



DME API. Database Trigger Demo

Contents

Requirements.....	3
Installation and Start.....	3
Visual Studio Solution	9
Sample Web Service project	10
Library for accessing Web Service from SQL Server	13
SQL CLR library for SQL Server	14
Deploying	14
Check installation.....	17
Example of using.....	17
Scripts for creating triggers.....	19
The TriggerDemoGui project	21

This document describes example for straight access from the database to matching service.

The Matching Engine is configured for suppressing management. It finds how many preloaded data source contains duplicates for given record. All calls are fired from the database they can be connected to triggers or stored procedures.

This document describes the architecture of example, requirements, installation into database process.

Requirements

For working with this example, the following applications should be installed on PC:

- SQL Server (or MS SQL Express Server)
- MS Management Studio
- Visual Studio 2017 (It may be Community Version)

Installation and Start

The steps of this section are quick-start installation. They adjust the execution of Visual Studio projects with hardcoded data sources, names, etc. It allows getting working environment fast and easy way. The next section describes how to work this sample and you will be able to adjust for your data sources and purposes.

- Build the Visual Studio projects. Check that projects *SampleWebService*, *SqlServerDataladderIntegration* built successfully
- Create in Sql Server database with name 'TriggerDemoBase'.
- Create folder 'C:\ApiDemo'. Put in this folder the output of *SqlServerDataladderIntegration* Project.
- Implement scripts from the 'Installation and Start.zip' archive to the created database.

Scripts should be executed one by one:

- First - Attach assemblies and SP.sql
 - Second - Create dummy tables.sql
 - Third - Create triggers for dummy tables.sql
 - Forth - Fill Master Table
- Create a project in DME. Project must have name: **ExampleWithTriggers**

- Attach **MasterTable** source created from DB to project
- Do not make transformation
- Select 'State' and 'ContactName' fields as Fuzzy Criteria
- Save the project
- Close project
- Remember 'Data path' and 'Project path' from DME settings

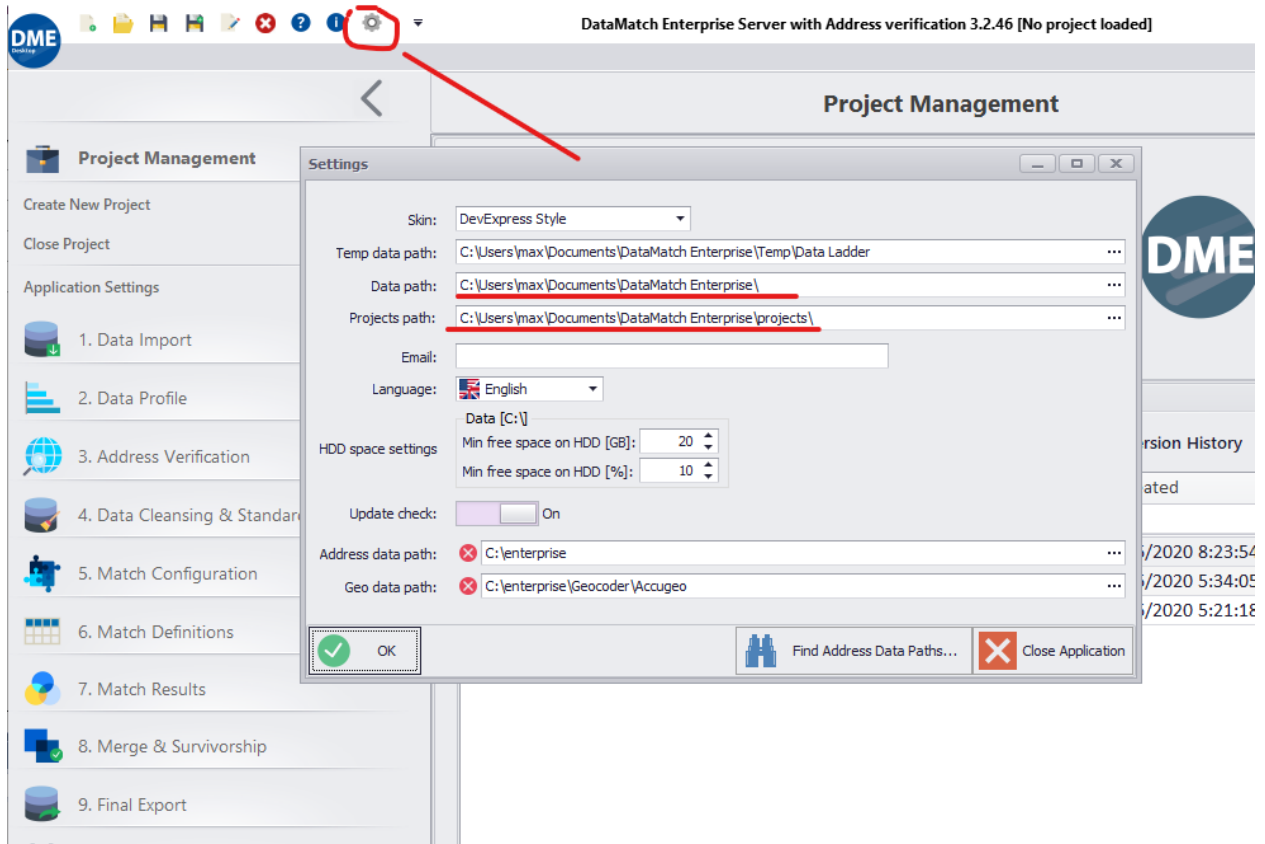


Fig.1

- Set Project Name and Data path in 'SampleWebProject' settings

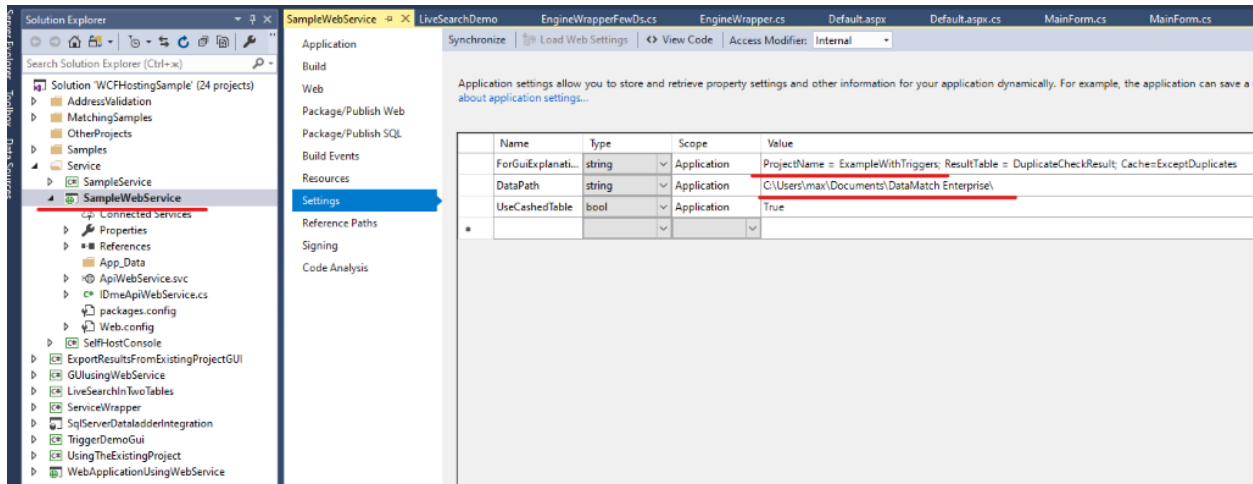


Fig.2

- Run *SampleWebService* project in Visual Studio. Run with **WCF Test Client**. This can be executed in two ways.
 - Way 1 (Fig.3)

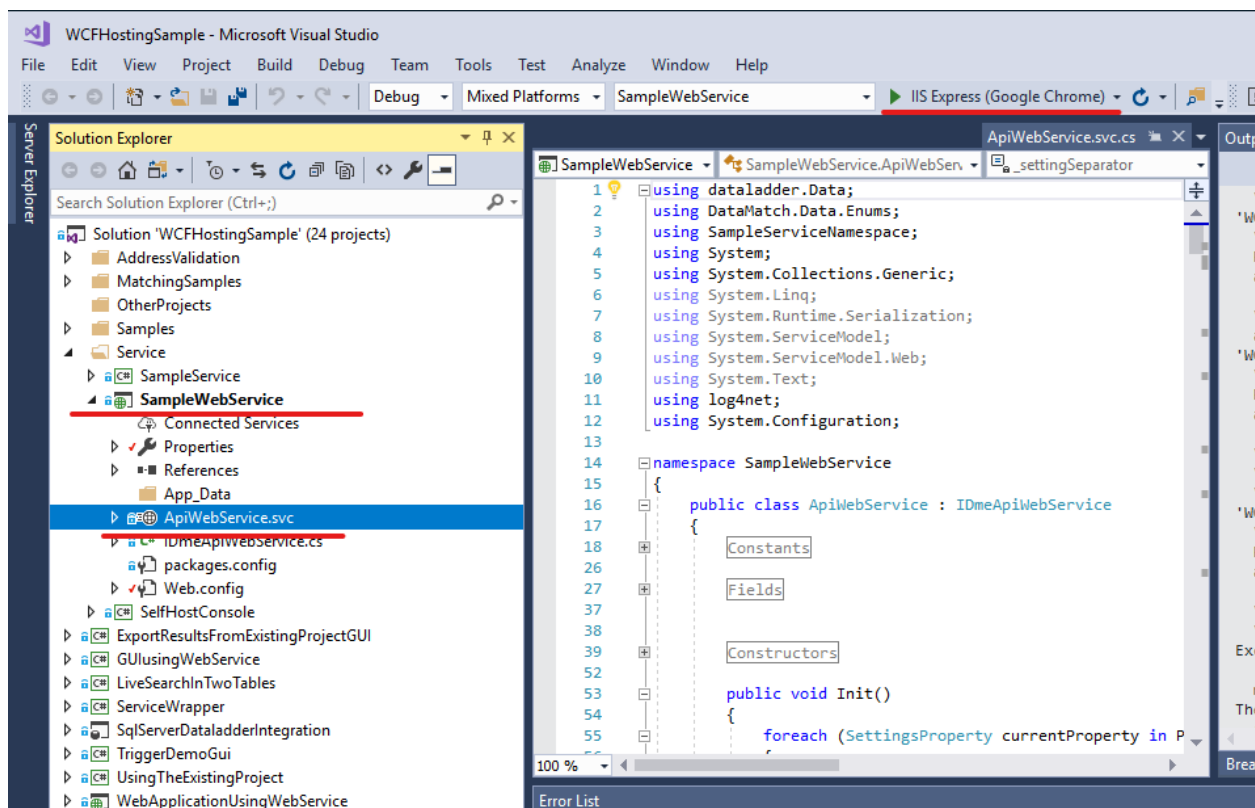


Fig.3 The first way to start WCF service – direct starting in Visual Studio

- Select this project as **Startup project**

- Choose **ApiWebService.svc**
- Run Debug

○ Way 2 (Fig.4)

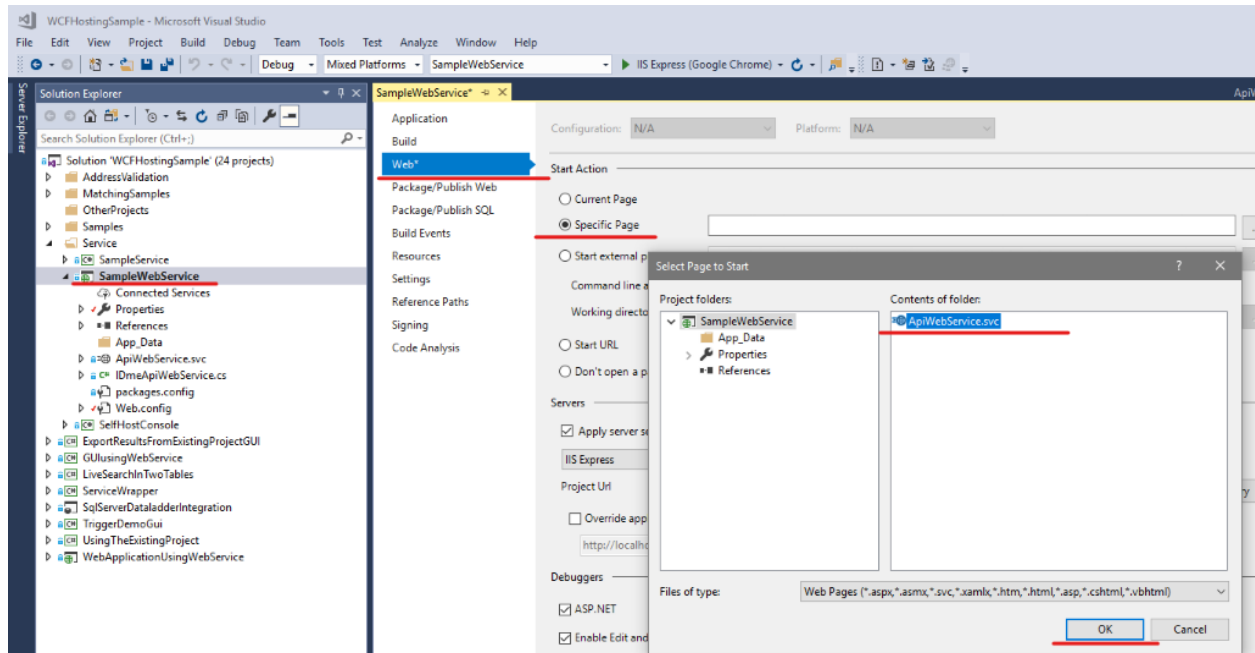


Fig.4 The second way to start WCF service – change settings of project starting

- Choose properties of project
- Choose *Web* page
- Choose *Specific Page* option
- Choose **ApiWebService.svc** file and press **OK**
- Run Debug
- Check that WCF Test Client has the same view as in Fig.5

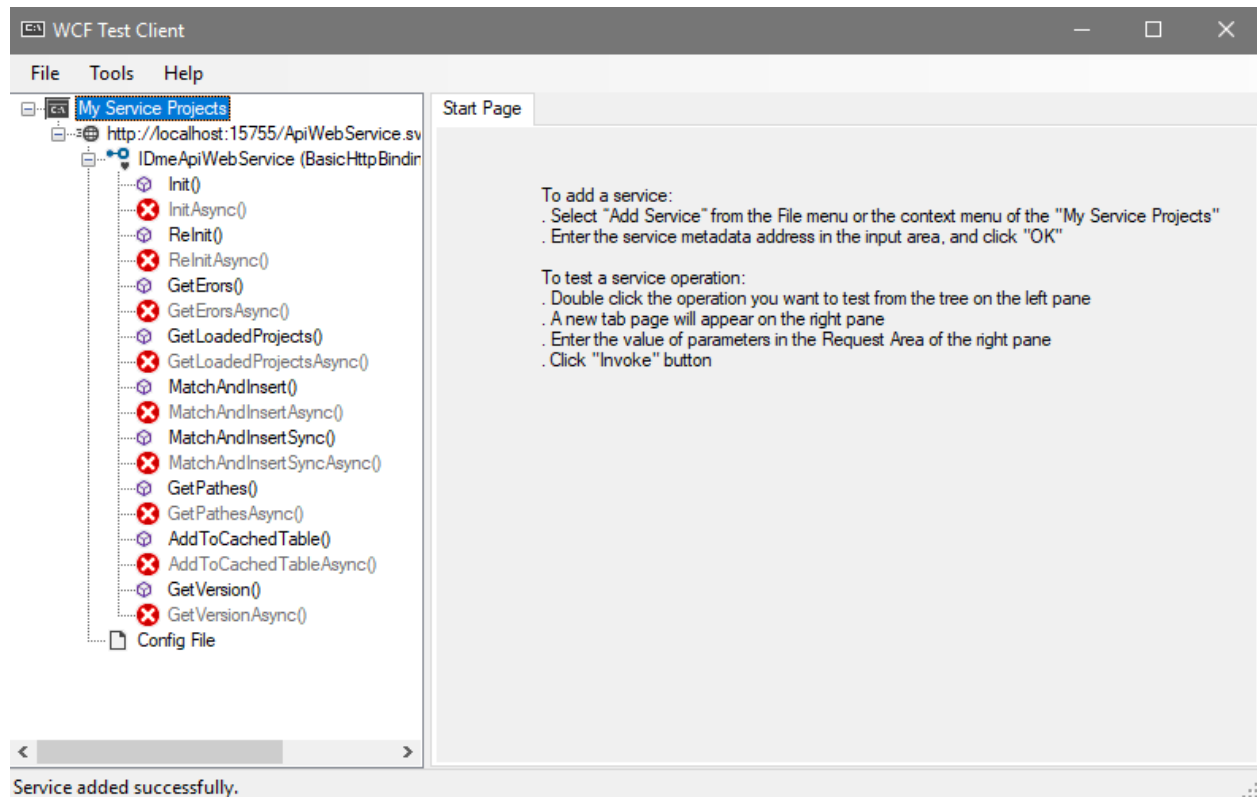


Fig 5

- Check that assemblies in database can interact with WCF service
Execute next query in database:

`EXEC dbo.spGetVersion`

Expected result (the first number is a version of DME API, the second is a version of API Demo):

`3.0.26.0 (1.0.4.0)`

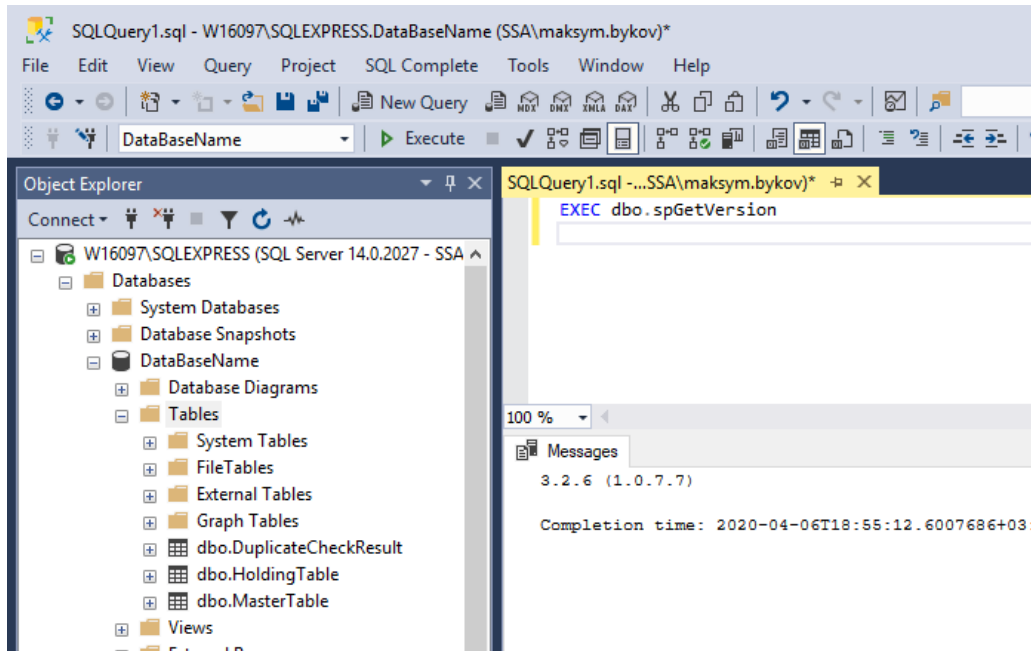


Fig.6

- Enter a correct connection string to settings of *TriggerDemoGui* Project

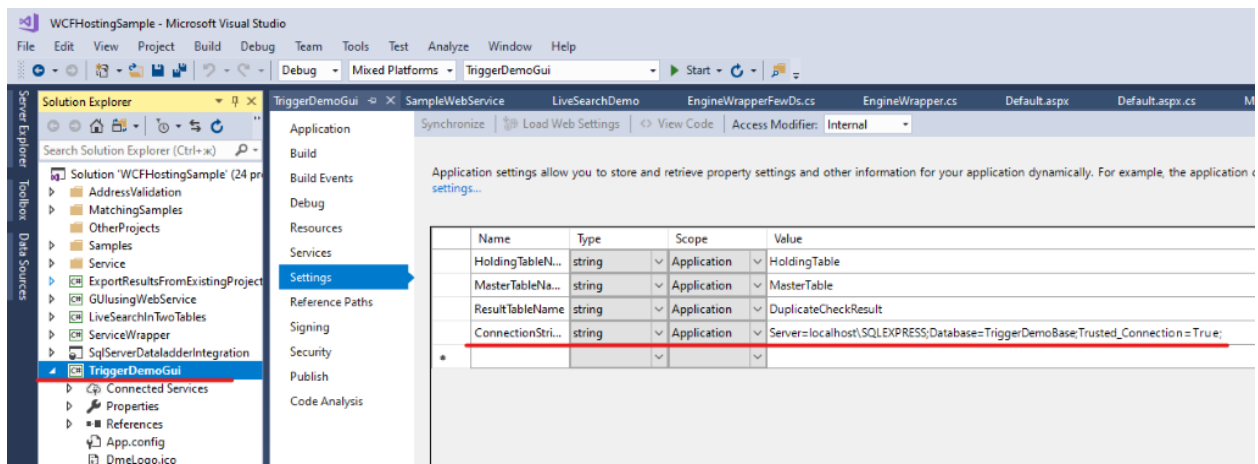


Fig.7

- Start Wcf Service in a separate thread. It possible to do in Visual Studio with keyboard shortcut Ctrl+F5
- Run TriggerDemoGui

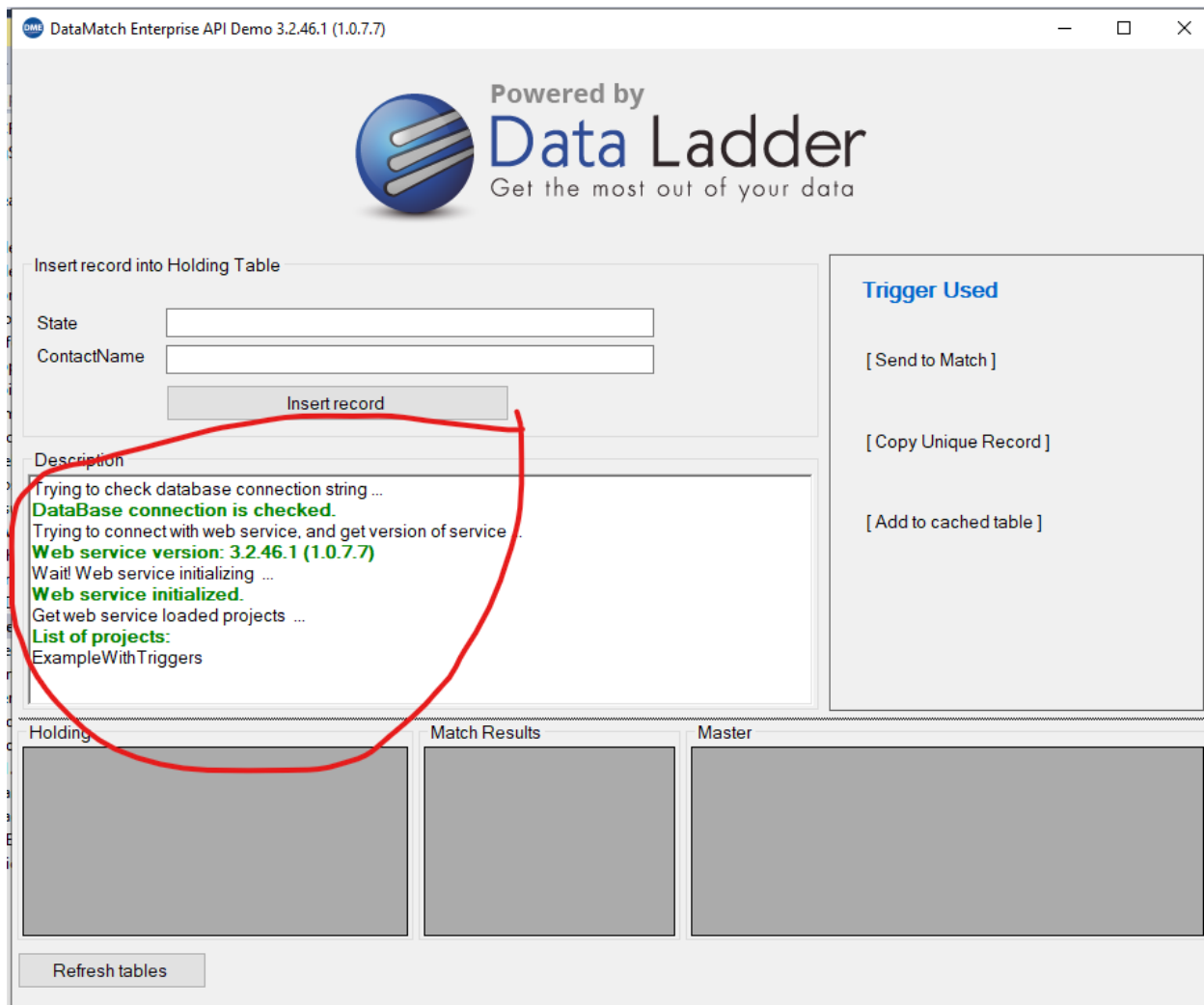


Fig.8 The sample Triger Demo is running and all checkings are successful finished

- Check that DB is connected and service is found (Fig.8)

Visual Studio Solution

This example consists of three projects.

The first project is a web application that provides functions of web service. This service can receive commands, data for matching and can write results of matching into the database. Service receives commands by SOAP so any database that can use SOAP can use this service

The second project is created for interacting with the web service. And this project is loaded into MS SQL Server. It allows using of web service from SQL Server.

The third project is SQL CLR assembly that work as a bridge between SQL Server objects (triggers, stored procedures) and the second project

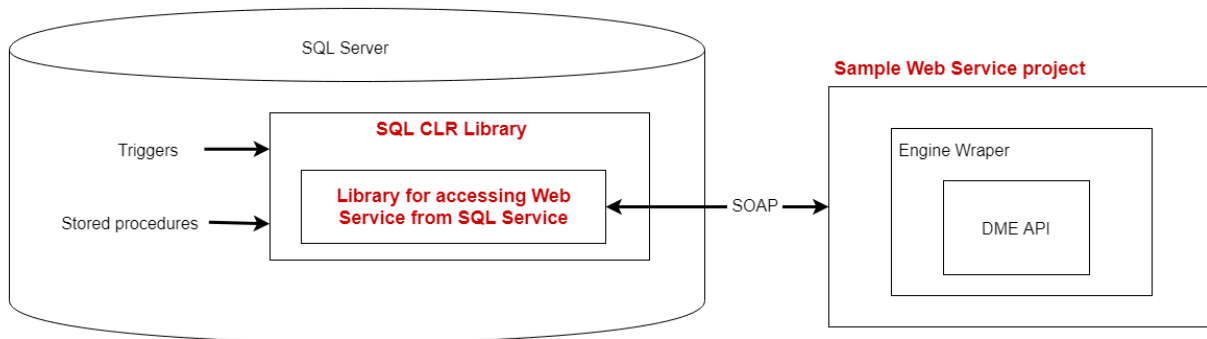


Fig. 9

All three projects are described below

Sample Web Service project

This project provides suppressing management service over HTTP.

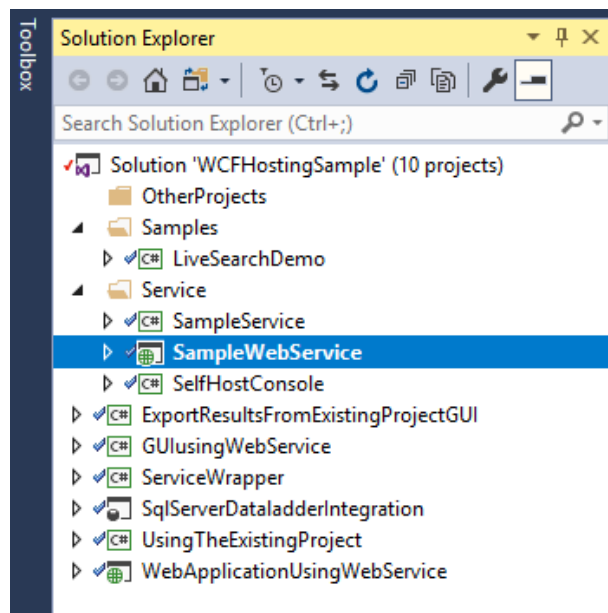


Fig. 10. The web application project inside Visual Studio

In initial stage project loads data from data source. Later, the service receives inserted records and verify them in the data source if they are similar ones or not. Check Results are written into predefined database table.

The table must have definite structure. It must have columns: 'id'(bigint), 'StrongDuplicate'(int), 'Duplicate'(int), how it shown in the Fig.11. The table can have other columns but these three columns must exist

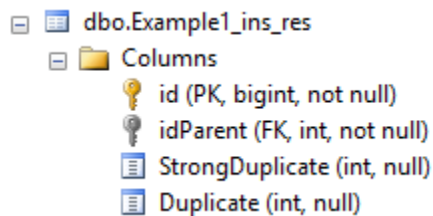


Fig.11 The web application project inside Visual Studio

In initial stage project also loads settings. They defined into 'Properties.Settings'. Settings define which DME projects is going to be used, where find the table for results, etc. Settings are shown in the Fig.12

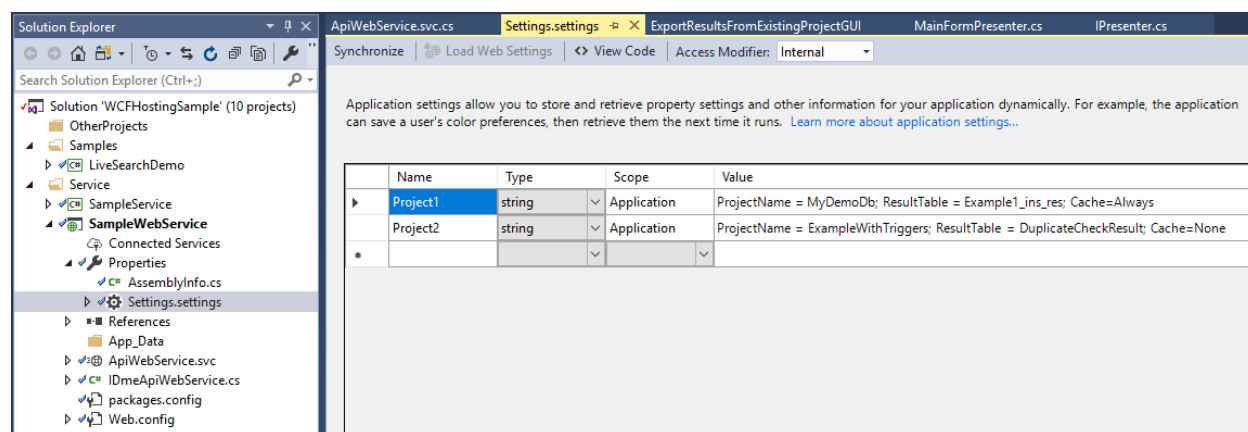


Fig. 12. Settings of Web Application project

The application can work with a few data sources in same time. Every string of settings defines one data source. The string must have such structure:

"ProjectName = MyDemoDb; ResultTable = Example1_ins_res; Cache=Always"

ProjectName – name of DME project, where the data source, matching fields are defined; **ResultTable** – the table for results, it is described above; **Cache** – how the application should add checked records into the table with cached data source after checking.

Cache can take one of the next value: *Always*, *ExceptStrongDuplicates*, *ExceptDuplicates*, *None*. *Always* – every record will be added, *ExceptStrongDuplicates* – record that doesn't have strong duplicates will be added, *ExceptDuplicates* – record that don't have strong duplicates and don't have duplicates will be added, *None* – record will not be added.

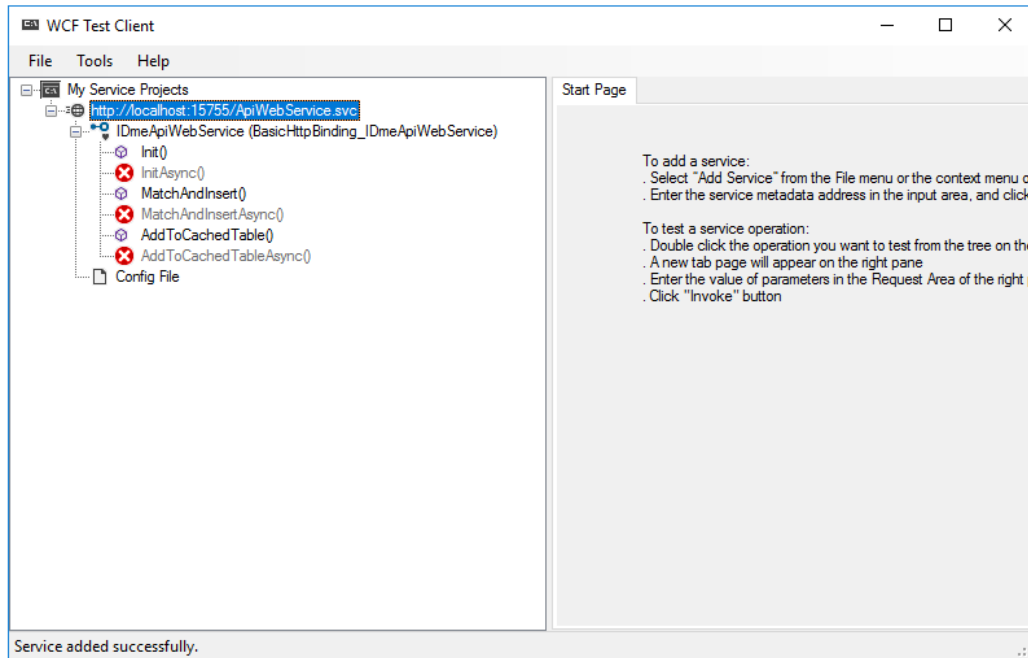


Fig. 13. Web Service after starting in Wcf Test Client

There are two ways to add new record in cache table inside Match Engine. The first automatically after matching. The second using web service method 'AddToCacheTable' (see Fig.13), this way can be used for inserting such records straight from the database.

After starting service all existing methods of service shown into 'WCF Test Client' (Fig. 13). Also, service can be connected with web browser (Fig.14)

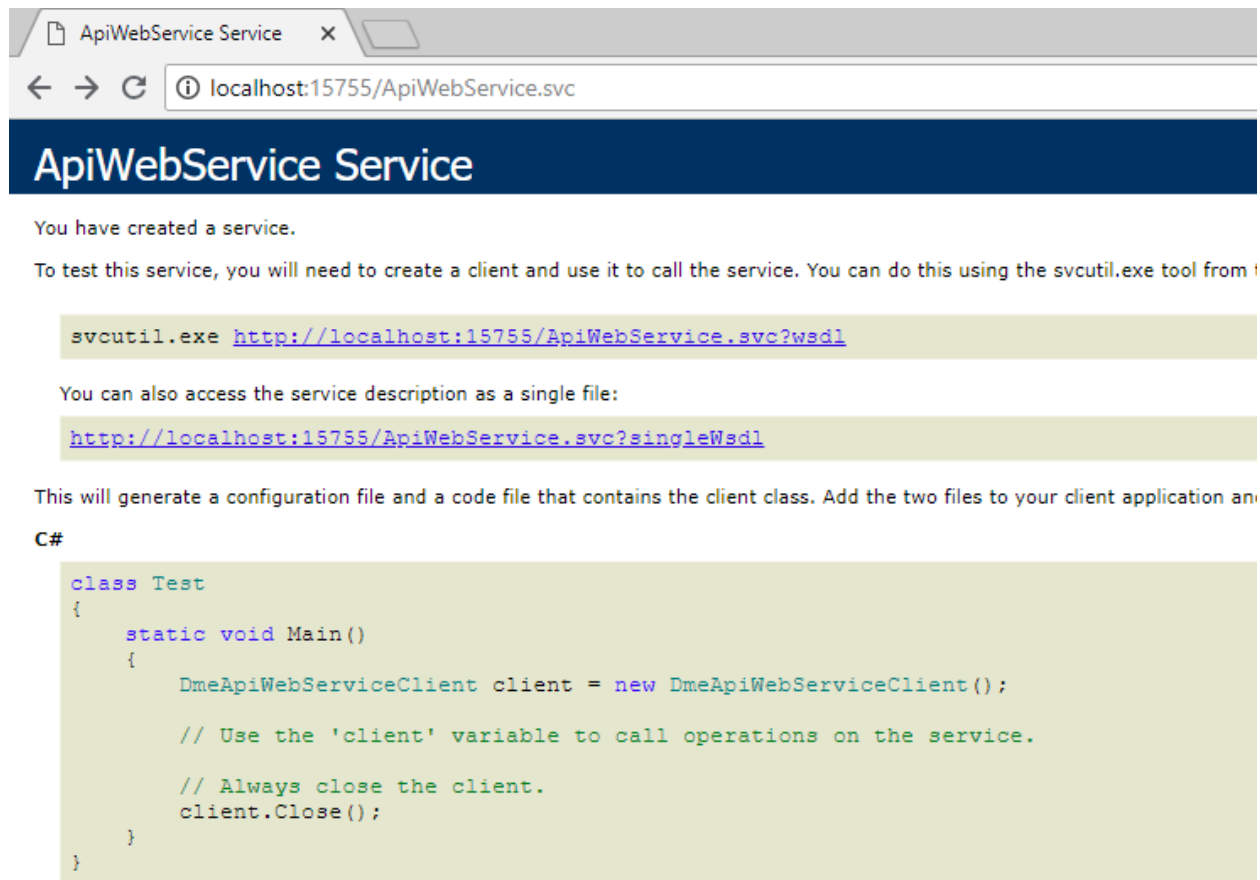


Fig. 14. Web Service in browser

Library for accessing Web Service from SQL Server

For accessing to web service from SQL Server, the application uses the assembly that loaded in the database that can interact with SOAP. How you can find it in Visual Studio Solution Explorer has shown on Fig.15a

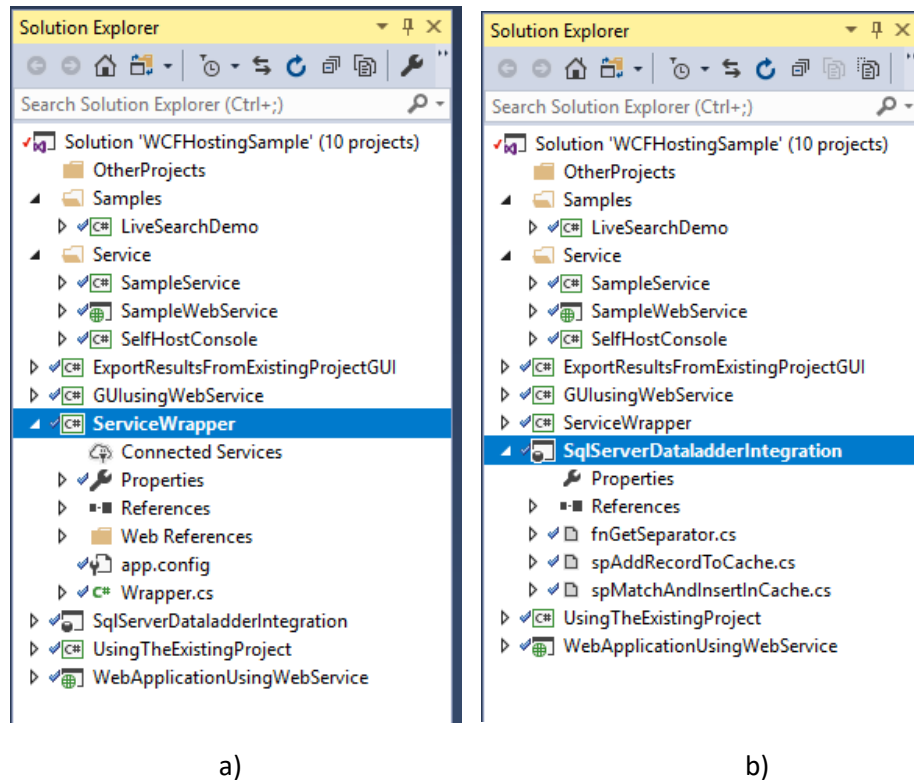


Fig. 15. Assemblies that interact with SOAP from SQL Server

This project has a Service Reference to the project described in previous.

SQL CLR library for SQL Server

This assembly connect C# and SQL languages. It is shown on Fig.15b. Stored procedures wraps SOAP calls in this project.

Deploying

SQL scripts describe below for allow loading assemblies and creating stored procedures.

The database name and the path to assemblies should be replaced!

- Load assemblies described in p.2 into SQL Server database

Enable unsafe assembly in database

```
alter database [DatabaseName]
set trustworthy on;
go
```

Enable CLR assemblies

```
USE [DatabaseName]
EXEC sp_configure 'clr enabled', '1';
RECONFIGURE;
```

Load assemblies into database

```
IF EXISTS (
    SELECT [name]
    FROM sys.assemblies
    WHERE [name] = N'SqlServerDataladderIntegration')

BEGIN
    DROP ASSEMBLY SqlServerDataladderIntegration
    ALTER ASSEMBLY SqlServerDataladderIntegration
    FROM 'D:\ApiDemo\SqlServerDataladderIntegration.dll'
    WITH PERMISSION_SET = UNSAFE ;
END
ELSE
BEGIN
    CREATE ASSEMBLY SqlServerDataladderIntegration
    FROM 'D:\ ApiDemo \SqlServerDataladderIntegration.dll'
    WITH PERMISSION_SET = UNSAFE ;
END
```

- Create Stored Procedures that allow use loaded assemblies from triggers and other objects of database

Use DatabaseName
Go

```
IF EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[spMatchInsert]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
DROP PROCEDURE [dbo].[spMatchInsert]
GO
```

```
IF EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[spAddToCacheTable]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)
DROP PROCEDURE [dbo].[spAddToCacheTable]
GO
```

```
IF EXISTS (select * from dbo.sysobjects where id =
object_id(N'[dbo].[spGetSeparator]') and OBJECTPROPERTY(id, N'IsScalarFunction') = 1)
DROP FUNCTION [dbo].[spGetSeparator]
GO
```

```
IF EXISTS (select * from dbo.sysobjects where id = object_id(N'[dbo].[spGetVersion]')
and OBJECTPROPERTY(id, N'IsProcedure') = 1)
DROP PROCEDURE [dbo].[spGetVersion]
GO
```

```
CREATE PROCEDURE [dbo].[spMatchInsert]
    @parameter1 nvarchar(max),
    @parameter2 bigint,
    @parameter3 nvarchar(max)
WITH EXECUTE AS CALLER
AS
```

```
EXTERNAL NAME SqlServerDataladderIntegration.StoredProcedures.spMatchAndInsertInCache
Go
```

```
CREATE PROCEDURE [dbo].spAddToCacheTable
    @parameter1 nvarchar(max),
    @parameter2 nvarchar(max)
WITH EXECUTE AS CALLER
AS
EXTERNAL NAME SqlServerDataladderIntegration.StoredProcedures.spAddRecordToCache
Go
```

```
CREATE FUNCTION [dbo].[spGetSeparator]()
RETURNS
    nvarchar(max)
WITH EXECUTE AS CALLER
AS
EXTERNAL NAME SqlServerDataladderIntegration.UserDefinedFunctions.spGetSeparator
Go
```

```
CREATE PROCEDURE [dbo].spGetVersion
WITH EXECUTE AS CALLER
AS
EXTERNAL NAME SqlServerDataladderIntegration.StoredProcedures.spGetVersion
Go
```

- View of Management Studio after executing scripts must be such it shown on Fig.16

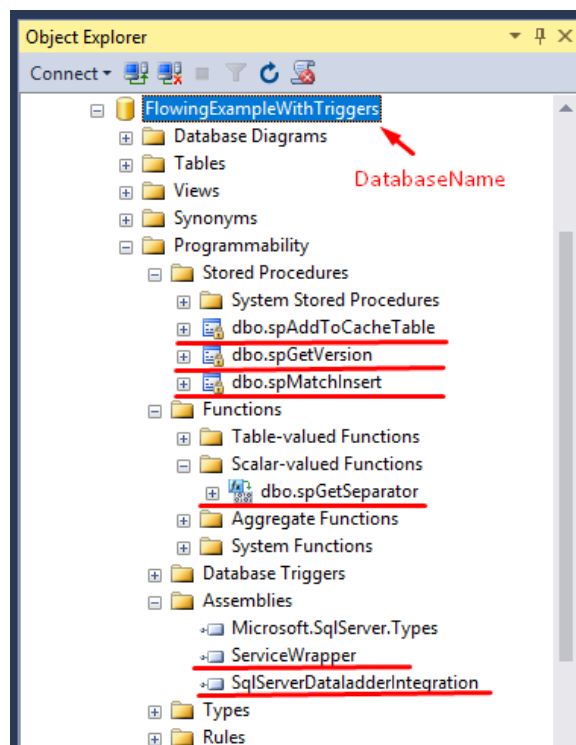


Fig. 16. Procedures and assemblies in the database

Check installation

Run Sample Web Service project in Visual Studio. Execute next query for checking installation:

```
EXEC dbo.spGetVersion
```

Expected result (the first number is a version of DME API, the second is a version of API Demo):

```
3.0.26.0 (1.0.4.0)
```

Example of using

Example of using is shown on Fig.17.

It has three tables Holding Table, DuplicateCheckResults Table, MasterTable. There are triggers that attached to every table.

User inserts automatically or manually records into Holding table. Holding table has attached trigger that creates for every inserted record a new row in Duplicate Check table and sends inserted record to matching service. Matching Web service searches duplicates and writes results into the row created by Holding Table trigger. Every inserting or updating of Duplicate Check Table fires the attached trigger. This trigger decides to copy new record in Master table or not. After inserting in Master Table, the trigger that attached to this table sends this record to Matching service for added to cached data.

Settings for this example are shown on Fig.12, as "Project2"

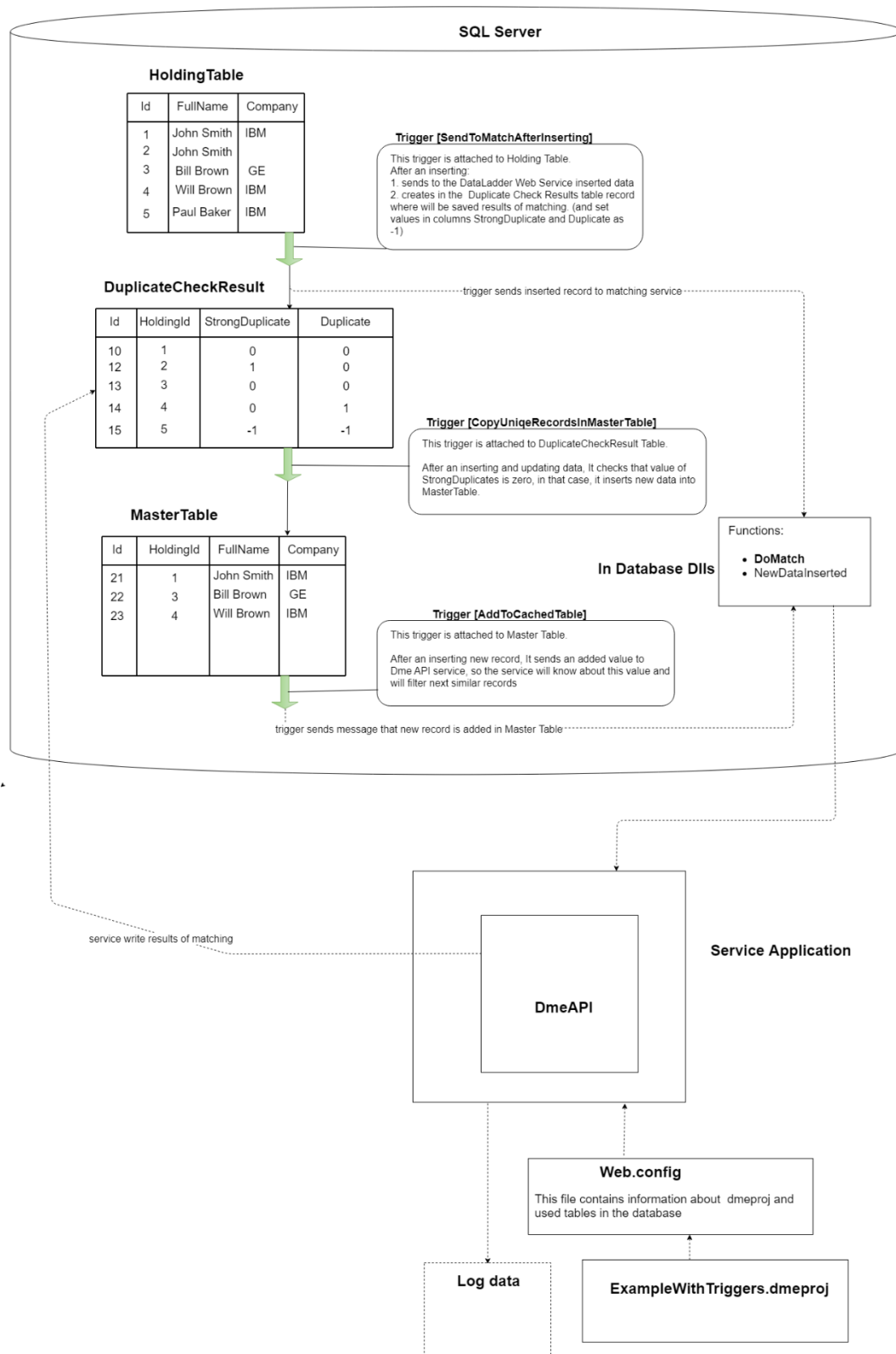


Fig. 17. Example of using

Scripts for creating triggers

SQL scripts for creating triggers are shown bellow

- Trigger for Holding Table

```
USE [DatabaseName]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [dbo].[SendToMatchAfterInserting]
    ON [dbo].[HoldingTable]
    AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @id INT
    DECLARE @State nvarchar(max)
    DECLARE @ContactName nvarchar(max)

    SELECT @id=Id, @State=[State], @ContactName=[ContactName]
        FROM    INSERTED;

    INSERT INTO DuplicateCheckResult
        (HoldingId, StrongDuplicate, Duplicate)
        values (@id, -1, -1);

    DECLARE @idResult bigint;
    SELECT @idResult=id FROM DuplicateCheckResult WHERE HoldingId=@id;

    DECLARE @separator nvarchar(max);
    EXEC @separator = spGetSeparator;

    DECLARE @mergedstring nvarchar(max);
    SET @mergedstring=@State + @separator + @ContactName

    EXEC dbo.spMatchInsert 'ExampleWithTriggers', @idResult, @mergedstring;

END
GO

ALTER TABLE [dbo].[HoldingTable] ENABLE TRIGGER [SendToMatchAfterInserting]
GO
```

- Trigger for Duplicate Check Table

```
USE [DatabaseName]
GO
SET ANSI_NULLS ON
```

```

GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [dbo].[CopyUniqueRecordsInMasterTable]
    ON [dbo].[DuplicateCheckResult]
    AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    Declare @HoldingId int
    Declare @Strong int
    Declare @Duplicates int

    SELECT @HoldingId=HoldingId, @Strong=StrongDuplicate, @Duplicates=Duplicate
    FROM INSERTED;

    if (@Strong IS NOT NULL)AND(@Duplicates IS NOT NULL)
    Begin
        IF (@Strong = 0)
        Begin
            DECLARE @id int;
            DECLARE @City nvarchar(max);
            DECLARE @Zip nvarchar(max);
            DECLARE @CompanyName nvarchar(max);
            DECLARE @Industry nvarchar(max);
            DECLARE @Title nvarchar(max);
            DECLARE @State nvarchar(max);
            DECLARE @ContactName nvarchar(max);
            DECLARE @Address nvarchar(max);

            SELECT @id=Id, @City=City, @Zip=Zip, @CompanyName=CompanyName,
                @Industry=Industry, @Title=Title, @State=[State],
                @ContactName=ContactName, @Address=[Address]
            FROM HoldingTable WHERE id=@HoldingId;

            INSERT INTO MasterTable
                (HoldingId, City, Zip, CompanyName, Industry, Title, [State],
                ContactName, [Address] )
            values
                (@id, @City, @Zip, @CompanyName, @Industry, @Title, @State,
                @ContactName, @Address);
        End
    End
END
GO
ALTER TABLE [dbo].[DuplicateCheckResult] ENABLE TRIGGER
[CopyUniqueRecordsInMasterTable]
GO

```

- Trigger for Master Table

```

USE [DatabaseName]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

```

```

CREATE TRIGGER [dbo].[AddToCachedTable]
ON [dbo].[MasterTable]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @id int;
    DECLARE @City nvarchar(max);
    DECLARE @Zip nvarchar(max);
    DECLARE @CompanyName nvarchar(max);
    DECLARE @Industry nvarchar(max);
    DECLARE @Title nvarchar(max);
    DECLARE @State nvarchar(max);
    DECLARE @ContactName nvarchar(max);
    DECLARE @Address nvarchar(max);

    SELECT @id=Id, @City=City, @Zip=Zip, @CompanyName=CompanyName,
           @Industry=Industry, @Title=Title, @State=[State], @ContactName=ContactName,
           @Address=[Address]
    FROM INSERTED

    DECLARE @separator nvarchar(max);
    EXEC @separator = spGetSeparator;

    DECLARE @mergedstring nvarchar(max);
    SET @mergedstring=@State + @separator + @ContactName

    EXEC dbo.spAddToCacheTable 'ExampleWithTriggers', @mergedstring;
END
GO
ALTER TABLE [dbo].[MasterTable] ENABLE TRIGGER [AddToCachedTable]
GO

```

The ‘SQL Scripts -Archive of Useful scripts.zip’ archive contains all these SQL scripts

The TriggerDemoGui project

This project checks the correct work of the Trigger Demo sample. All data, names of columns in this project are hardcoded and adjusted for using with for quick installation that is described in ‘Installation and Start’ section. If you want to adjust to your data sources you can do it only change the source code of the project. It’s possible to do because the size of the project is not big.

The view of this project application is shown in Fig.8. Users can enter data. And this data will be inserted into HoldingTable shown in Fig.17. If everything adjusted correctly all interactions shown in Fig.17 will be executed automatically. This project periodically requires data from the database tables and analyzes answers. It means, while tasks from Fig.17 are executing this project gets the freshest updates of tables and shows progress for users. Users can see what happens now.

This project should be used carefully because the central part of this project is polling database tables. If these tables have a big size the project will require increasing of resources.