На рисунках 1 - 11 изображены код для тестирования

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 2 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 3 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 5 – код

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 – код

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – код

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Рисунок 8 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 9 – код

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – код

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 11 – код

На рисунке 12 изображён результат выполнения Unit-тестов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 12 – результат

Текст программы:

DatabaseWatcher:

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Timers;

namespace PracticeTools

{

public class DatabaseWatcher : IDisposable

{

private readonly Func<DataTable> reader;

private readonly Func<DataTable, DataRow, object> findRow;

private DataTable latestState;

private readonly Timer checker;

public DatabaseWatcher(Func<DataTable> reader, Func<DataTable, DataRow, object> findRow, int period)

{

this.reader = reader;

this.findRow = findRow;

checker = new Timer(period);

checker.Elapsed += Checker\_Elapsed;

}

private void Checker\_Elapsed(object sender, ElapsedEventArgs e)

{

checker.Stop();

try

{

var currentState = reader.Invoke();

if (currentState.Rows.Count < latestState.Rows.Count && OnDataNew != null)

{

if (OnDataAbsent != null)

{

foreach (var row in latestState.Rows)

{

var lastR = (DataRow)row;

var curR = findRow(currentState, lastR);

if (curR == null)

OnDataAbsent(this, lastR);

}

}

}

else if (OnDataChanged != null || OnDataNew != null)

{

foreach (var row in currentState.Rows)

{

var curR = (DataRow)row;

var lastR = (DataRow)findRow(latestState, curR);

if (lastR == null)

{

if (OnDataNew != null)

OnDataNew(this, curR);

}

else if (OnDataChanged != null)

{

for (int c = 0; c < currentState.Columns.Count; c++)

{

var lastCell = lastR[c];

var curCell = curR[c];

if (!lastCell.Equals(curCell))

{

OnDataChanged(this, curR);

break;

}

}

}

}

}

latestState = currentState;

}

finally

{

checker.Start();

}

}

public delegate void OnDataChangedDelegate(DatabaseWatcher sender, DataRow changedRow);

public delegate void OnNewDataPresetnsDelegate(DatabaseWatcher sender, DataRow newRow);

public delegate void OnExistingDataAbsentsDelegate(DatabaseWatcher sender, DataRow absentRow);

public event OnDataChangedDelegate OnDataChanged;

public event OnNewDataPresetnsDelegate OnDataNew;

public event OnExistingDataAbsentsDelegate OnDataAbsent;

public void Dispose()

{

StartWatching();

checker.Dispose();

}

public void StartWatching()

{

latestState = reader.Invoke();

checker.Start();

}

public void StopWatching()

{

checker.Stop();

}

}

}

ImageLoader:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace PracticeTools

{

public class ImageLoader

{

private readonly PictureBox pictureBox;

private readonly Label label;

public ImageLoader(PictureBox image, Label altText)

{

pictureBox = image;

label = altText;

}

public void LoadImage(string path)

{

try

{

pictureBox.Image = Image.FromFile(path);

pictureBox.Visible = true;

label.Visible = false;

}

catch

{

ResetImage();

label.Text = string.IsNullOrWhiteSpace(path)

? "Файл не указан"

: $"Не удалось загрузить изображение из файла '{path}'";

}

}

public void ResetImage()

{

pictureBox.Visible = false;

label.Visible = true;

label.Width = pictureBox.Width;

label.Height = pictureBox.Height;

label.Text = "";

}

}

}

DatabaseWatcherUnitTest:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PracticeTools;

using System;

using System.Collections.Generic;

using System.Data;

using System.Threading;

namespace PracticeToolsTest

{

[TestClass]

public class DatabaseWatcherUnitTest

{

private const int Period = 2000;

private const int TestRowCount = 5;

bool wasDatabaseRead = false;

int findRowCallCount = 0;

private readonly DataTable FirstStepDataTable = new DataTable();

private readonly DataTable SecondStepDataTable = new DataTable();

private readonly List<DataRow> ChangedRow = new List<DataRow>();

private readonly List<DataRow> AbsentRow = new List<DataRow>();

private readonly List<DataRow> NewRow = new List<DataRow>();

public DatabaseWatcherUnitTest()

{

FirstStepDataTable.Columns.Add(new DataColumn("TestKey"));

FirstStepDataTable.Columns.Add(new DataColumn("TestData"));

FirstStepDataTable.PrimaryKey = new DataColumn[] { FirstStepDataTable.Columns[0] };

SecondStepDataTable.Columns.Add(new DataColumn("TestKey"));

SecondStepDataTable.Columns.Add(new DataColumn("TestData"));

SecondStepDataTable.PrimaryKey = new DataColumn[] { SecondStepDataTable.Columns[0] };

}

private DataTable FakeReadDatabase()

{

if (wasDatabaseRead)

{

return SecondStepDataTable;

}

else

{

wasDatabaseRead = true;

return FirstStepDataTable;

}

}

private object FakeFindRow(DataTable source, DataRow key)

{

++findRowCallCount;

return source.Rows.Find(key[0]);

}

private void DataChangeHandler(DatabaseWatcher sender, DataRow changedRow)

{

ChangedRow.Add(changedRow);

}

private void DataAbsentHandler(DatabaseWatcher sender, DataRow absentRow)

{

AbsentRow.Add(absentRow);

}

private void DataNewHandler(DatabaseWatcher sender, DataRow newRow)

{

NewRow.Add(newRow);

}

private DatabaseWatcher PrepareDatabaseWatcherForTest(Func<DataTable> reader, Func<DataTable, DataRow, object> findRow)

{

DatabaseWatcher w = new DatabaseWatcher(reader, findRow, Period);

w.OnDataAbsent += DataAbsentHandler;

w.OnDataNew += DataNewHandler;

w.OnDataChanged += DataChangeHandler;

return w;

}

private void FillDataTables()

{

for (int i = 1; i <= TestRowCount; i++)

{

FirstStepDataTable.Rows.Add(i, -i);;

SecondStepDataTable.Rows.Add(i, -i);

}

}

[TestMethod]

public void StartWatchTest()

{

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

Thread.Sleep(Period \* 3 / 2);

Assert.IsFalse(wasDatabaseRead);

Assert.AreEqual(findRowCallCount, 0);

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.IsTrue(wasDatabaseRead);

Assert.AreEqual(findRowCallCount, TestRowCount);

Thread.Sleep(Period);

Assert.AreEqual(findRowCallCount, TestRowCount \* 2);

}

[TestMethod]

public void StopWatchTest()

{

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.AreEqual(findRowCallCount, TestRowCount);

w.StopWatching();

Thread.Sleep(Period \* 2);

Assert.AreEqual(findRowCallCount, TestRowCount);

}

[TestMethod]

public void NoChangesTest()

{

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

Thread.Sleep(Period);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

w.StopWatching();

}

[TestMethod]

public void ChangeRowTest()

{

var changeData = "500";

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

SecondStepDataTable.Rows[2]["TestData"] = changeData;

SecondStepDataTable.Rows[4]["TestData"] = changeData;

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.AreEqual(2, ChangedRow.Count);

Assert.AreEqual(changeData, ChangedRow[0]["TestData"]);

Assert.AreEqual("3", ChangedRow[0]["TestKey"]);

Assert.AreEqual(changeData, ChangedRow[1]["TestData"]);

Assert.AreEqual("5", ChangedRow[1]["TestKey"]);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

ChangedRow.Clear();

Thread.Sleep(Period);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

w.StopWatching();

}

[TestMethod]

public void NewRowTest()

{

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

SecondStepDataTable.Rows.Add(6, 600);

SecondStepDataTable.Rows.Add(7, 700);

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(2, NewRow.Count);

Assert.AreEqual("6", NewRow[0]["TestKey"]);

Assert.AreEqual("7", NewRow[1]["TestKey"]);

Assert.AreEqual(0, AbsentRow.Count);

NewRow.Clear();

Thread.Sleep(Period);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

w.StopWatching();

}

[TestMethod]

public void AbsentRowTest()

{

var w = PrepareDatabaseWatcherForTest(FakeReadDatabase, FakeFindRow);

FillDataTables();

var delRow = SecondStepDataTable.Rows[1];

var delKeyData = delRow["TestKey"];

SecondStepDataTable.Rows.Remove(delRow);

w.StartWatching();

Thread.Sleep(Period \* 3 / 2);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(1, AbsentRow.Count);

Assert.AreEqual(delKeyData, AbsentRow[0]["TestKey"]);

AbsentRow.Clear();

Thread.Sleep(Period);

Assert.AreEqual(0, ChangedRow.Count);

Assert.AreEqual(0, NewRow.Count);

Assert.AreEqual(0, AbsentRow.Count);

w.StopWatching();

}

}

}