Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

Integracja systemów

Laboratorium 8
Prowadzący: Marek Kowal
(M.Kowal@issi.uz.zgora.pl)

Cleansing Data with the Script Component

Sometimes you can't accomplish your data cleansing goal in a **Derived Column Transform**, and you must get more advanced. Say, for example, you want to run a routine where any character data is removed from the data, or, if the data coming in is an invalid date, perhaps you want to replace it with today's date. In these examples, you can use a Script Component in the Data Flow Task. The Script Component can play one of three roles: transform, source, or destination:

- **Transform**—Generally, the focus of your Data Flow will be on using the script as a transform. In this role, you can perform advanced cleansing with the out-of-the-box components.

- **Source**—When the script is used as a source, you can apply advanced business rules to your data as it's being pulled out of the source system. (This happens sometimes with COBOL files.)

- **Destination**—When the script is used as a destination, you can use the script to write out to a non-OLE DB destination, like XML or SharePoint.

You can write your script in VB.NET or C#, but once you select a language, you can't change it. You can select the language by double-clicking the **Script Component** and going to the Script page of the **Script Transformation Editor**. You can also select any variables you want to pass into the script in this page. Make sure to select the variable for **ReadWrite** only if the variable needs to be written to. Otherwise, the variable will be locked for the duration of the script's execution.

On the Input Columns page, select each column that you want to be passed into the script from the Data Flow and select whether you want to allow them to be accessed for writing (see Fig. 1). If you don't need the column for writing, make sure it's set to **ReadOnly**, because **ReadWrite** columns require more resources. All columns that are not checked are passed through to the next transform or destination seamlessly. You can also add more columns that are not part of the source or a previous transform using the Inputs and Outputs page. This page enables you to add other buckets of data that you can use to direct the data down multiple paths. To do this, you must first create an additional output by clicking **New Output**. Then you need to set the **SynchronousInputID** property to the same number for each output. Set the **ExclusionGroup** to the same non-zero number. In the script,

you can then use the **DirectRowTo<outputbuffername>** method to send the data to each of the paths. Because this is in the script, the data can be sent to multiple paths at the same time.

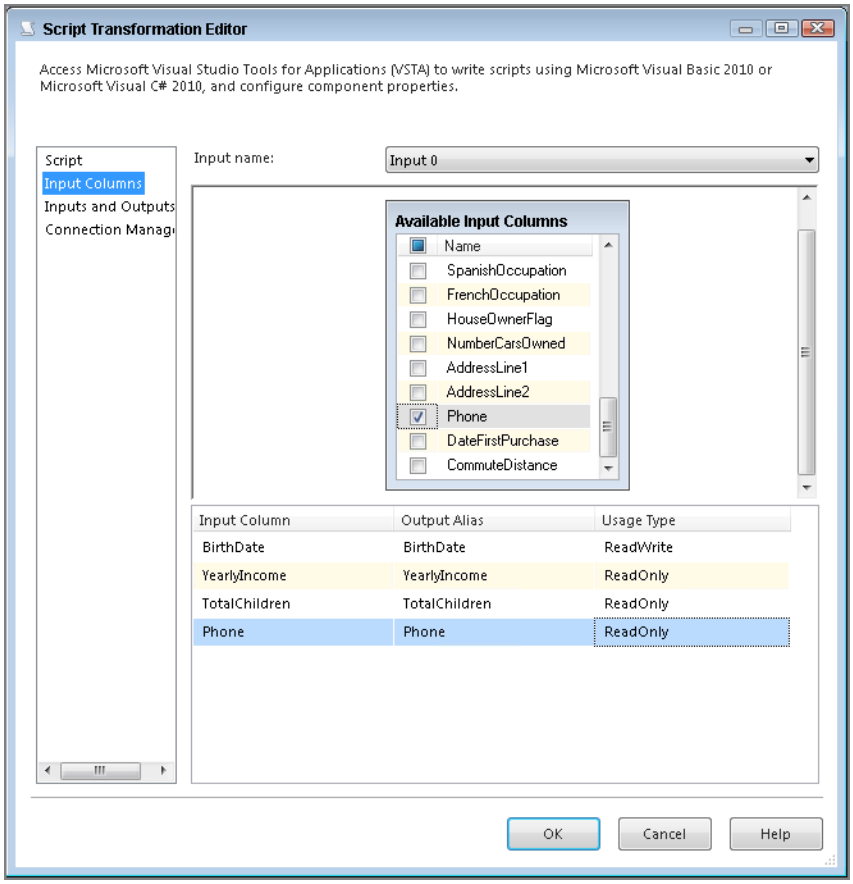

Fig. 1

To edit the script, go to the Script page and click **Edit Script**. This opens the Visual Studio environment. Three subroutines are the most important to your design: **PreExecute, PostExecute,** and **ProcessInputRow**:

- **PreExecute** executes once per transform execution and is a great place to initialize objects or connections that you hope to use later.
- **PostExecute** is where you can close connections and objects or set variables.
- **ProcessInputRow** is run for every row going through the transform; from this subroutine you cannot set variables.

Accessing a row from the **ProcessInputRow** subroutine is simple. To do so, you must use the Row object, which contains an individual row as it is looping. For example, to read a row coming into the transform, like BRIAN KNIGHT, and translate that to a proper-cased value, like Brian Knight, use the following code, where `ColumnName` holds the customer name. `StrConv` is a string conversion function to convert a string to a new format.

```
Public Overrides Sub Input0_ProcessInputRow(ByVal Row As
Input0Buffer)
      'This is the line that performs the magic to Proper Case.
            Row.ColumnName = StrConv(Row.ColumnName,
            VbStrConv.ProperCase)
                  End Sub
```

Variables can be read from any subroutine, but you will only be able to write to them in the **PostExecute** subroutine. To read or write to a variable, you can use a `Me.Variables` statement, as shown in the following

```
Row.YearlyIncome = Row.YearlyIncome + Me.Variables.PremiumIncome
```

1. **Cel ćwiczenia**

   You have recently begun to receive data from an entity that has occasional issues with date data. The source application allows users to enter whatever they'd like for the birth date, so occasionally you receive invalid characters in the date or invalid dates. After completing this lesson, you'll have a better idea of how to use the Script Component to perform more complex cleansing or checking of your data..

2. **Przebieg ćwiczenia**

   1) Create a new package

   2) Create a connection to the file called **Lesson27Data.txt**.

   3) Create a new **Flat File Connection Manager** called **Extract** and point to the Lesson27Data.txt file. In the **General page**, check the Column names in the first data row box and ensure that the **EmailAddress** column in the **Flat File Connection Manager** is set to 100 characters in the **Advanced page**.

   4) Create a **Data Flow Task**. In the Data Flow tab, create a **Flat File Source** that points to the new connection manager that you just created in Step 3 called **Extract**.

   5) Drag a **Script Component** onto the design pane. You are immediately prompted for what type of script you want to use (source, transform, or destination). Select **Transformation** for the type and connect the transform to the **Flat File Source**.

   6) In the **Script Transform Editor**, select the **Input Columns** page and check **BirthDate** and **DateFirstPurchase**. Ensure that **DateFirstPurchase** is set to **ReadWrite**.

   7) Go to the **Inputs** and **Outputs** page and highlight the **Output 0 Buffer** and click **Remove Output**, found on the bottom of the window. Then click **Add Output** twice. Rename the first output you just created **BadData** and the second to **ValidatedData**.

For both of the outputs, set the **SynchronousInputID** to the same input buffer and set the **ExclusionGroup** property to 1, as shown in Fig. 2.

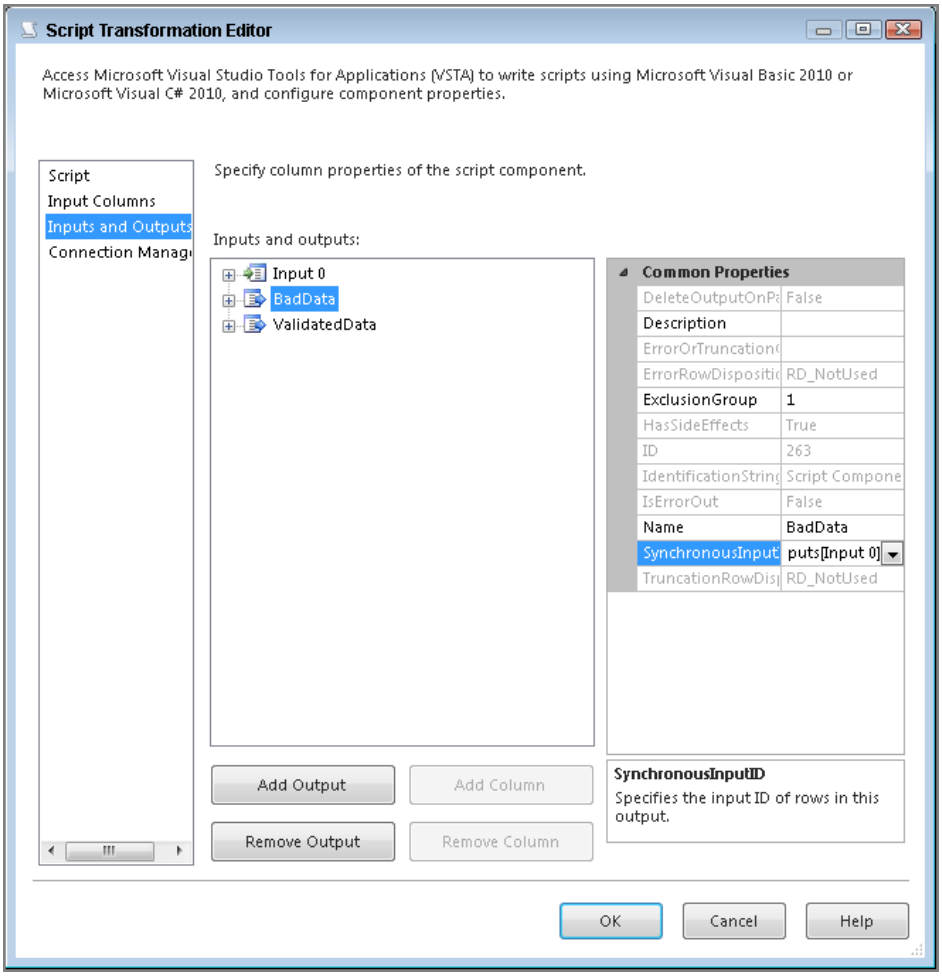

Fig. 2

8) Go to the Script page, select Microsoft Visual Basic 2010 for the **ScriptLanguage**, and click **Edit Script**. Then add the following script in the **ProcessInputRow** subroutine (note that the subroutine will already exist in your code block):

```
Public Overrides Sub Input0_ProcessInputRow(ByVal Row As
Input0Buffer)
      If IsDate(Row.DateFirstPurchase) = False Then
            Row.DateFirstPurchase = Now
      End If
      If IsDate(Row.BirthDate) = True Then
            Row.DirectRowToValidatedData()
      Else
            Row.DirectRowToBadData()
      End If
End Sub
```

9) Ensure there is nothing underlined blue (showing bad code), then close the script and return to the designer.

10) Drag two **Union All** Transforms onto the design pane and connect the Script Transform to each of the Union All Transforms.

11) Execute the package, and you should see five bad rows go down the **BadData** path, as shown in Fig. 3.

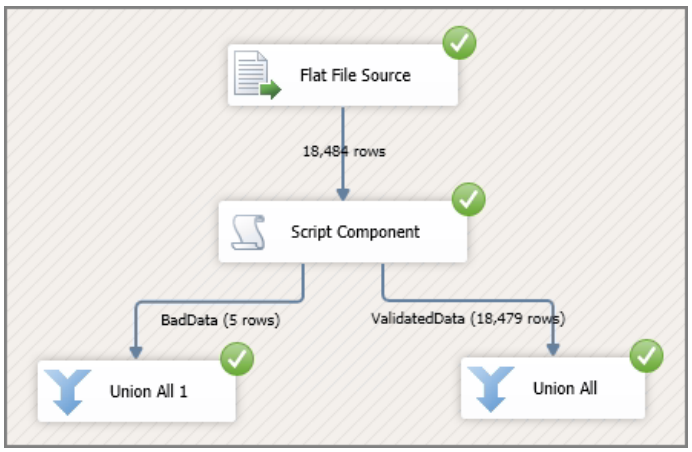

Fig. 3

Separating Data with the Conditional Split Transform

Sometimes you deal with source data that may require different treatments applied to it. For example, you want to generate a mailing list for a direct mail campaign, but you want to target only customers with children. You want to make sure to separate the customers without kids before preparing the list. You would also like anyone who has more than five kids to receive a buy-two-get-one-free coupon with the mailer.

The best way to separate data within a package to apply different types of actions is with the **Conditional Split Transform**. With this transform, you can send data from a single data path to multiple outputs based on conditions set in the **Conditional Split Transformation Editor**, shown in Fig. 4. To open the editor, drag the transform in the design surface and doubleclick it.

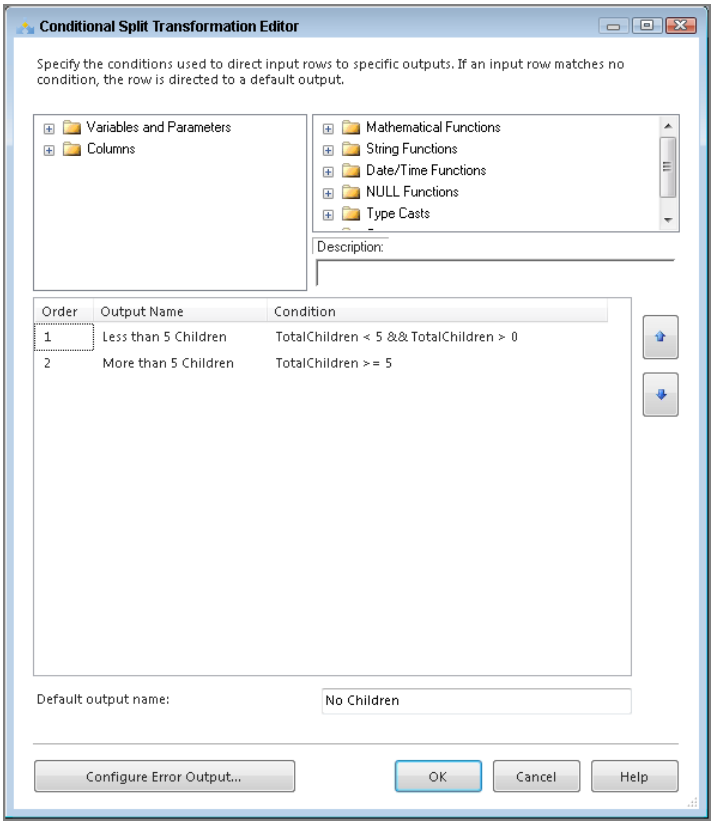

Fig. 4

The **Conditional Split Transform** uses the SSIS expression language to determine how the data pipeline should be split. For this example, all you need to know is that the Conditional Split Transform is checking to see if customers have more than five kids so they can receive the extra coupon. This check produces three possible outputs:

- For customers with more than five children
- For customers with between one and four children
- For customers with no children

It may look like you have only two outputs, but if you look on the bottom of the Conditional **Split Transformation Editor**, the **Default Output Name** provides an output for data that

doesn't apply to the conditions declared. In the case of this package, you need only those customers with at least one child; you will see only these outputs in the final package. You do not need to use the output for customers with no children.

1. **Cel ćwiczenia**

   Your company needs a list of customers for a direct mail campaign that is only going to be sent regionally. You need to create an SSIS package that generates two different mailing lists because one region is going to receive a different promotion than the other.

2. **Przebieg ćwiczenia**

1) Create a new package and add a **Data Flow Task** named **DFT - Regional Mailing List** to the **Control Flow** design surface.

2) Create a new **OLE DB Connection Manager** using the **AdventureWorksDW2014** database as the source. Then drag an **OLE DB Source** on the designer and rename it **Customer Source**.

3) In **Customer Source**, select **AdventureWorksDW2014** as the connection manager and **SQL Command** as the **Data access mode**.

4) Enter the following query in the **Data access mode** window:

```
Select FirstName,
MiddleName, LastName, AddressLine1, AddressLine2, EmailAddress,
Phone, GeographyKey
From DimCustomer
```

5) Drag a **Lookup Transform** on to the design pane and name it **LKP - Geography**. Open the **Lookup Transformation Editor** and select **AdventureWorksDW2014** as the connection manager.

6) Next, select Use results of an SQL query and use the following query:

```
SELECT GeographyKey, StateProvinceCode
FROM DimGeography
```

7) Go to the **Columns tab** to add the **StateProvinceCode** to the data stream shown in Fig. 5.

8) Now bring a **Conditional Split Transform** to the design surface and connect it to the **Lookup Transform**. When prompted, select **Lookup Match Output** for the Output of the Lookup Transform.

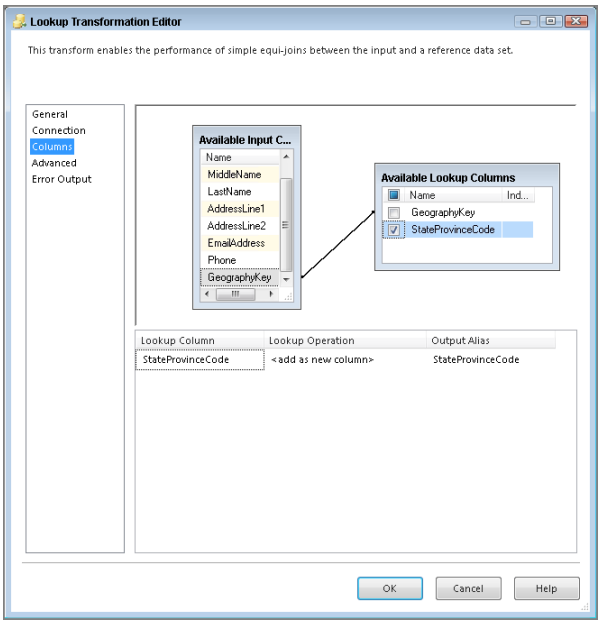

Fig. 5

9) Open the **Conditional Split Transformation Editor**. Add a new output in the **Conditional Split Transformation Editor** called **Campaign1**, and then add the following condition:

```
StateProvinceCode == "FL" || StateProvinceCode == "GA"
```

10) Add a second output named **Campaign2** with the following condition:

```
StateProvinceCode == "CA" || StateProvinceCode == "WA"
```

11) Make **No Ad Campaign** the **Default Output Name** and click **OK**. After making these changes, the editor should look like Fig. 6.

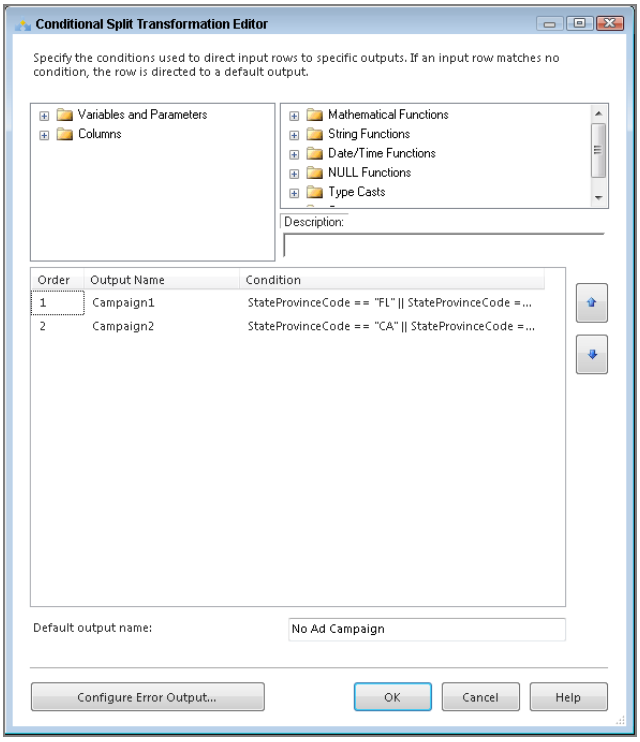

Fig. 6

12) Bring two **Flat File Destinations** into the Data Flow and name them **Campaign1 Mailing List** and **Campaign2 Mailing List**. Create separate connection managers for them pointing to the file location of your choice.

13) The **Conditional Split** will have three blue output arrows. When you connect the first blue arrow to one of the two destinations, a dialog box opens asking which output you want. Connect each output to the destination that has the name associated with it. This action leaves one output unused: **No Ad Campaign**.

14) Open each **Flat File Destination** to make sure the mapping is set correctly.

15) The package is now complete, you can run them.

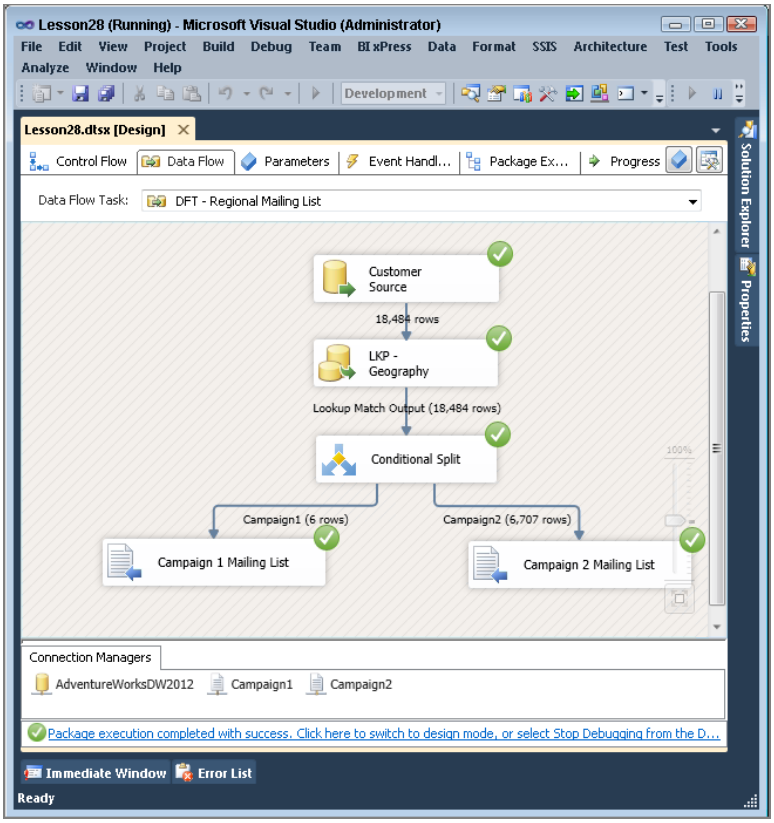

Fig. 7

Altering Rows with the OLE DB Command Transform

The **OLE DB Command Transform** is used to run a SQL statement for each row in the **Data Flow**. This involves kicking off an update, insert, or delete statement for each row in an input stream. To put this into perspective, imagine you are loading a product dimension table in your ETL process. Your predecessor decided it would be best to update and delete these rows using an **OLE DB Command**. The company you work for is a major department store, and the new spring clothing line is coming in. So, all the winter clothes are being marked down. This means you are going to get an update with a price reduction for all the winter clothes your company has in inventory at one time. Using the **OLE DB Command Transform** would mean that your package would be running several thousand update statements and your package would run for hours. A situation like that one is why we recommend you avoid using the **OLE DB Command Transform**. This doesn't mean you should never use this transform, but it is important to understand its shortcomings when working with large amounts of data.

To use the **OLE DB Command Transform**, drag it from the Toolbox to the Data Flow design surface and double-click it. The configuration looks more complicated than it really is. From the **Connection Managers tab**, specify which **OLE DB Connection** you want to execute the SQL statement on.

You set the SQL statement you plan to execute on the **Component Properties tab**. To enter your SQL statement, click the ellipsis next to the **SqlCommand** property. Remember that to tell SSIS that you are going to be using parameters in a SQL statement, you use a question mark (?).

You can also configure the amount of time before a timeout occurs in the **CommandTimeout** property. This uses an interval of seconds where 0 denotes no timeout.

The **Column Mappings** tab in the **Advanced Editor** for **OLE DB Command** window is similar to the **Mappings page** in a destination editor. It displays the input stream and destination columns, which are really the parameters indicated in the **SqlCommand** property. Any input column mapped to a parameter replaces the parameter with the value of that field. When you are mapping, remember that the order in which you place the parameters while writing the SQL statement is also the order in which they must be mapped. In Fig. 8 you see how to map the following Update statement:

```
Update Production.TransactionHistory
Set ModifiedDate = ?
Where ProductID = ?
```

The last tab is the **Input and Output Properties** tab, which you will likely not ever have to change; it simply provides another place where you can add or remove columns that are used in the transform.

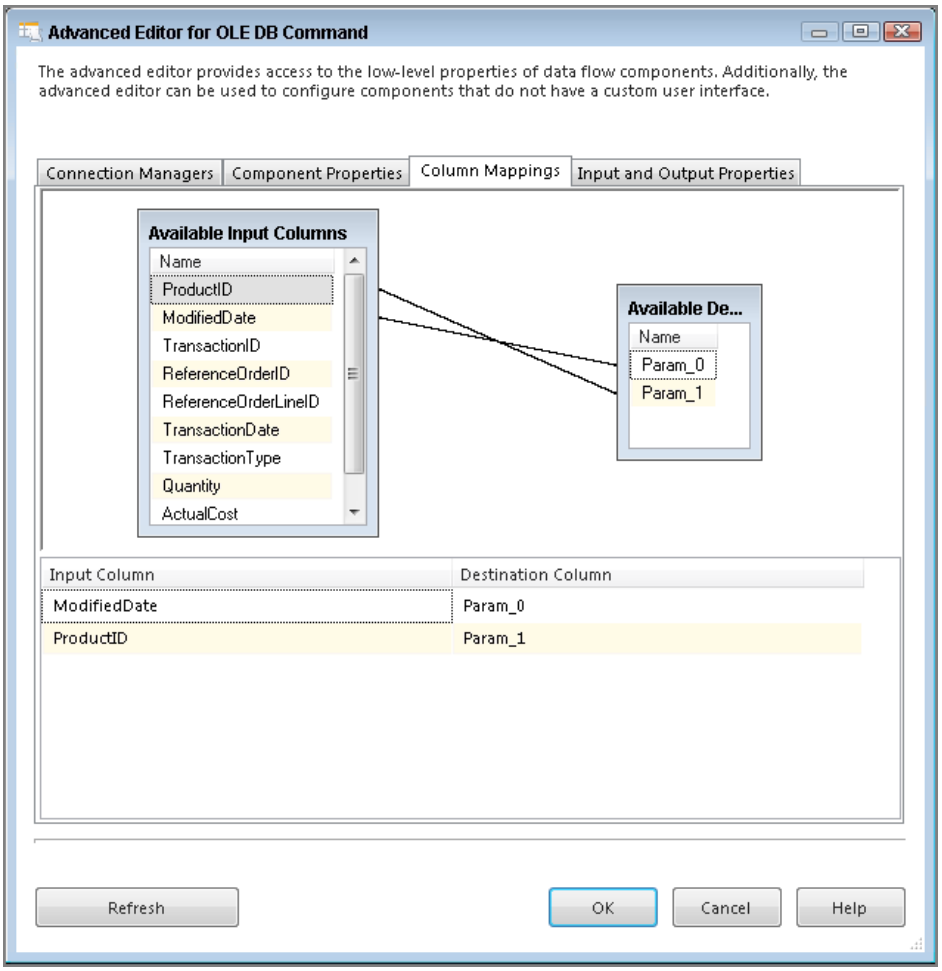

Fig. 8

1. **Cel ćwiczenia**

   You work for a company that sells dartboard supplies. As new supplies are added to your inventory, some of the older products are being discounted. Use the flat file extract provided and update the price on all required products. After completing this lesson, you will know how to use the **OLE DB Command Transform** to alter data with a SQL statement inside the **Data Flow**

   NOTE: The small package created in this exercise is meant only to show the capabilities of the **OLE DB Command Transform**. Our recommendations stated earlier in the lesson for why you might want to avoid using the **OLE DB Command Transform** for these sorts of situations still stand.

2. **Przebieg ćwiczenia**

   1) Create a new package

   2) Drag a **Data Flow Task** onto your designer and name it **DFT - OLE DB Command**.

3) Create a new **Flat File Connection Manager**, name it **Product Price Change**, and point it to **OLEDBCommandExample.txt**. Also, check the Column names in the first data row option. The editor should look like Fig. 9.

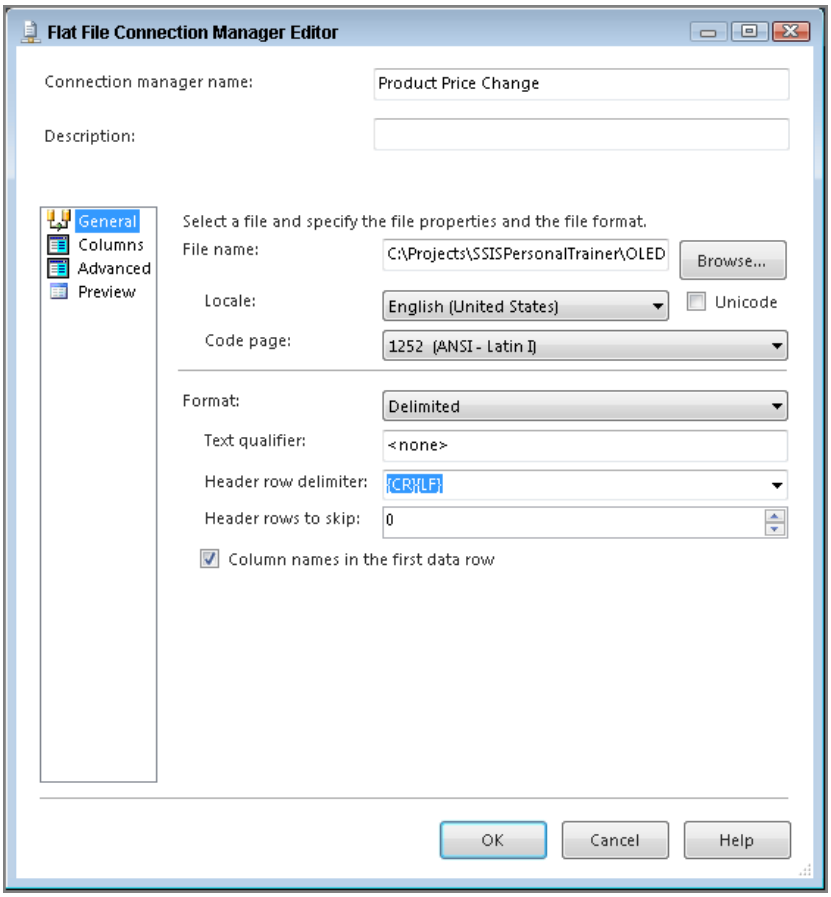

Fig. 9

4) In the **Data Flow**, bring a new **Flat File Source** over and name it **Discounted Products**. Open the editor and make the connection manager the newly created **Product Price Change**.

5) Open **Management Studio**, connect to the **AdventureWorks2014** database, and run the following query to create a new table called **Product_OLEDBCommand**

```
CREATE TABLE [dbo].[Product_OLEDBCommand](
[ProductID] [smallint] IDENTITY(1,1) NOT NULL,
[ProductBusinessKey] int,
[ProductName] [varchar](50) NOT NULL,
[ListPrice] [money],
[CurrentFlag] [smallint],
[RowStartDate] [datetime],
```

```
[RowEndDate] [datetime]
CONSTRAINT   [PK_Product_OLEDBCommand_ProductID]   PRIMARY   KEY
CLUSTERED
(
[ProductID] ASC
) ON [PRIMARY]
) ON [PRIMARY]
GO
INSERT INTO [dbo].[Product_OLEDBCommand] Select 101,
'Professional Dartboard','49.99', '1', '1/1/2006',Null
INSERT INTO [dbo].[Product_OLEDBCommand] Select 102,
'Professional Darts',15.99,1, '1/1/2006',Null
INSERT INTO [dbo].[Product_OLEDBCommand] Select 103,
'Scoreboard',26.99,1, '1/1/2006',Null
INSERT INTO [dbo].[Product_OLEDBCommand] Select 104,
'Beginner Dartboard',45.99,1, '1/1/2006',Null
INSERT INTO [dbo].[Product_OLEDBCommand] Select 105,
'Dart Tips',1.99,1, '1/1/2006',Null
INSERT INTO [dbo].[Product_OLEDBCommand] Select 106,
'Dart Shafts',7.99,1, '1/1/2006',Null
```

6) Next, create another connection manager, this time an **OLE DB Connection Manager**, using the **AdventureWorks2014** database.

7) Bring an **OLE DB Command Transform** onto the design surface, connect it to the source called **Discounted Products**, and after opening the transform's editor, select **AdventureWorks2014** as the connection manager on **the Connection Managers tab**.

8) Enter the following SQL statement in the **SqlCommand** property on the **Component Properties tab** (see Fig. 10)

```
Update Product_OLEDBCommand
Set CurrentFlag = 0,
RowEndDate = GETDATE()
Where ProductBusinessKey = ?
and RowEndDate is null
```

This statement means that for every **ProductBusinessKey** you have, the **CurrentFlag** will be set to 0, and the **RowEndDate** will be given today's date.

9) Next, on the **Column Mappings** tab you need to connect **ProductBusinessKey** from the **Available Input Columns** to **Param_0** in the destination. Fig. 11 shows there is only one parameter in this statement, so there is only one destination column.

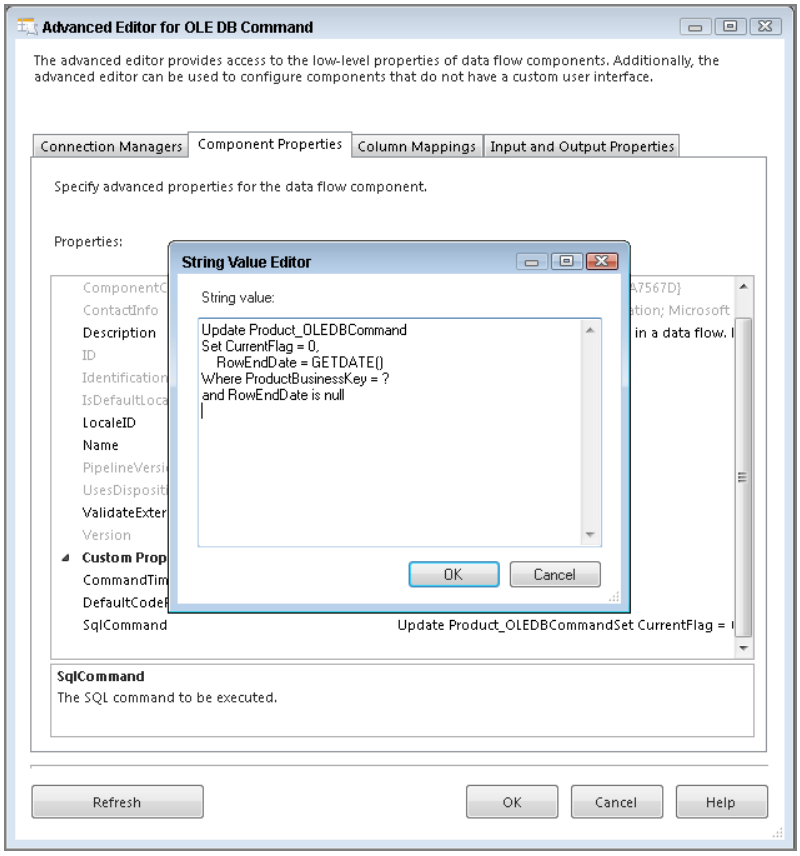
Advanced Editor for OLE DB Command
The advanced editor provides access to the low-level properties of data flow components. Additionally, the advanced editor can be used to configure components that do not have a custom user interface.
Connection Managers
Component Properties
Column Mappings
Input and Output Properties
Specify advanced properties for the data flow component.
Properties:
String Value Editor
String value:
Update Product_OLEDBCommand
Set CurrentFlag = 0,
RowEndDate = GETDATE()
Where ProductBusinessKey = ?
and RowEndDate is null
OK
Cancel
Description
LocaleID
Name
Version
Custom Prop
SqlCommand
SqlCommand
The SQL command to be executed.
Refresh
OK
Cancel
Help

Fig. 10

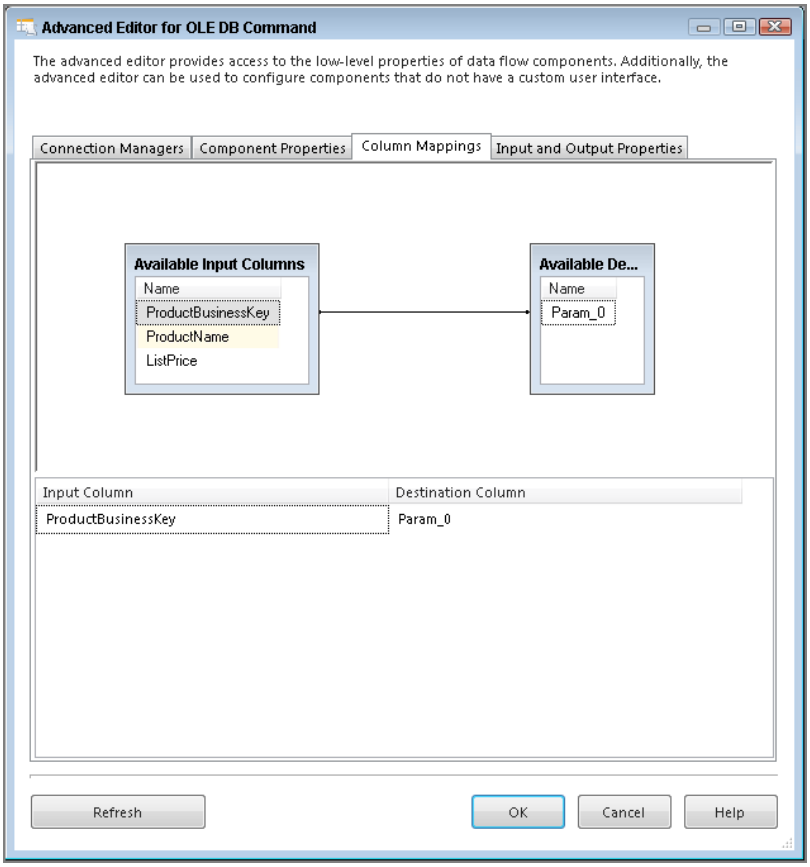
Advanced Editor for OLE DB Command
The advanced editor provides access to the low-level properties of data flow components. Additionally, the advanced editor can be used to configure components that do not have a custom user interface.
Connection Managers
Component Properties
Column Mappings
Input and Output Properties
Available Input Columns
Name
ProductBusinessKey
ProductName
ListPrice
Available De...
Name
Param_0
Input Column
Destination Column
ProductBusinessKey
Param_0
Refresh
OK
Cancel
Help

Fig. 11

10) Now bring a **Derived Column Transform** to the **Data Flow** and connect the **OLE DB Command** to it. Open the **Derived Column Transform Editor** and add two new columns called **RowStartDate** and **CurrentFlag**. For the **RowStartDate** column, use the **GETDATE()** function in the **Expression field**, and **CurrentFlag** just needs a 1 in the Expression box. The **Derived Column Transformation Editor** should look like Fig. 12. Click OK.

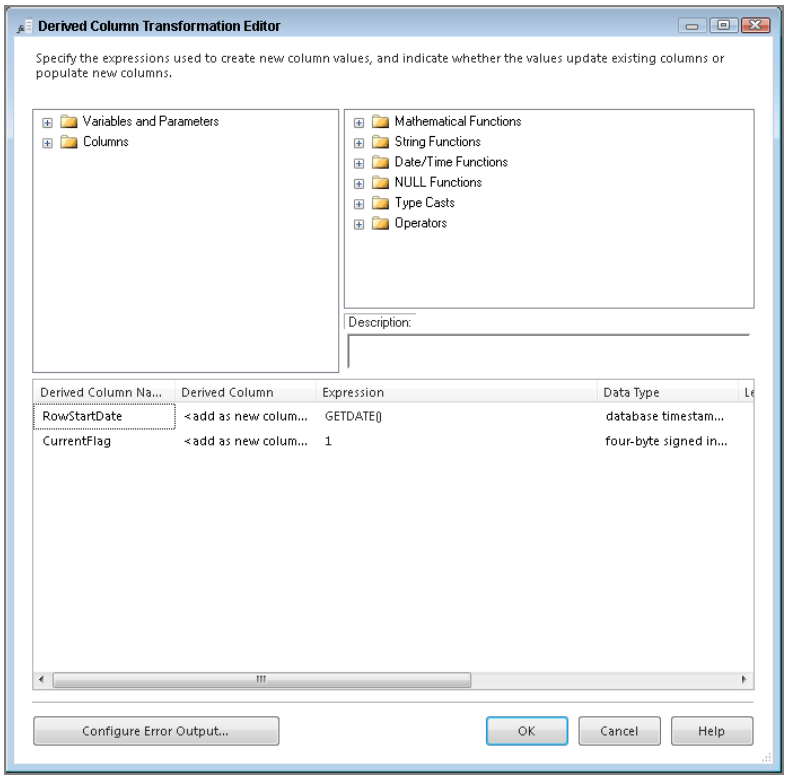

Fig. 12

11) To finish this package, you need to load the new rows' results into the **Product_ OLEDBCommand** table. Bring an **OLE DB Destination** onto the design surface, and from within the editor, select **Product_OLEDBCommand** as the destination table.

12) Go to the **Mappings page** of the **OLE DB Destination Editor**; notice how all the columns are automatically mapped except for **RowEndDate**, which is set in the **OLE DB Command Transform**. Fig. 13 shows how the final mapped columns should look.

13) A successful run of this package should look like Fig. 14.

14) Take a look at the table in Fig. 15 to see the results of a completed package. Notice that the package created a new row for each product with the new price. It also closed the old row by updating the row's end date and the current flag. This is what's known as a Type 2 change in a dimension table.

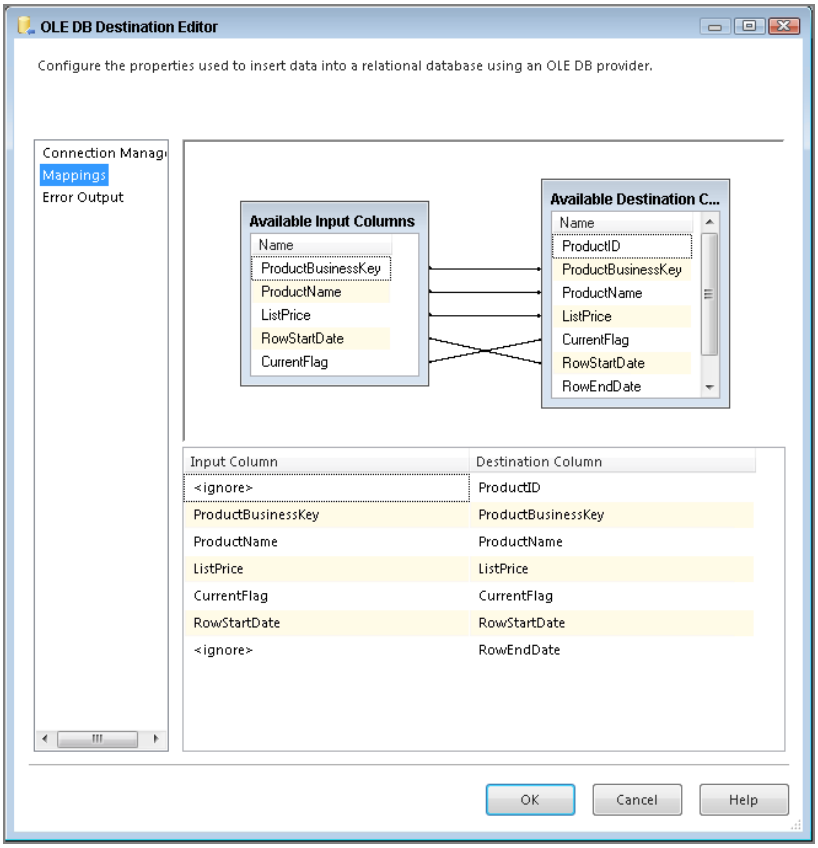
OLE DB Destination Editor
Configure the properties used to insert data into a relational database using an OLE DB provider.
Connection Manag
Mappings
Error Output
Available Input Columns
Name
ProductBusinessKey
ProductName
ListPrice
RowStartDate
CurrentFlag
Available Destination C...
Name
ProductID
ProductBusinessKey
ProductName
ListPrice
CurrentFlag
RowStartDate
RowEndDate
Input Column
Destination Column
<ignore>
ProductID
ProductBusinessKey
ProductBusinessKey
ProductName
ProductName
ListPrice
ListPrice
CurrentFlag
CurrentFlag
RowStartDate
RowStartDate
<ignore>
RowEndDate
OK
Cancel
Help

Fig. 13

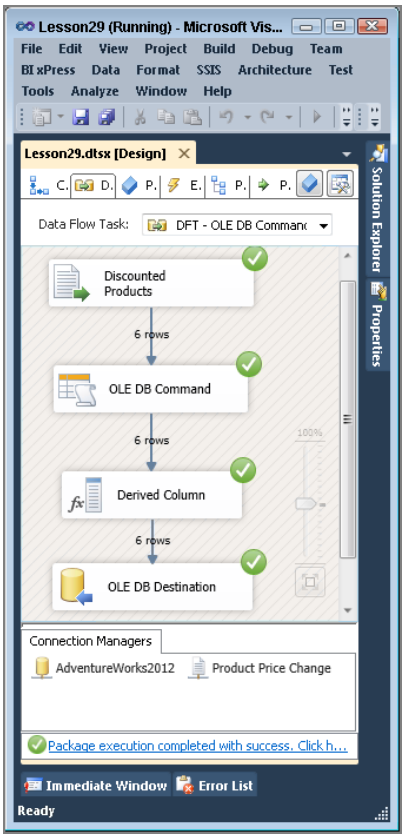
Lesson29 (Running) - Microsoft Vis...
File Edit View Project Build Debug Team
BIxPress Data Format SSIS Architecture Test
Tools Analyze Window Help
Lesson29.dtsx [Design]
Data Flow Task:
DFT - OLE DB Command
Discounted Products
6 rows
OLE DB Command
6 rows
Derived Column
6 rows
OLE DB Destination
Connection Managers
AdventureWorks2012
Product Price Change
Package execution completed with success. Click h...
Immediate Window
Error List
Ready
Solution Explorer
Properties

Fig. 14

	ProductID	ProductBusinessKey	ProductName	ListPrice	CurrentFlag	RowStartDate	RowEndDate
1	1	101	Professional Dartboard	49.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.890
2	2	102	Professional Darts	15.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.973
3	3	103	Scoreboard	26.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.973
4	4	104	Beginner Dartboard	45.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.977
5	5	105	Dart Tips	1.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.980
6	6	106	Dart Shafts	7.99	0	2006-01-01 00:00:00.000	2012-07-05 21:24:00.980
7	7	101	Professional Dartboard	44.99	1	2012-07-05 21:24:01.033	NULL
8	8	102	Professional Darts	11.99	1	2012-07-05 21:24:01.033	NULL
9	9	103	Scoreboard	17.99	1	2012-07-05 21:24:01.033	NULL
10	10	104	Beginner Dartboard	38.99	1	2012-07-05 21:24:01.033	NULL
11	11	105	Dart Tips	1.09	1	2012-07-05 21:24:01.033	NULL
12	12	106	Dart Shafts	4.99	1	2012-07-05 21:24:01.033	NULL

Fig. 15

Bibliografia

1) Knight B., Knight D., Davis M, Snyder W. (2013): Knight's Microsoft® SQL Server® 2012 Integration Services 24-Hour Trainer, John Wiley & Sons.
2) Knight B., Veerman E., Moss J.M., Davis M., Rock C. (2012): PROFESSIONAL Microsoft® SQL Server® 2012 Integration Services, John Wiley & Sons.
3) http://www.wrox.com/WileyCDA/Section/id-814197.html
4) https://msdn.microsoft.com/library/ms169917(SQL.120).aspx
5) Tok W-H., Parida R. Masson M. Ding X. Sivashanmugam (2012): Microsoft SQL Server 2012 Integration Services, Promise (tłumaczenie j. polski).
6) Kimball R. (2004): The Data Warehouse ETL Toolkit. John Wiley & Sons