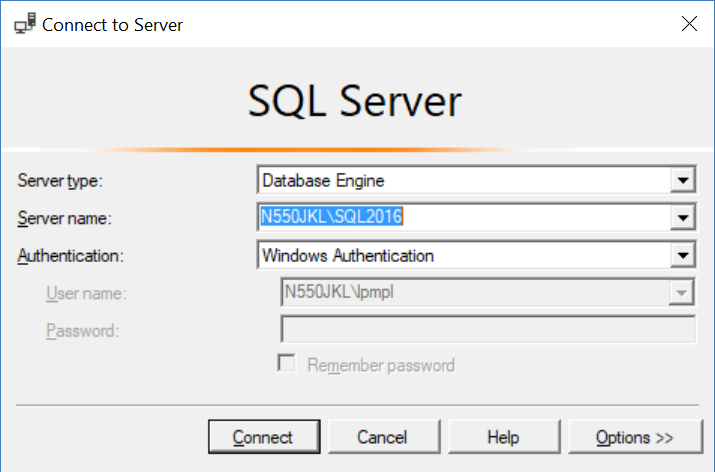
Graphical user interface, text, application

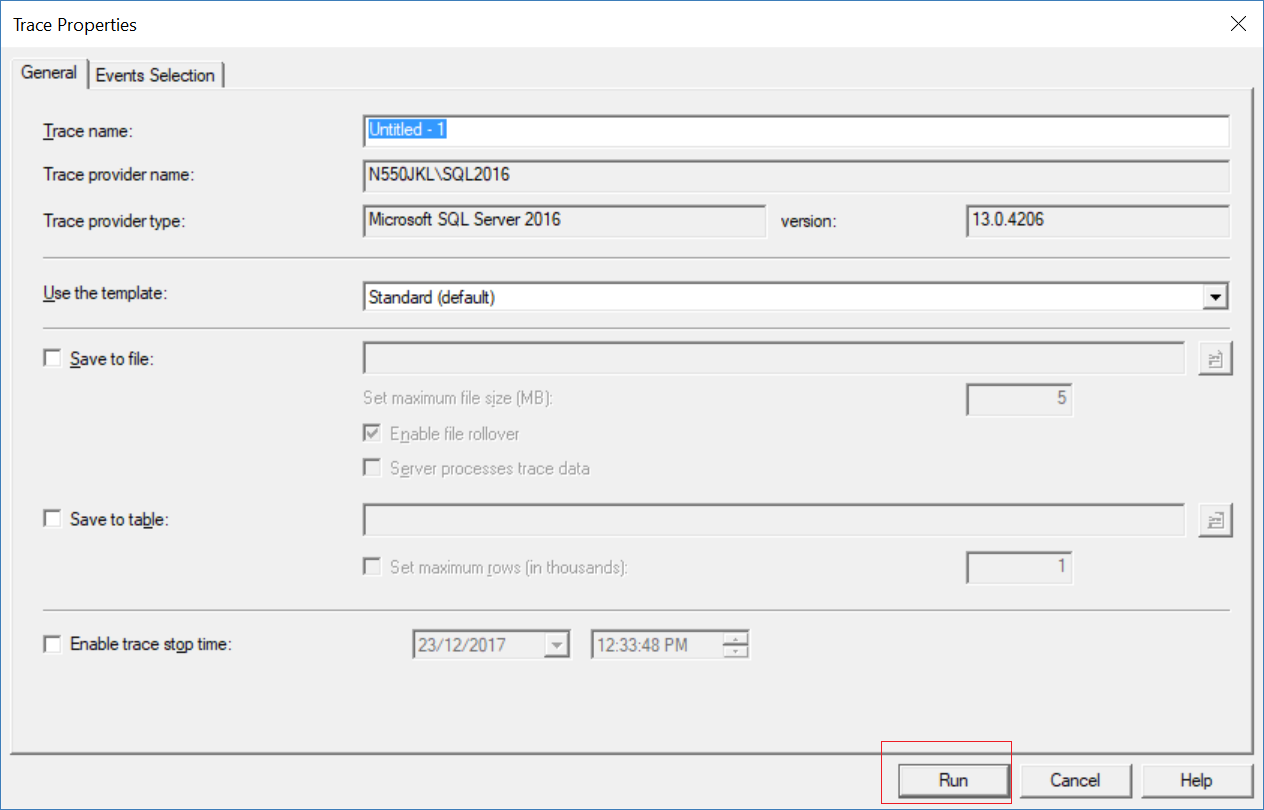
Description automatically generated

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.

Graphical user interface, application, Word

Description automatically generated

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