db Software Laboratory

User manual

© DB Software Laboratory 2011

www.dbsoftlab.com

Contents

Contents ... 2
1. Introduction ... 5
2. Requirements ... 5
3. Key features ... 7
4. Visual Importer ETL Architecture ... 8
  5.1 Screen Overview ... 10
  5.2 Main tool bar ... 11
  5.3 Source tool bar ... 11
  5.4 Mapping panel ... 12
  5.5 SQL Statements ... 13
  5.6 Template tab ... 14
  5.7 Import Log Tab ... 14
  5.8 Rejected Records ... 15
  5.9 Import Process ... 16
  5.10 How to load data from a Flat File(s) ... 17
  5.11 How to perform Auto mapping ... 22
  5.12 How to load data from an ODBC Data Source ... 25
  5.15 How to load data from the Cross/Pivot table ... 31
  5.16 How to perform Calculations ... 33
  5.17 How to filter data ... 34
  5.18 Working with Date fields ... 36
  5.19 How to Update/Delete Records ... 37
  5.20 MS SQL Server specific parameters ... 39
  5.21 Oracle specific parameters ... 40
  5.22 ODBC specific parameters ... 42
    5.22.1 ODBC connection strings ... 44
  5.23 Error Handling ... 45
6. Scripting Language ... 46
  6.1 The Basic Structure ... 46
  6.2 Variables ... 47
  6.3 Logical Operations ... 52
  6.4 Repeating sets of commands ... 56
  6.5 Functions ... 59
  6.6 Script Examples ... 60
  6.7 Passing variables between objects ... 61
  6.7 Supported Functions List ... 62
    6.7.1 String Functions ... 62
    6.7.2 Conversion Functions ... 67
    6.7.3 File System Functions ... 69
    6.7.4 Date and Time Functions ... 73
    6.7.5 Numeric Functions ... 77
    6.7.5 Miscellaneous Functions ... 79
    6.7.6 Procedures ... 81
    6.7.7 Math functions ... 82
7. Date formats ... 87

Copyright © 2011 DB Software Laboratory

8. Command Line 88
9. Support Procedure 89
10. License Agreement 90

Copyright © 2011 DB Software Laboratory Limited. All rights reserved.

No portion of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, without prior written consent of DB Software Laboratory Limited.

Note to U.S. Government users:
Documentation and programs related to restricted rights - use, duplication or disclosure is subject to restrictions set forth in:
GSA FMSS Schedule Contract No. GS00K92AFS2505-PS05.

License Information

You have access to **Visual Importer ETL** software and documentation pursuant to the terms of a Software License Agreement granted by DB Software Laboratory Limited. As a user of this software and documentation, you are bound by the terms of the Software License Agreement. All rights, title, and interest to this software remain with DB Software Laboratory Limited.
Requests for copies of this publication and for technical information about DB Software Laboratory products should be made directly to DB Software Laboratory Limited.

Disclaimer

All information in this manual is subject to periodic change and revision without notice. While every effort has been made to ensure that this manual is accurate, DB Software Laboratory Limited excludes its liability for errors or inaccuracies (if any) contained herein.

Registered Marks

Any products or services mentioned or depicted in this document are identified by the trademarks or service marks of their respective companies or organisations.

Edition Information

This document refers to **Visual Importer ETL** version 4.9.9.4

1. Introduction

Visual Importer ETL is a business intelligence tool that provides solution for data import between different data sources and targets.

2. Requirements

Below is a List of Software that must be installed before installation of **Visual Importer ETL**:

Software	**Version**		**Notes**
Microsoft Windows	98 or higher		
MDAC	2.6 or higher	Part of OS on W2K, XP, Vista.	
MS Access ODBC driver	4.00.6364.00 or higher	Part of OS on W2K, XP, Vista.	Only to load data from MS Access 95-2003 Databases
MS Access 2007 ODBC driver	12.00.4518.1014 or higher	Separate download	Only to load data from MS Access 2007 Databases
FoxPro ODBC driver	6.1.8629.1 or higher	Separate download	Only to load data from DBF/FoxPro Files
SQL ODBC driver	2000.81.9041.40	Part of OS on W2K, XP, Vista.	Only to import data into MS SQL Server 7/2008
Interbase client		GDS32.DLL	Only to work with Interbase or Firebird Databases
SQLite		Sqlite3.dll	Only to import data into SQLite databases
Oracle Client	7.3.4 or higher	Provided by Oracle	Only to import data into Oracle Databases

Separate Downloads:

FoxPro ODBC driver
http://msdn.microsoft.com/en-us/vfoxpro/bb190233.aspx

Office 2007 Data Access Components
http://www.microsoft.com/downloads/details.aspx?FamilyID=7554F536-8C28-4598-9B72-EF94E038C891&displaylang=en

Copyright © 2011 DB Software Laboratory

Working with Oracle:

Oracle client 8.1.7 to load data into/from Oracle
Or
Oracle client 9 to load data into/from Oracle
Or
Oracle client 10 to load data into/from Oracle
Or
Oracle client 11 to load data into/from Oracle

Note:
Depending on the Requirements you may or may not need to have all components installed
There is no need to install clients for MySql and PostgreSQL they are integrated into the software itself.

3. Key features

Data import

Data targets:

- Oracle 7-11g database (using OCI API)
- SQL server 7- 2008 (using BCP API)
- ODBC source (using ODBC API)
- Interbase/Firebird
- MySQL
- PostgreSQL
- SQLite

Data sources:

- Multiple Delimited or Fixed width Text files
- Multiple Excel files + Multiple Excel Spreadsheets
- Multiple MS Access Databases + Multiple Tables
- Multiple DBF Files
- Multiple Tables
- Oracle 7-11g database
- SQL server 7- 2005
- ODBC source
- Interbase/Firebird
- MySQL
- PostgreSQL
- SQLite
- XML

This product features:

- Great performance - hundreds of records per second
- Comprehensive Error log
- Rejected records file
- Integrated Expressions builder
- Data Preview

Allows the user to perform calculation during the loading process such as fields splitting, concatenations, data formatting, and load data from cross tables.

Oracle

There are two ways of loading available:

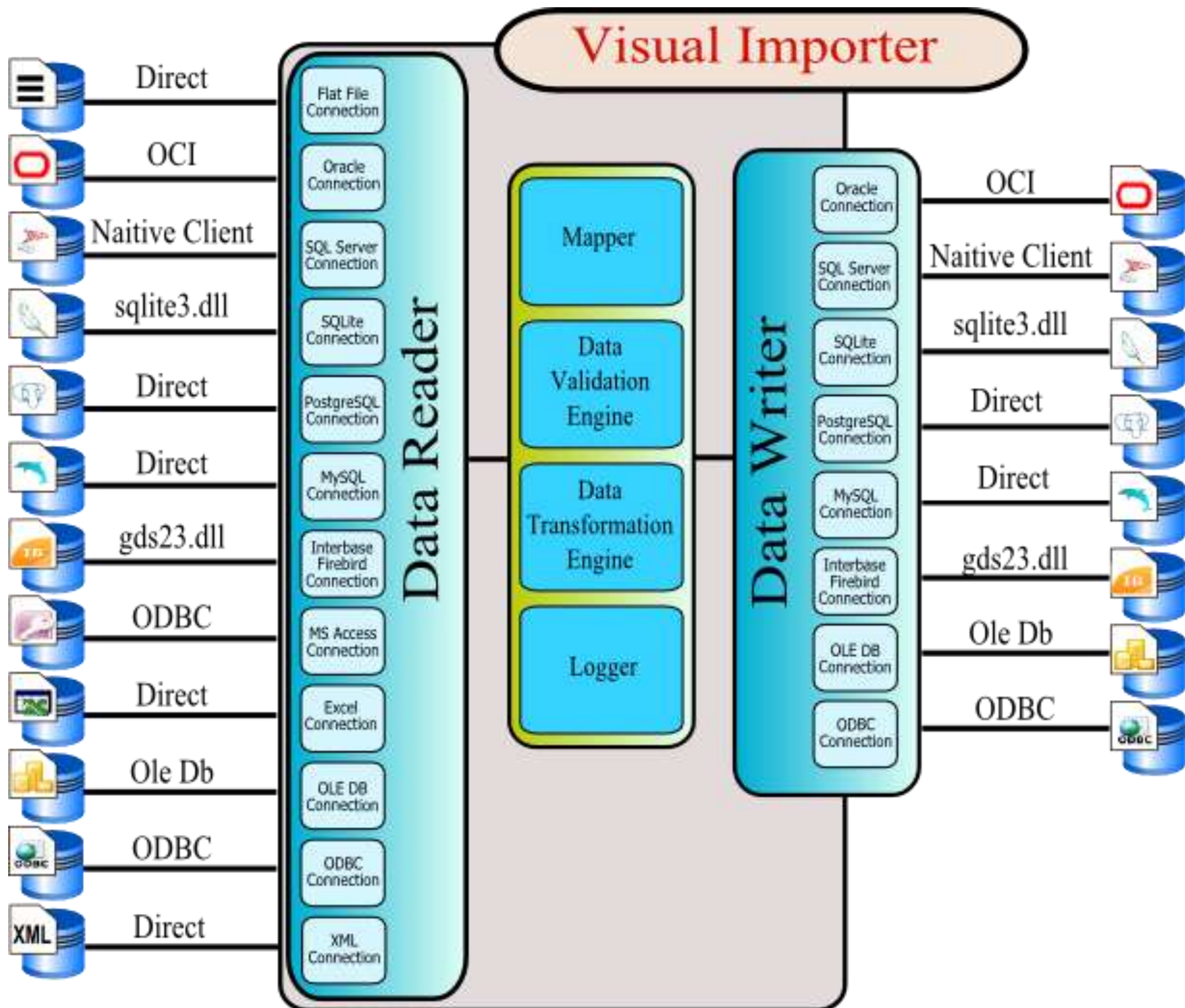
- Oracle direct path loading
- Conventional path

SQL server

This software uses the same API as Microsoft DTS service.

Copyright © 2011 DB Software Laboratory

4. Visual Importer ETL Architecture



Copyright © 2011 DB Software Laboratory

5. Importing Data

In order to load data from the data source into the data target you must define data mapping between target table and data source.

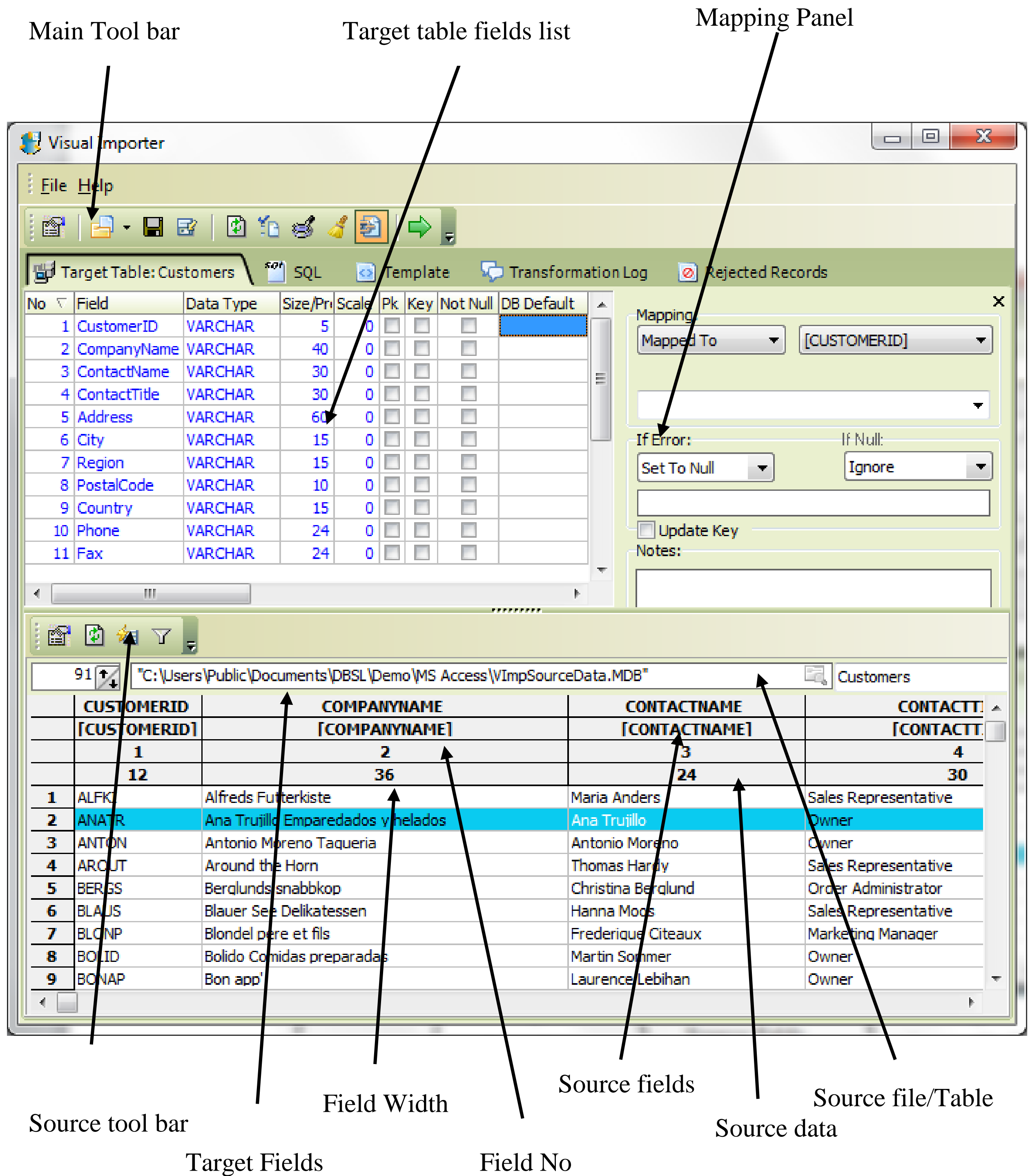
Possible data sources are:

- Multiple Delimited or Fixed width Text files
- Multiple Excel files + Multiple Excel Spreadsheets
- Multiple MS Access Databases + Multiple Tables
- Multiple DBF Files
- Multiple Tables
- Oracle 7-11g database
- SQL server 7- 2005
- ODBC source
- Interbase/Firebird
- MySQL
- PostgreSQL
- SQLite
- XML

Imports Scripts screen is designed to allow user create, delete, modify, and test data mapping to the target databases.

Copyright © 2011 DB Software Laboratory

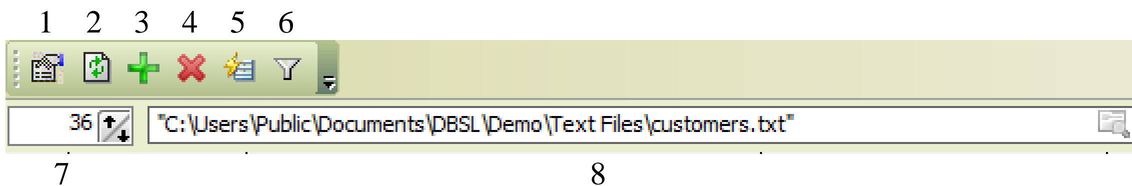
5.1 Screen Overview



5.2 Main tool bar

1. Data Target options
2. Loads Import Script From the file
3. Saves Import Script to the file
4. Saves as
5. Refreshes fields’ list from the database
6. Checks script for the errors
7. Data preview
8. Allows user to clear field mapping
9. Hides mapping panel
10. Data Import

5.3 Source tool bar

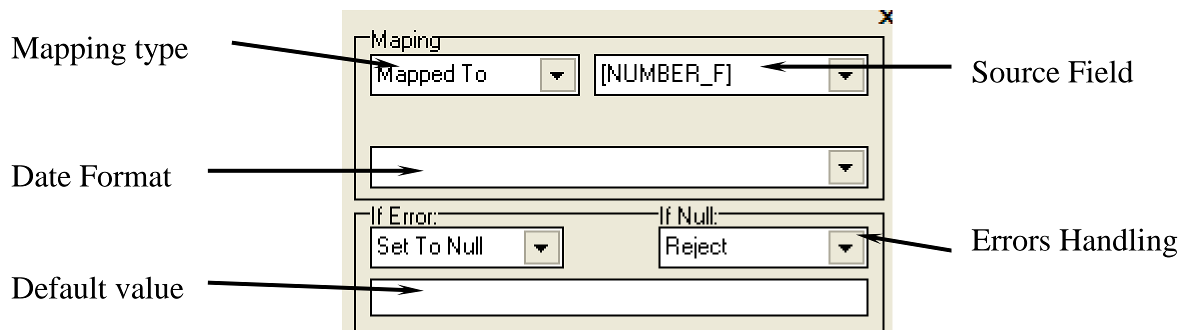


1. Data source options
2. Refreshes Source data
3. Adds new column
4. Deletes last column
5. Auto map the source fields to the target fields
6. Filter
7. Records to Show
8. Sources file name/ table name

Copyright © 2011 DB Software Laboratory

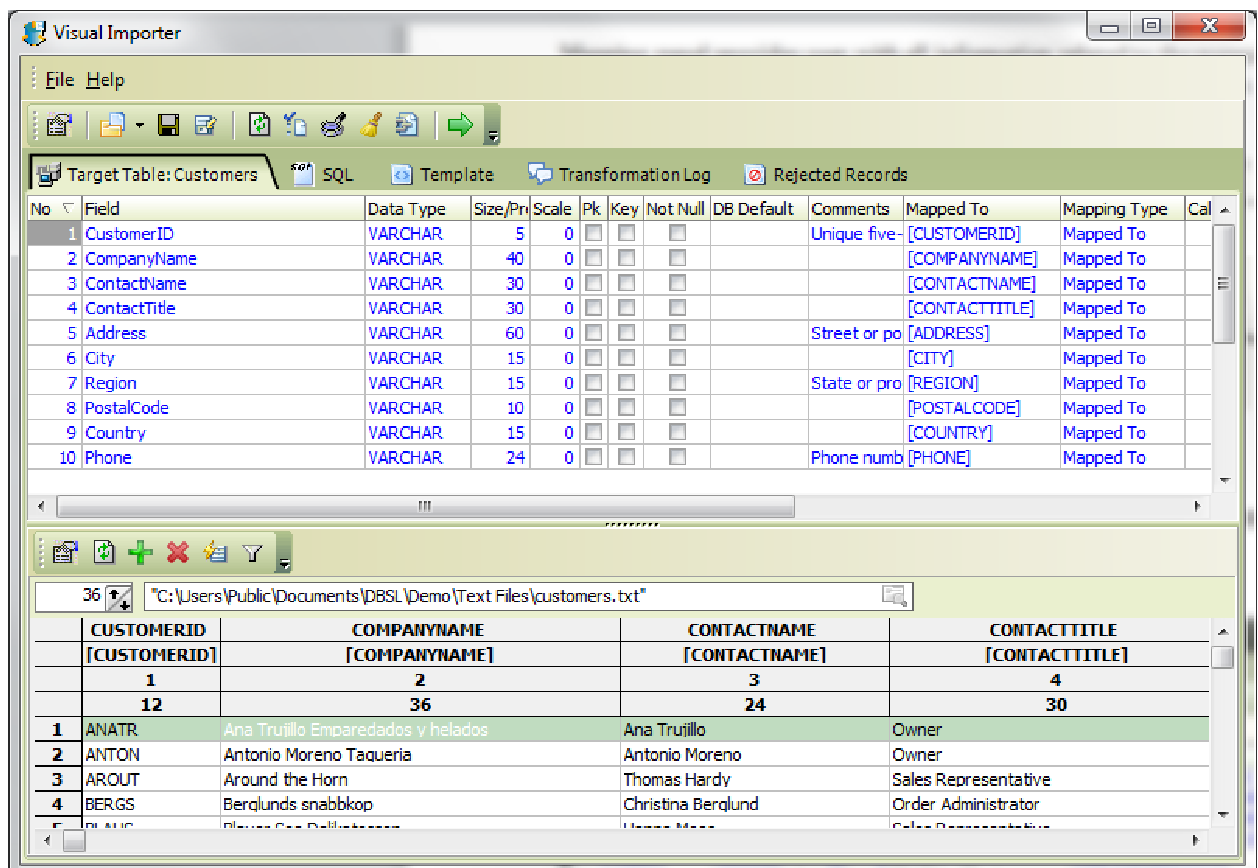
5.4 Mapping panel

Mapping panel provides user with all information related to the mapping of one particular field. There are two ways of mapping: direct and through calculations.



Alternately you may hide Mapping panel and use grid to perform mapping.

See the picture below



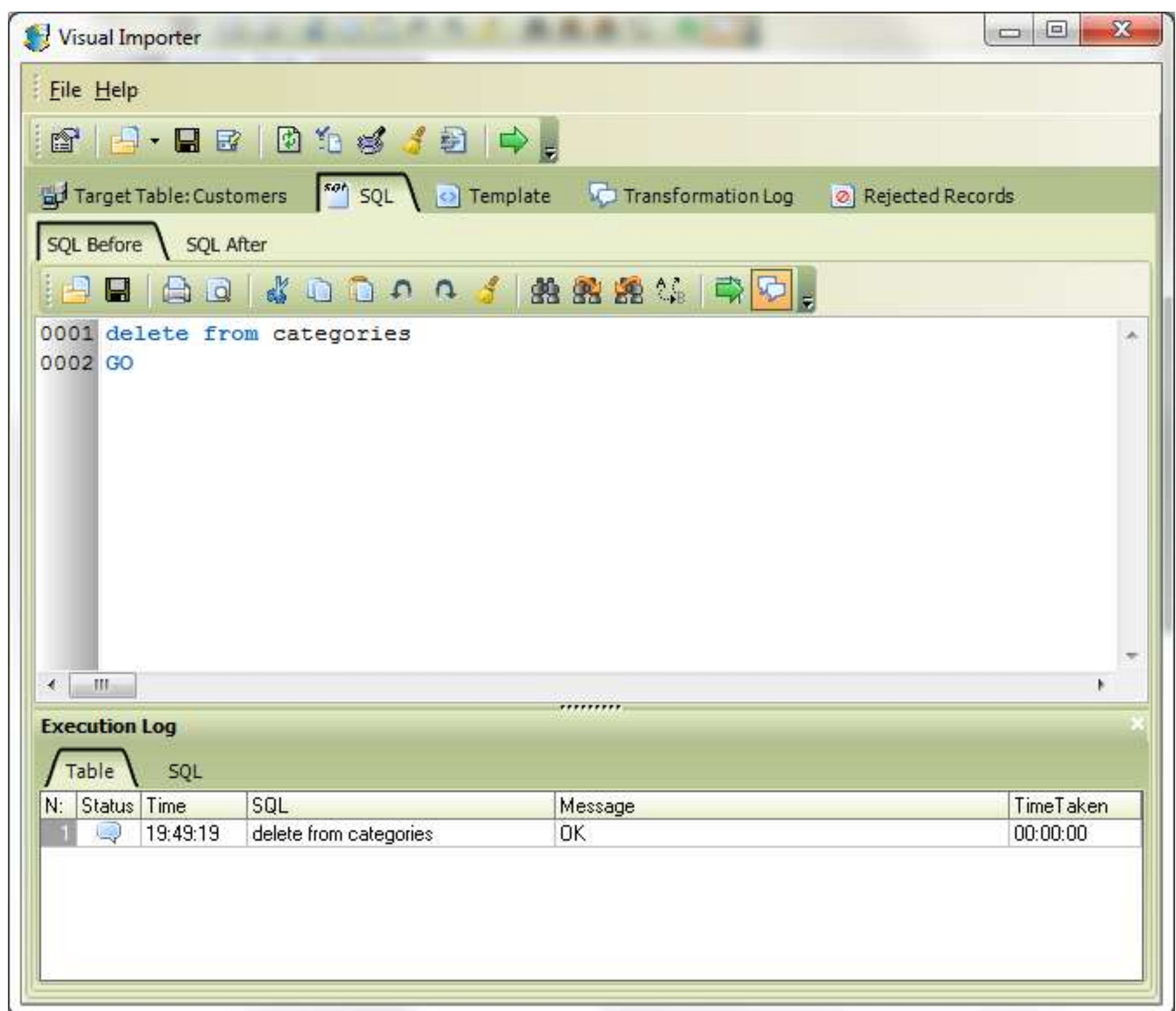
Copyright © 2011 DB Software Laboratory

5.5 SQL Statements

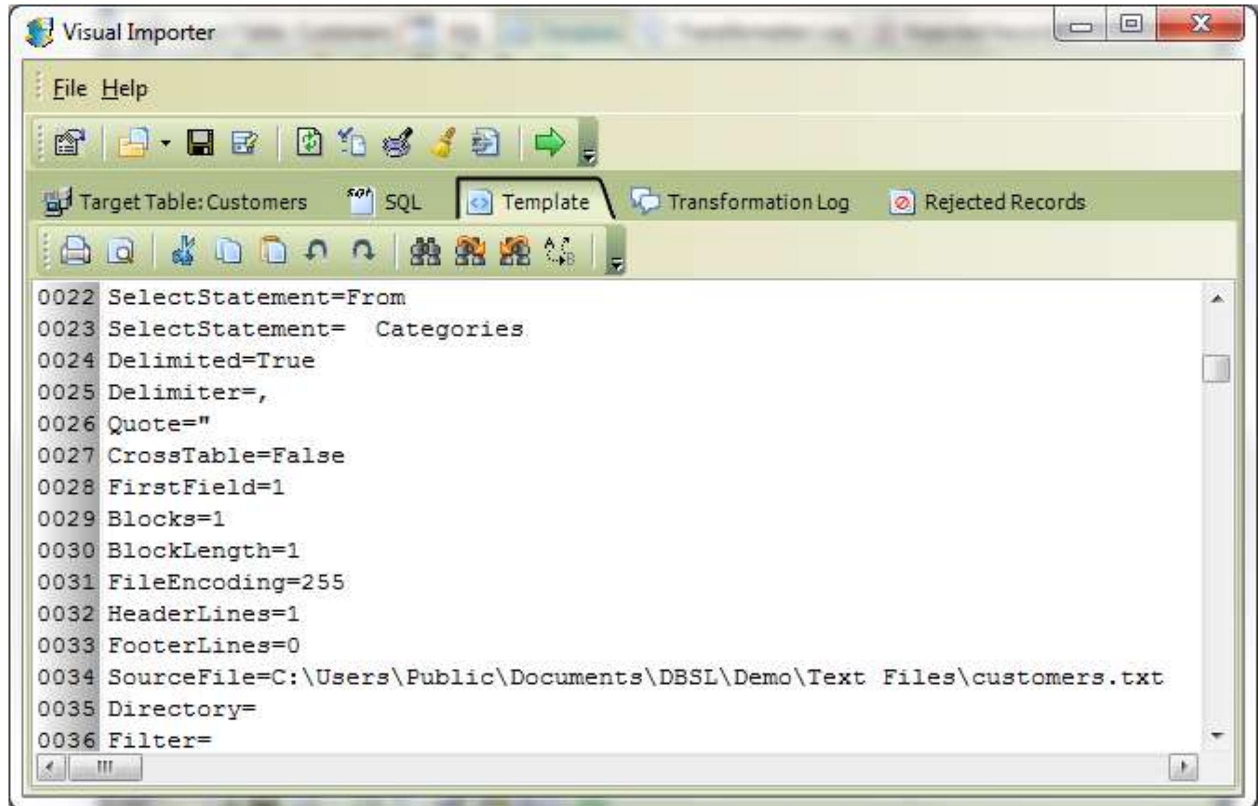
Visual Importer ETL provides functionality to run SQL statements before and after data import.

Note:

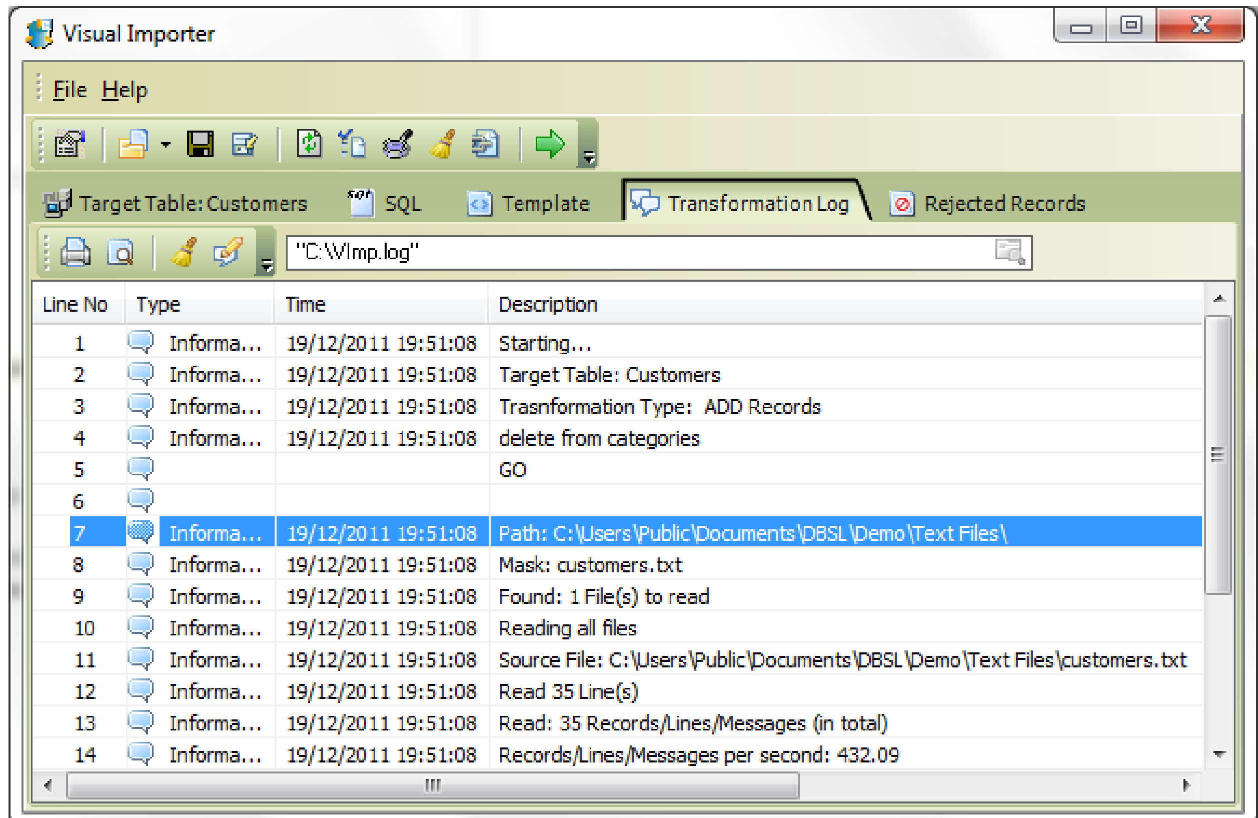
- In order to Execute several SQL statements user must specify SQL delimiter
- No select statements allowed



5.6 Template tab

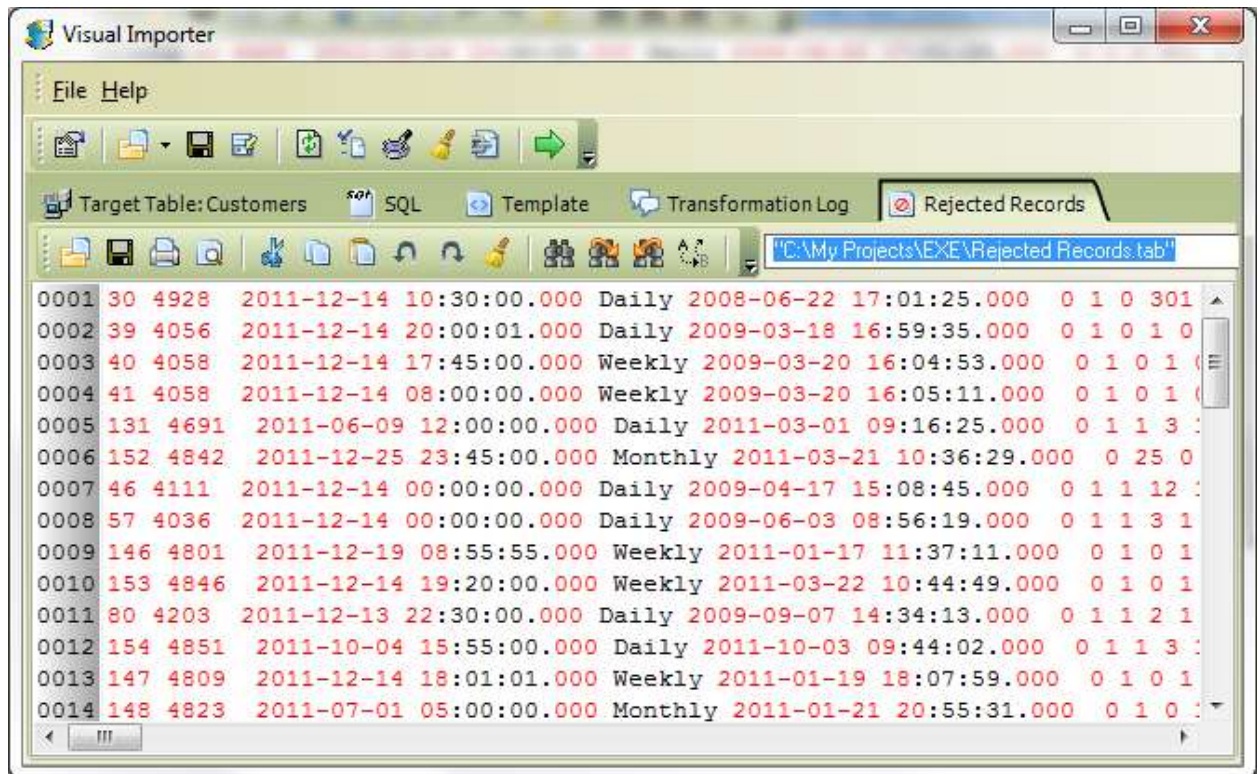


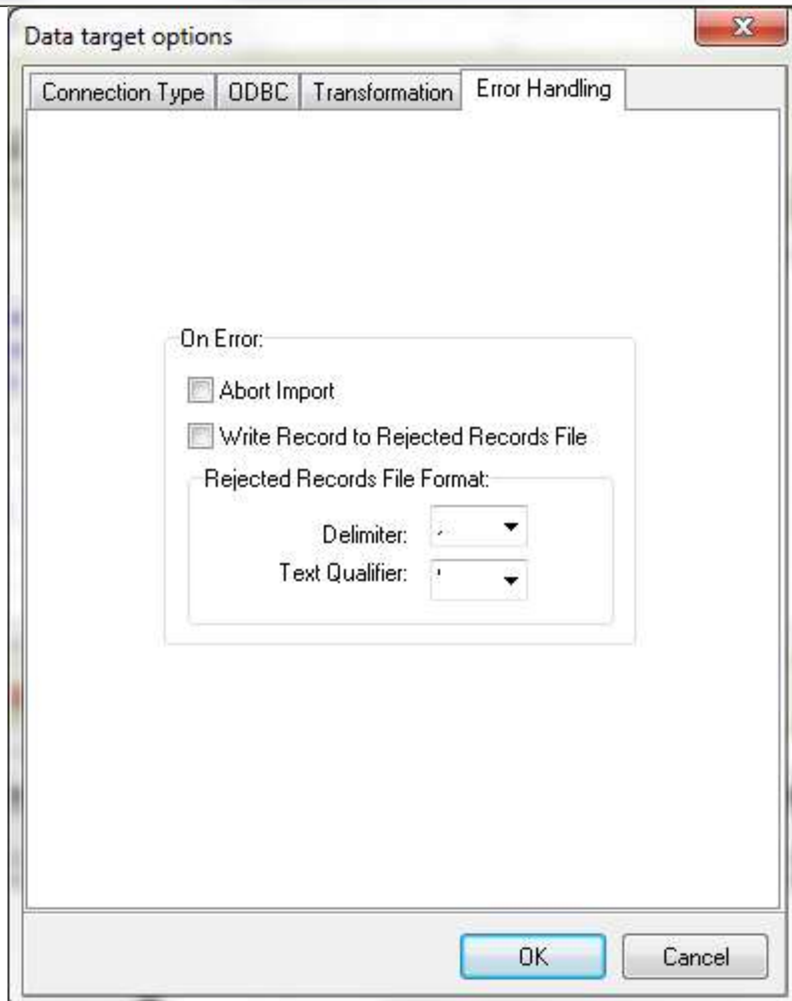
5.7 Import Log Tab



Copyright © 2011 DB Software Laboratory

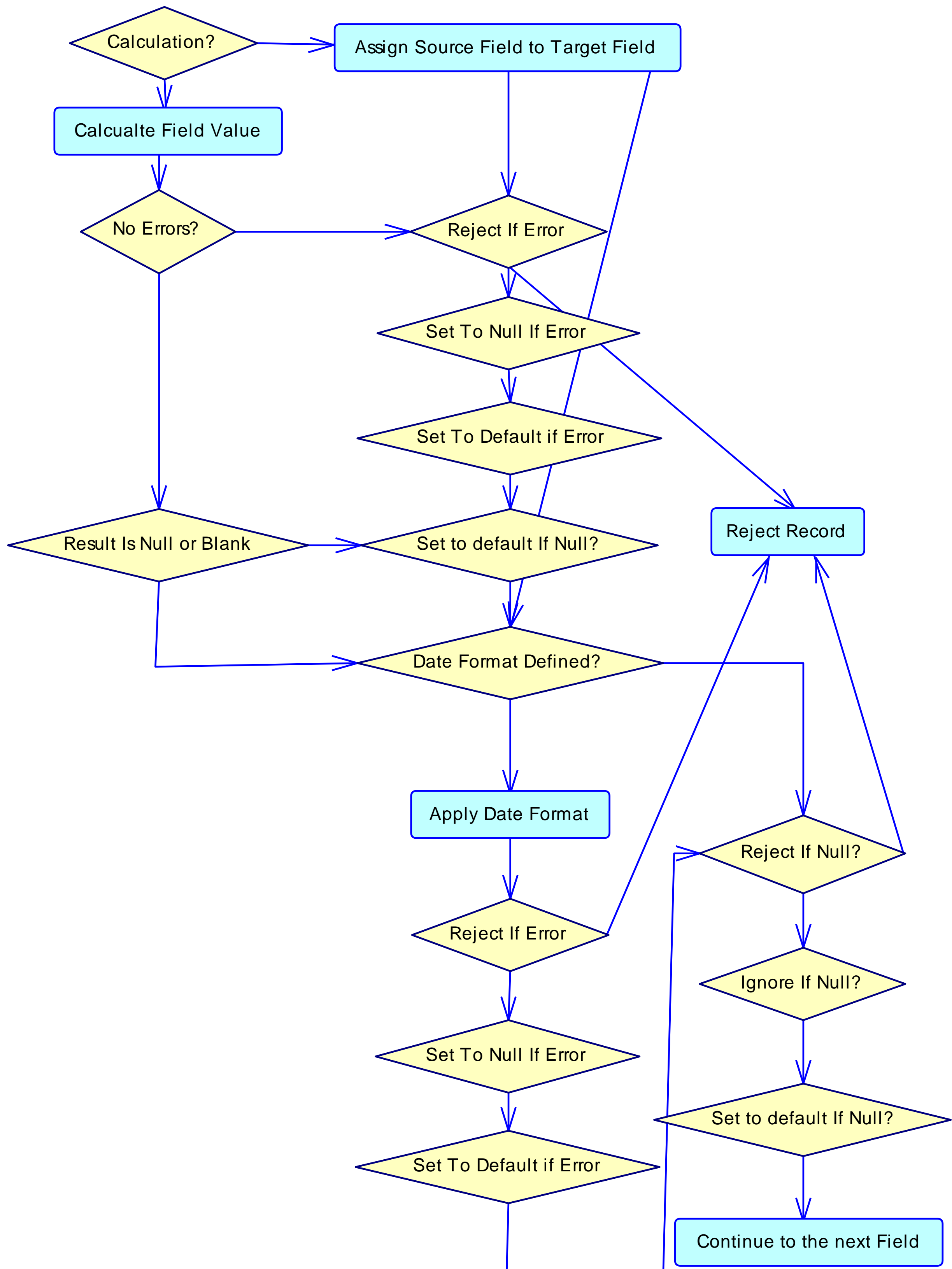
5.8 Rejected Records





Copyright © 2011 DB Software Laboratory

5.9 Import Process



Note:
Records can also be rejected by the Server.

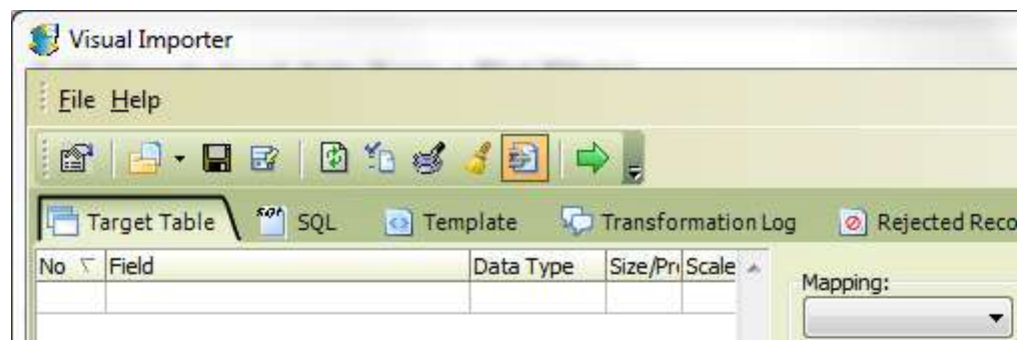
Copyright © 2011 DB Software Laboratory

5.10 How to load data from a Flat File(s)

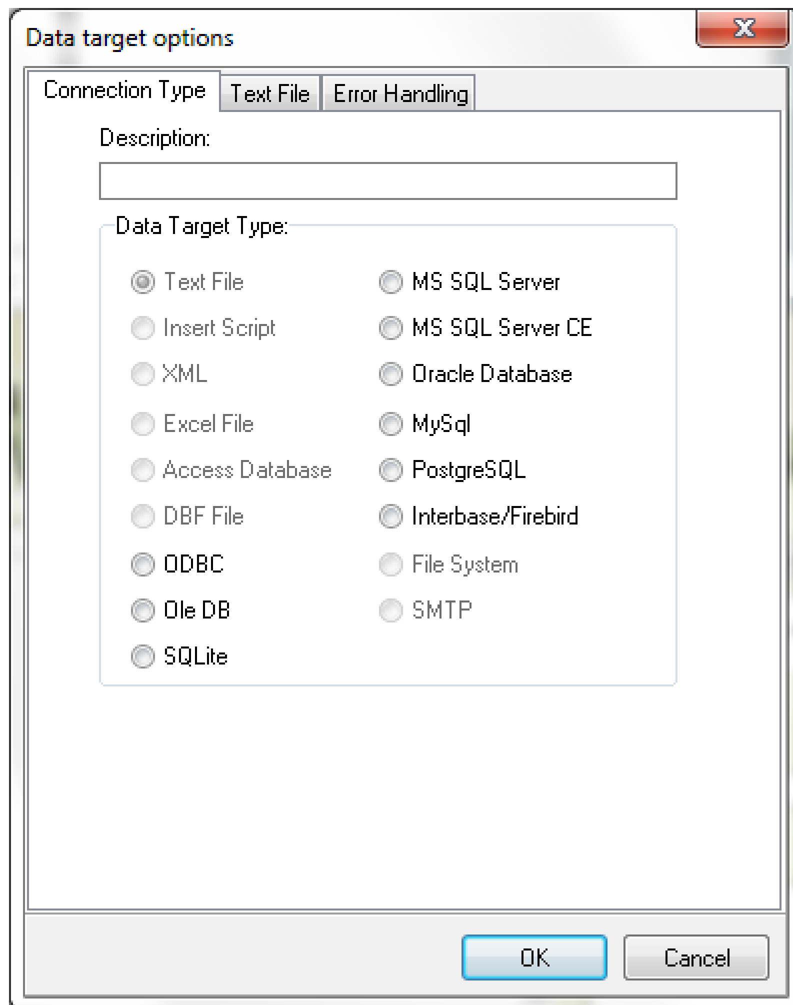
Visual Importer ETL can load data from ASCII and Unicode files in UTf8, UTF16BE and UTF16LE formats with BOM marker and without

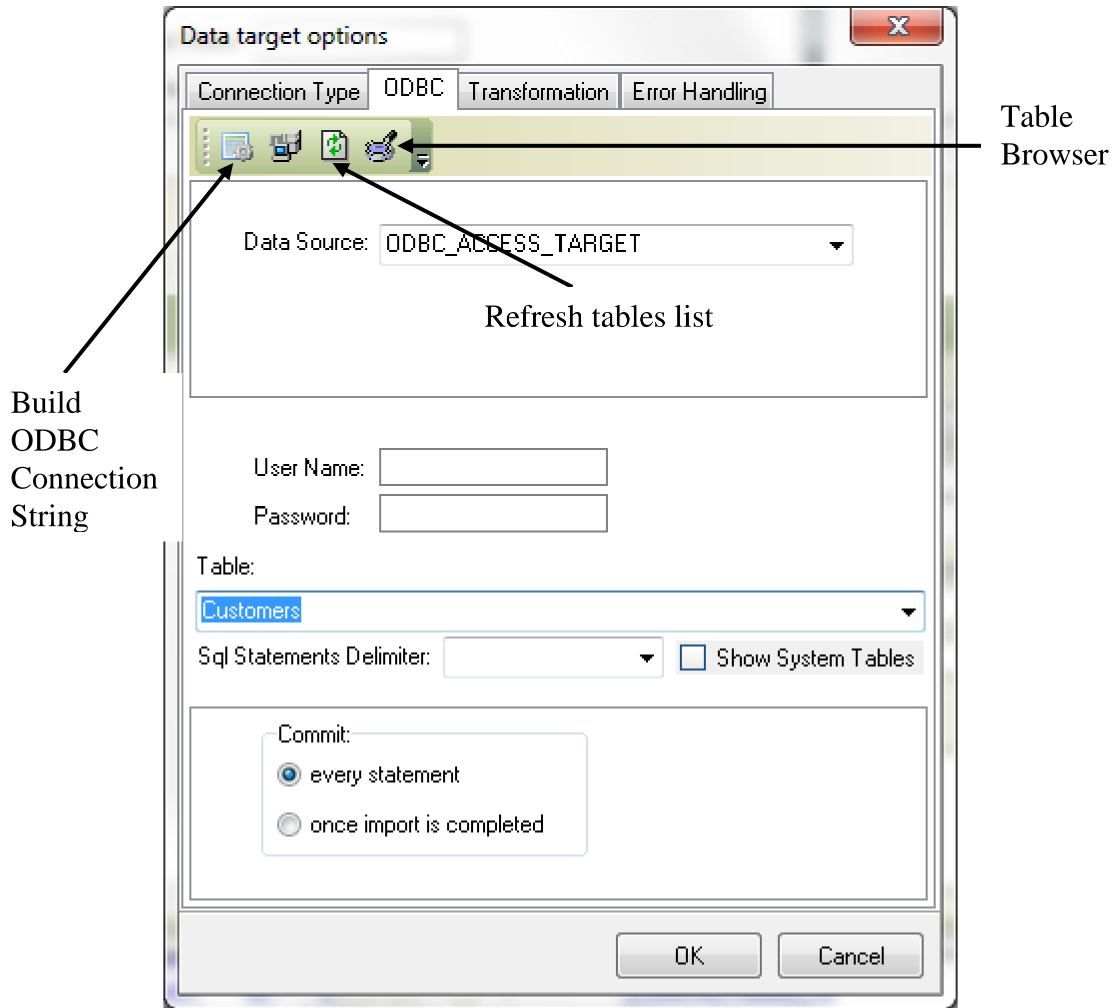
Unicode is a computing industry standard allowing computers to consistently represent and manipulate text expressed in most of the world's writing systems.
To perform data mapping:

- Click Data Target Options button

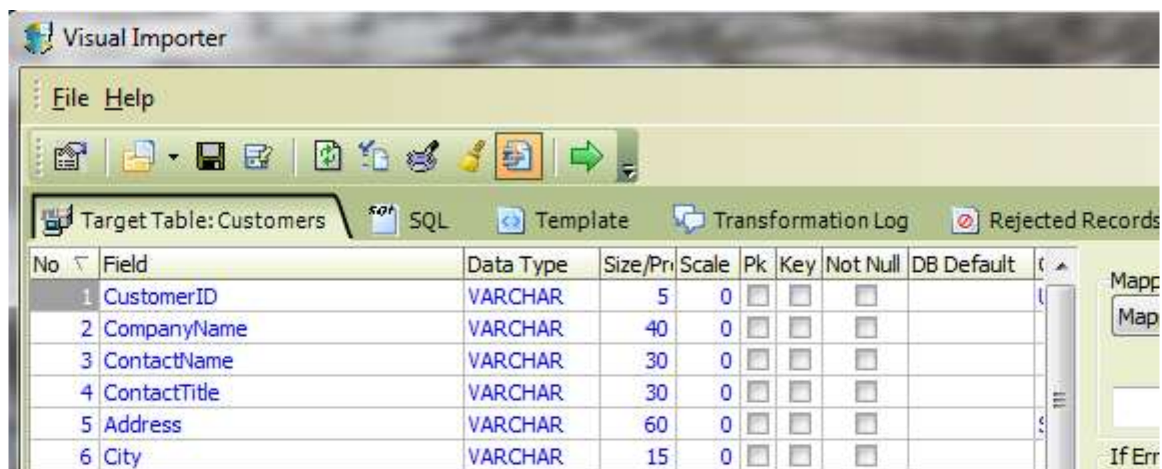


- Dialog box will appear
- Select appropriate target type
- Click Get tables list
- Select Table you would like to import data into from Drop Down List
- Click Transformation tab
- Select Add Records
- Click OK


Data target options
Connection Type
Text File
Error Handling
Description:
Data Target Type:
Text File
MS SQL Server
Insert Script
MS SQL Server CE
XML
Oracle Database
Excel File
MySql
Access Database
PostgreSQL
DBF File
Interbase/Firebird
ODBC
File System
Ole DB
SMTP
SQLite
OK
Cancel



The following list of fields should appear

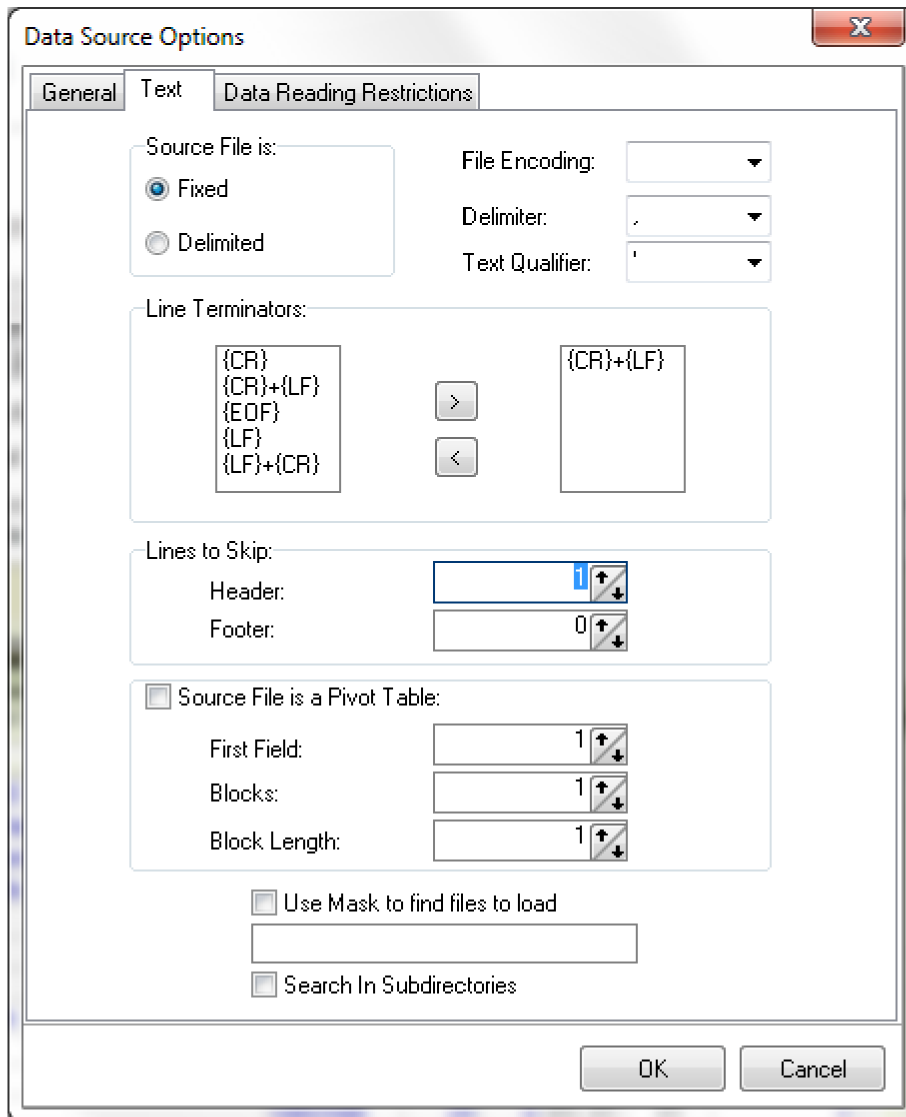


No	Field	Data Type	Size/Pr	Scale	Pk	Key	Not Null	DB Default
1	CustomerID	VARCHAR	5	0				
2	CompanyName	VARCHAR	40	0				
3	ContactName	VARCHAR	30	0				
4	ContactTitle	VARCHAR	30	0				
5	Address	VARCHAR	60	0				
6	City	VARCHAR	15	0				

Click Data Source Option Button

Copyright © 2011 DB Software Laboratory

Dialog box will Appear
Set Delimiter and Quota to appropriate values



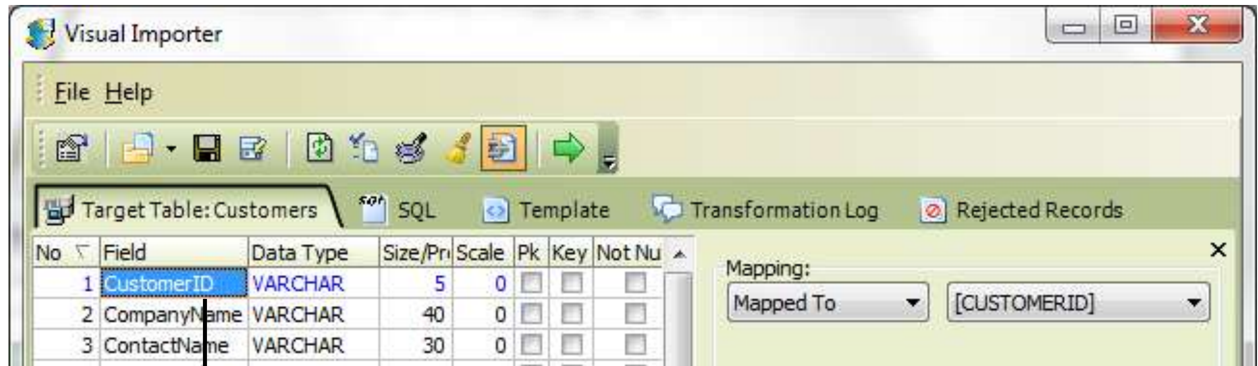
Note:
If you want to load data from several files specify mask.

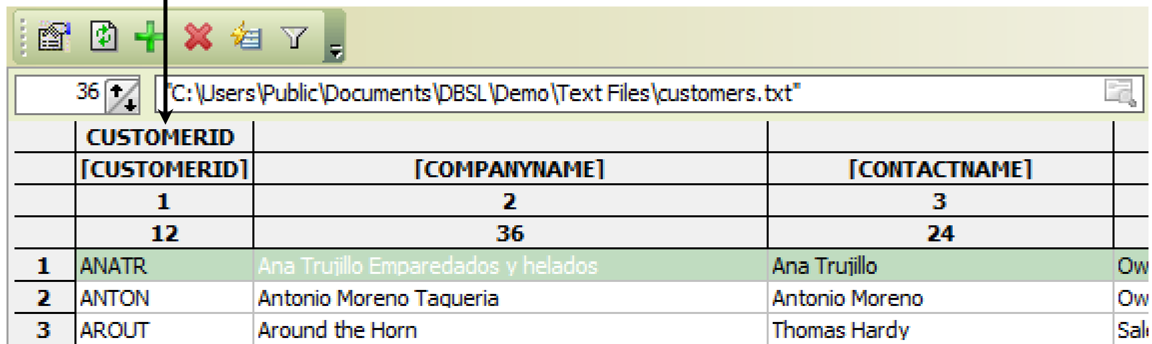
Click OK. Click and select the file you would like to import.

92 "C:\Program Files\DB Software Laboratory\Demo\Buffer\Customers.tab"

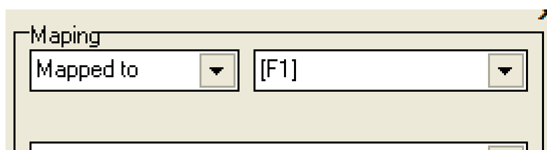
	[F1]	[F2]	[F3]	
	1	2	3	
	13	36	23	
1	CustomerID	CompanyName	ContactName	Conta
2	ALFKI	Alfreds Futterkiste	Maria Anders	Sales
3	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owne

Select Fist field in the Data Target fields list and drag and drop it above [F1] field.





You may change field mapping by using mapping panel at any time.



5.11 How to perform Auto mapping

If the Data Source and Data Target have got the same fields' names you may use Auto map feature.

Click , Fill in all necessary data and click map.

Auto Map

Auto Map Fields using:

Match By Field Position

Match by Field Names

Ignore Case Differences

Ignore Special Characters

Use Source Field Prefix [

Use Source Field Suffix]

Use Target Field Prefix [

Use Target Field Suffix]

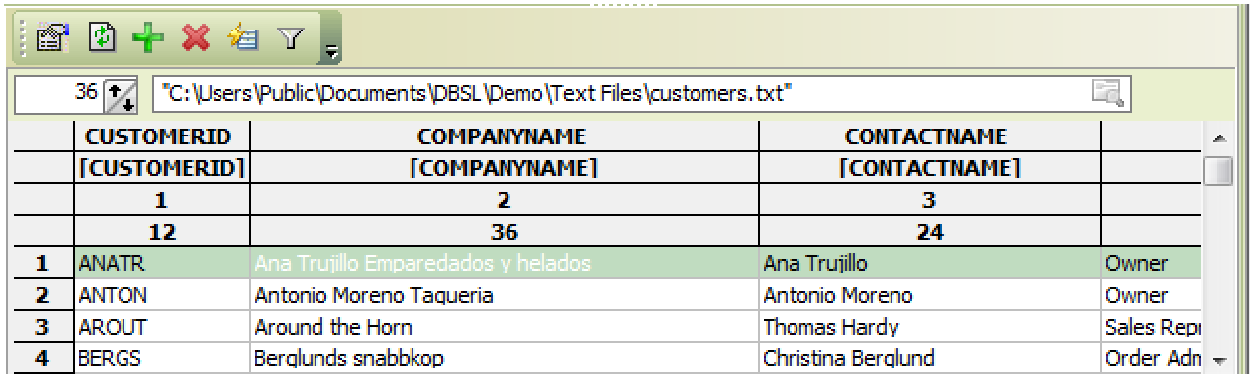
Set Date Format to YYYY-MM-DD HH:NN:SS

Map Now

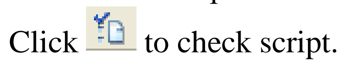
No	Source Field	Target Fields	Date Format
1	[CUSTOMERID]	CustomerID	
2	[COMPANYNAME]	CompanyName	
3	[CONTACTNAME]	ContactName	
4	[CONTACTTITLE]	ContactTitle	
5	[ADDRESS]	Address	
6	[CITY]	City	
7	[REGION]	Region	
8	[POSTALCODE]	PostalCode	
9	[COUNTRY]	Country	
10	[PHONE]	Phone	
11	[FAX]	Fax	

OK Cancel

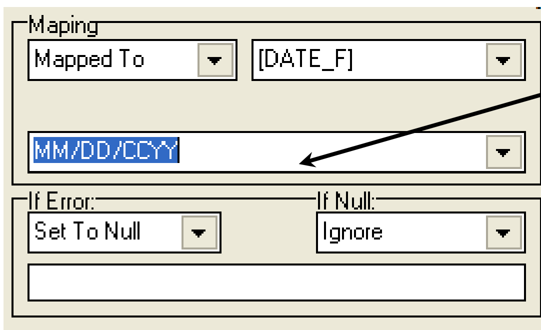
Copyright © 2011 DB Software Laboratory



Now we are ready to import data.
Let's check script first.

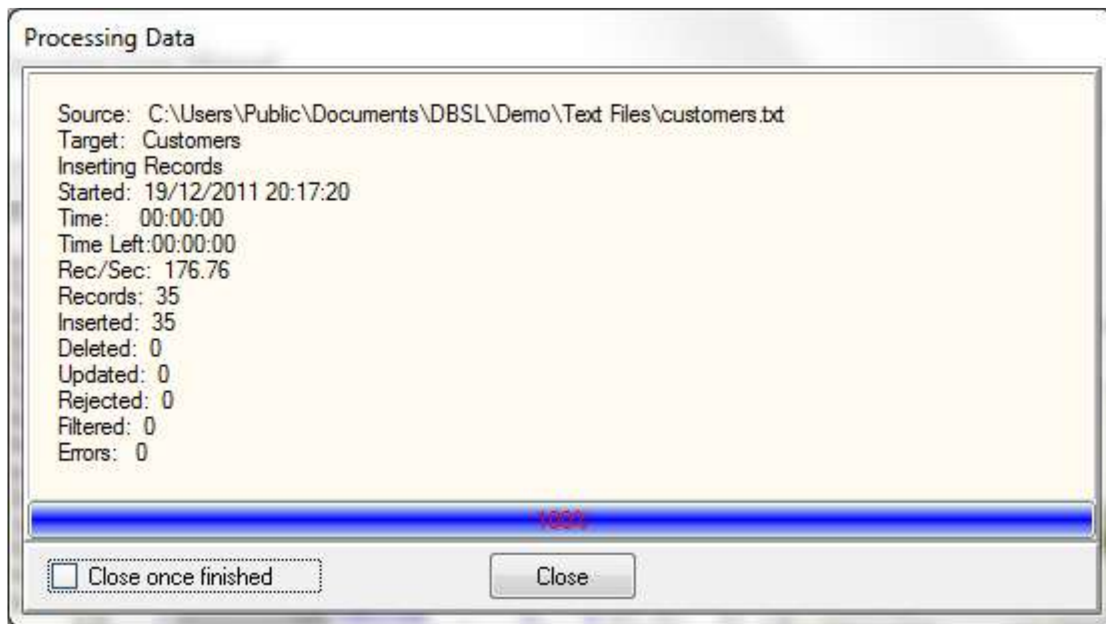
Click  to check script.

We have to correct the error first.



Date format is missing

Click to load data into the database



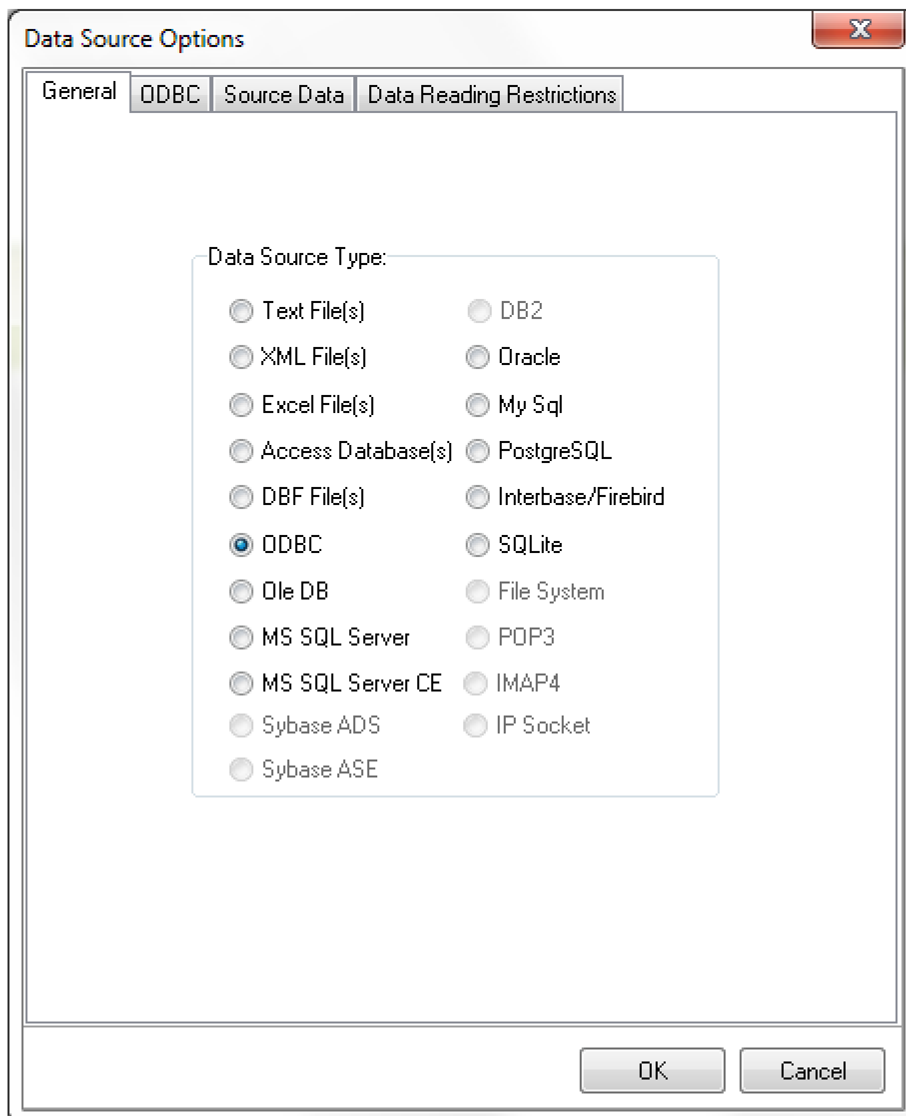
Once loading is finished you may check Log file or Rejected records file.

5.12 How to load data from an ODBC Data Source

Data mapping for ODBC is very similar to the flat file mapping.

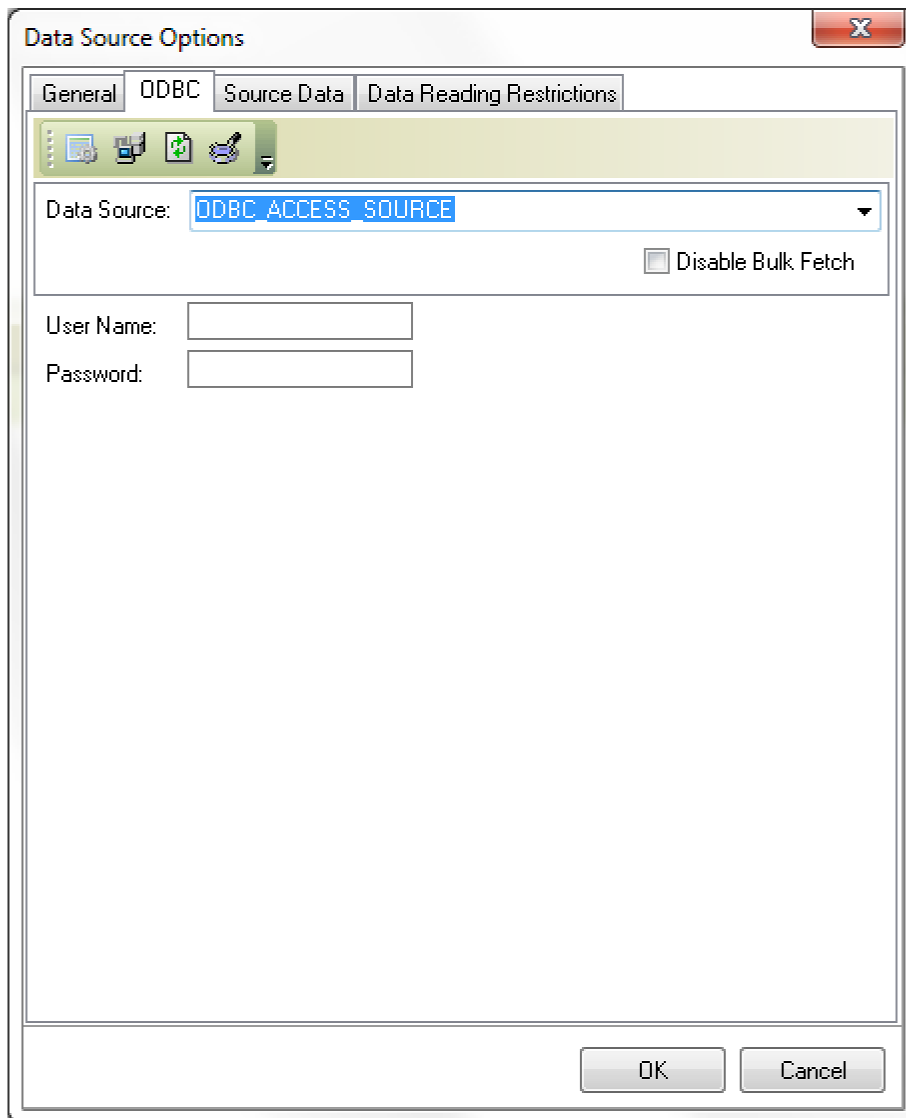
Click Data Source Option Button.

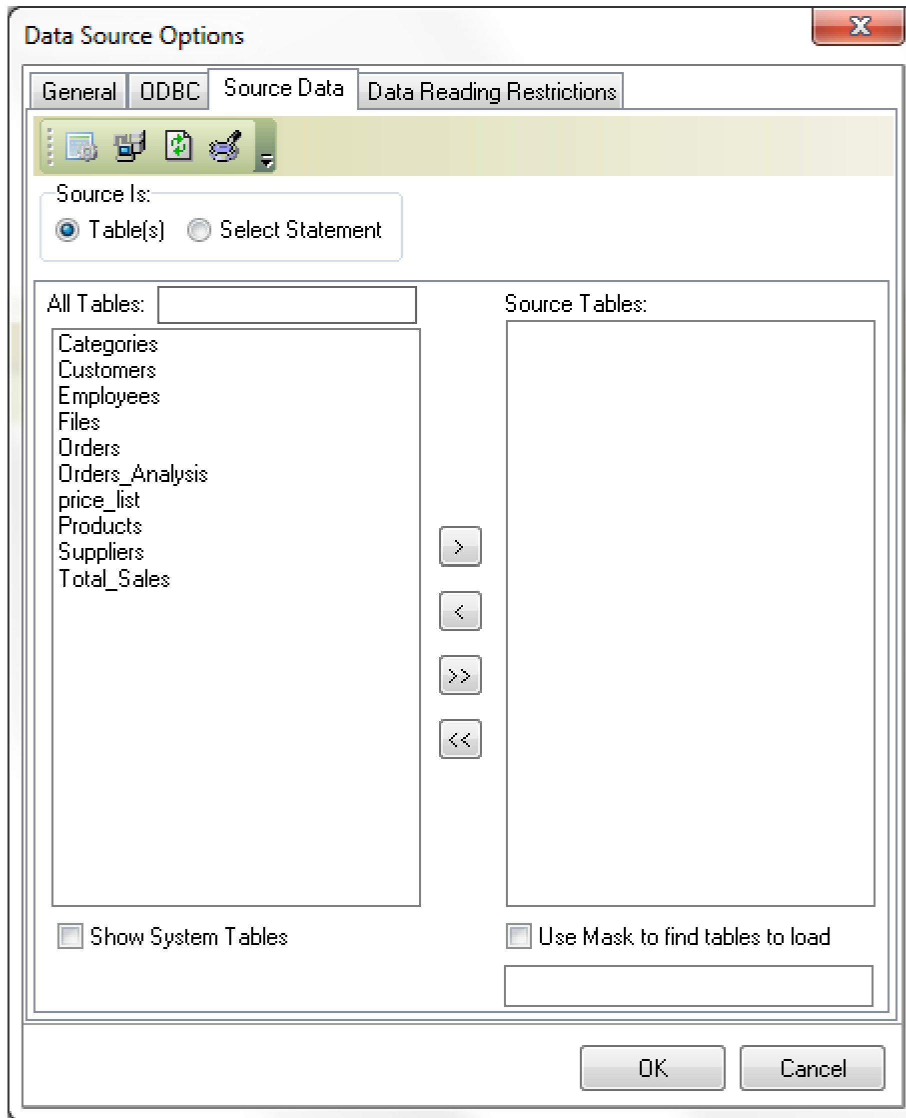
Dialog box will appear.
Set Data Source Type to 'ODBC'.



Select ODBC DSN from the Drop Down List or alternatively create a new ODBC DSN or modify the old one by using ODBC administrator.
Fill in User name and Password if required.
Click Update Tables List.
Specify tables you want to load data from or altenatevely type mask to find tables automatically every time you import data.
Click OK.

Copyright © 2011 DB Software Laboratory


Data Source Options
General
ODBC
Source Data
Data Reading Restrictions
Data Source:
ODBC_ACCESS_SOURCE
Disable Bulk Fetch
User Name:
Password:
OK
Cancel



Select a table you want to see from the drop down box.

91 Customers

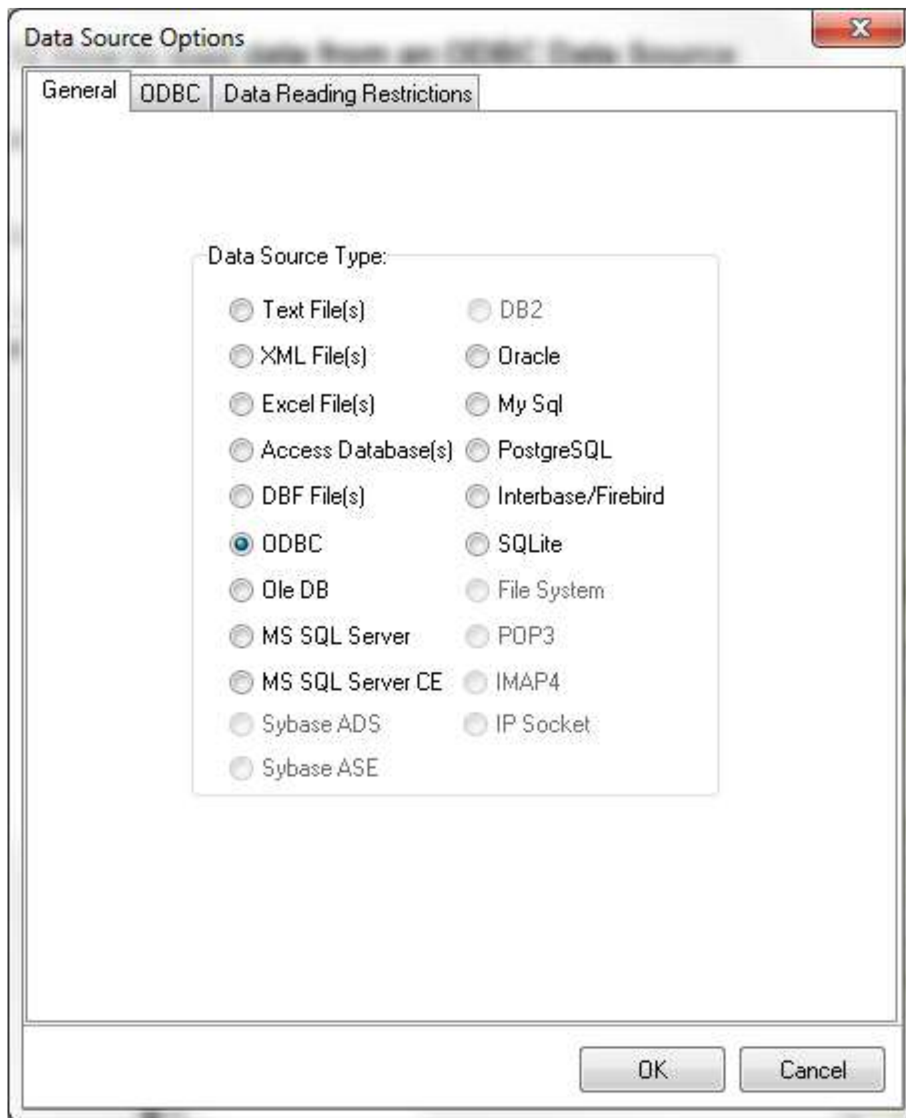
	[CUSTOMERID]	[COMPANYNAME]	[CONTACTNAME]
	1	2	3
	20	36	23
1	ALFKI	Alfreds Futterkiste	Maria Anders
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
3	ANTON	Antonio Moreno Taqueria	Antonio Moreno
4	AROUT	Around the Horn	Thomas Hardy

5.13 How to load data from Excel/Access/Dbf File(s)

Data mapping for Excel file is very similar to the flat file mapping.

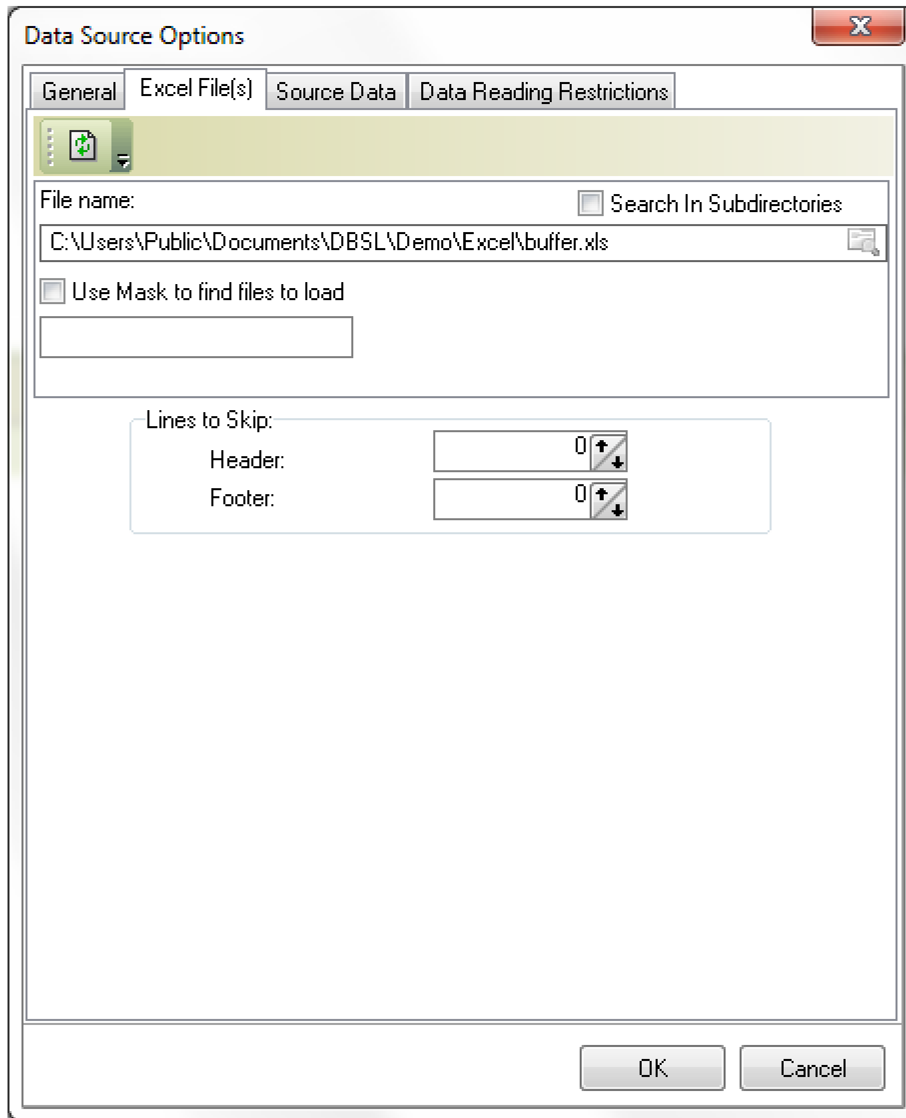
Click Data Source Option Button.

Dialog box will appear.

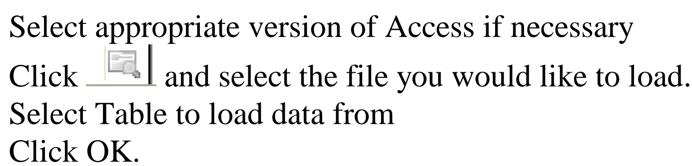


Select appropriate Data source type.

Copyright © 2011 DB Software Laboratory



Select appropriate version of Access if necessary



Click and select the file you would like to load.
Select Table to load data from
Click OK.

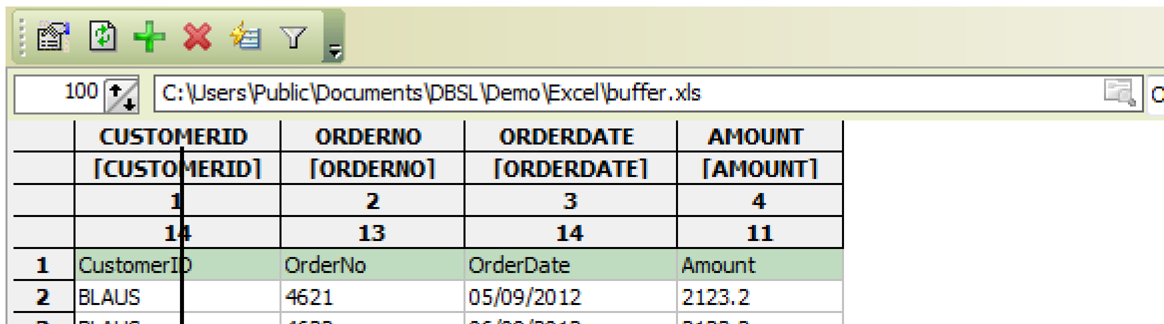
Note:
If you want to load data from several files/tables specify mask.

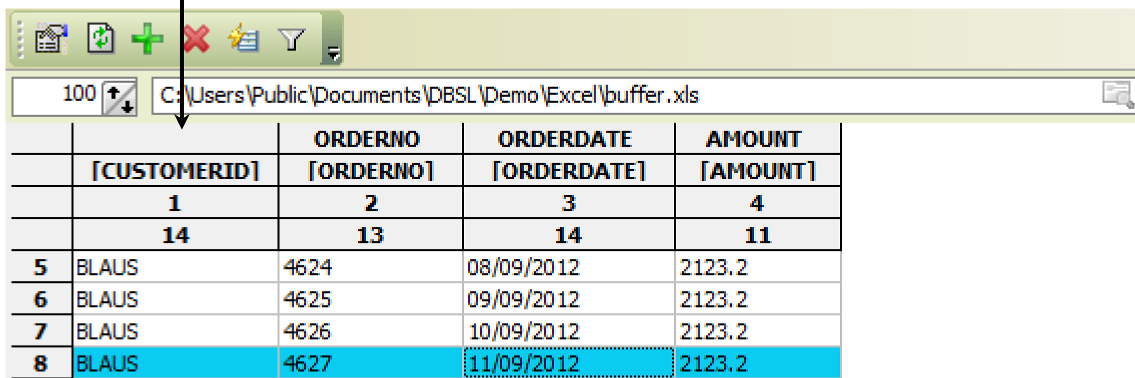
100 C:\Users\Public\Documents\DBSL\Demo\Excel\buffer.xls

	CUSTOMERID	ORDERNO	ORDERDATE	AMOUNT
	[CUSTOMERID]	[ORDERNO]	[ORDERDATE]	[AMOUNT]
	1	2	3	4
	14	13	14	11
1	CustomerID	OrderNo	OrderDate	Amount
2	BLAUS	4621	05/09/2012	2123.2

5.14 How to clear mapping

Click .
Click on the field you would like to clear





Copyright © 2011 DB Software Laboratory

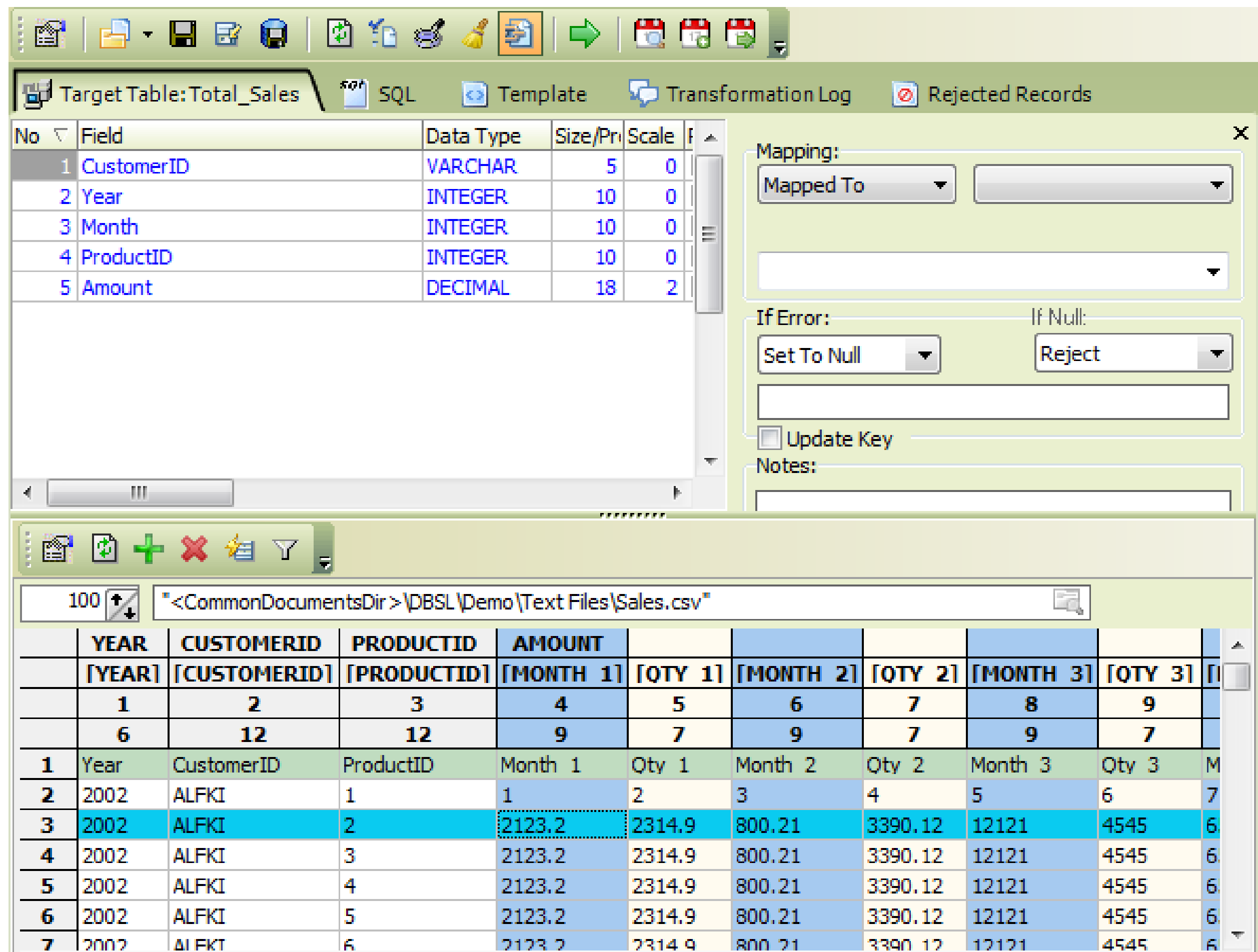
5.15 How to load data from the Cross/Pivot table

Let us say we have table like the following in the database:

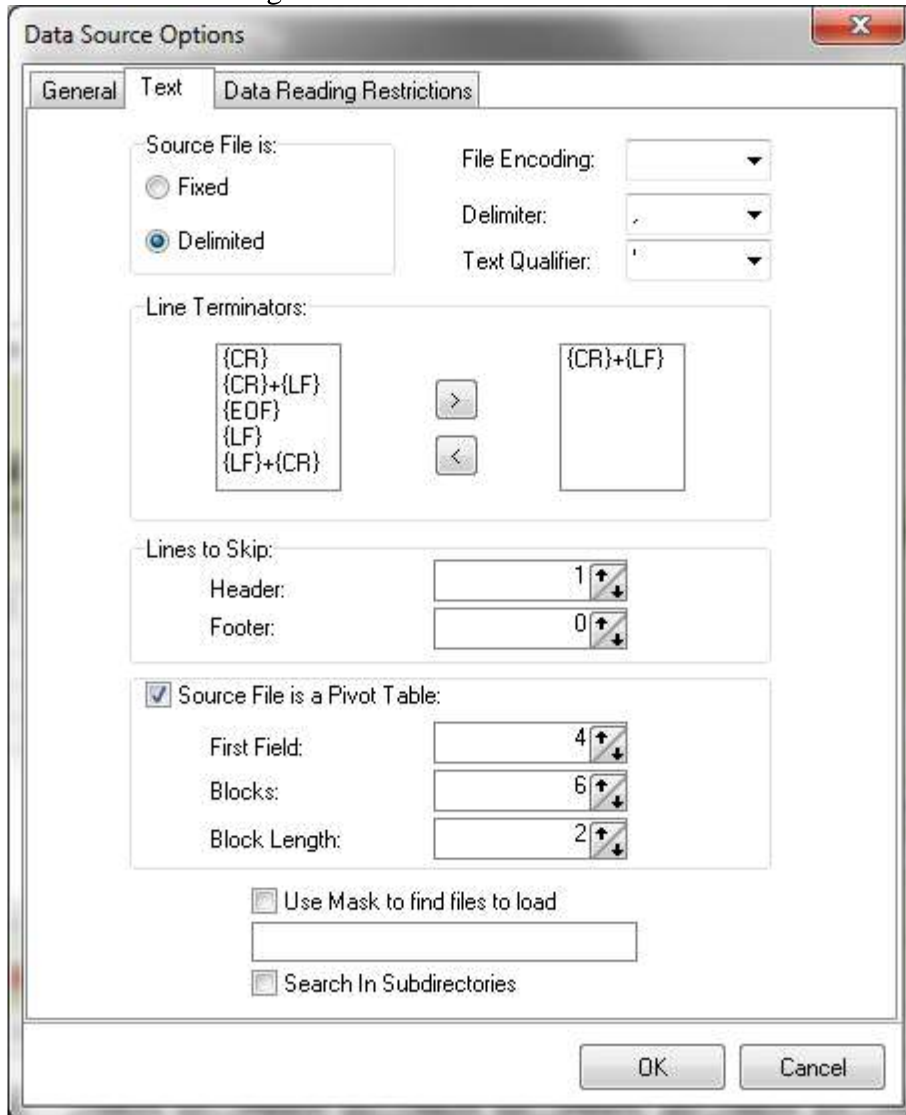
No	Field	Data Type
1	CUSTOMERID	CHAR
2	YEAR	DECIMAL
3	MONTH	DECIMAL
4	PRODUCTID	DECIMAL
5	AMOUNT	DECIMAL

And a text files like the one below:

Year
CustomerID
ProductID
Month_1
Qty_1
…
Month_12
Qty_12



Click Data Source Button and check ‘Source file is a Cross table’ check box and set First Field to 4, Blocks to 12 and Block length to 2



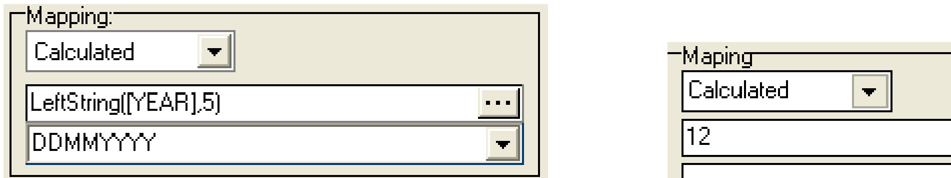
Finally we are ready to import data

Click to load data into the database

5.16 How to perform Calculations

Visual Importer ETL is capable of performing calculations during import.

To perform a simple calculation set mapping type to calculation and type constant or formula into calculation edit box

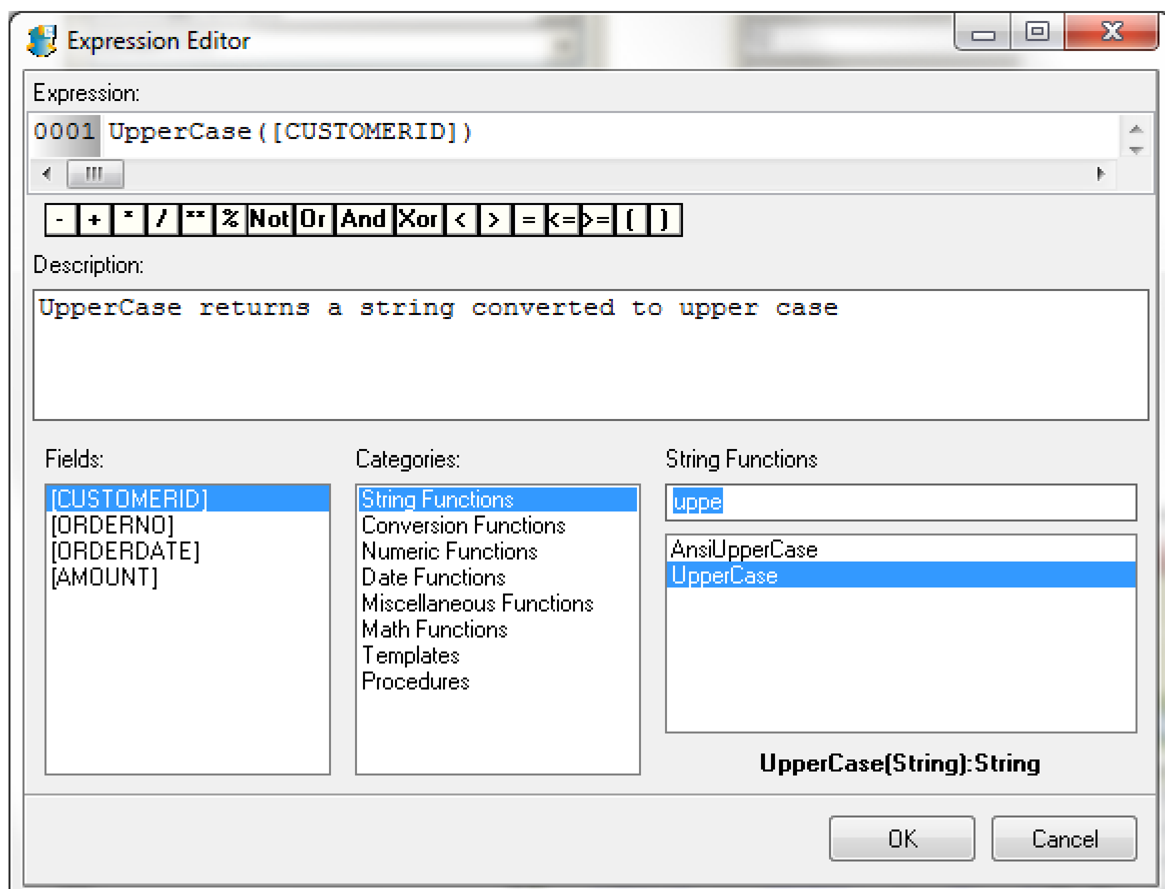


Multiplying fields
StrToFloat([INTEGER_F])* StrToFloat([FLOAT_F])

Concatenation
[INTEGER_F]+ kilos"

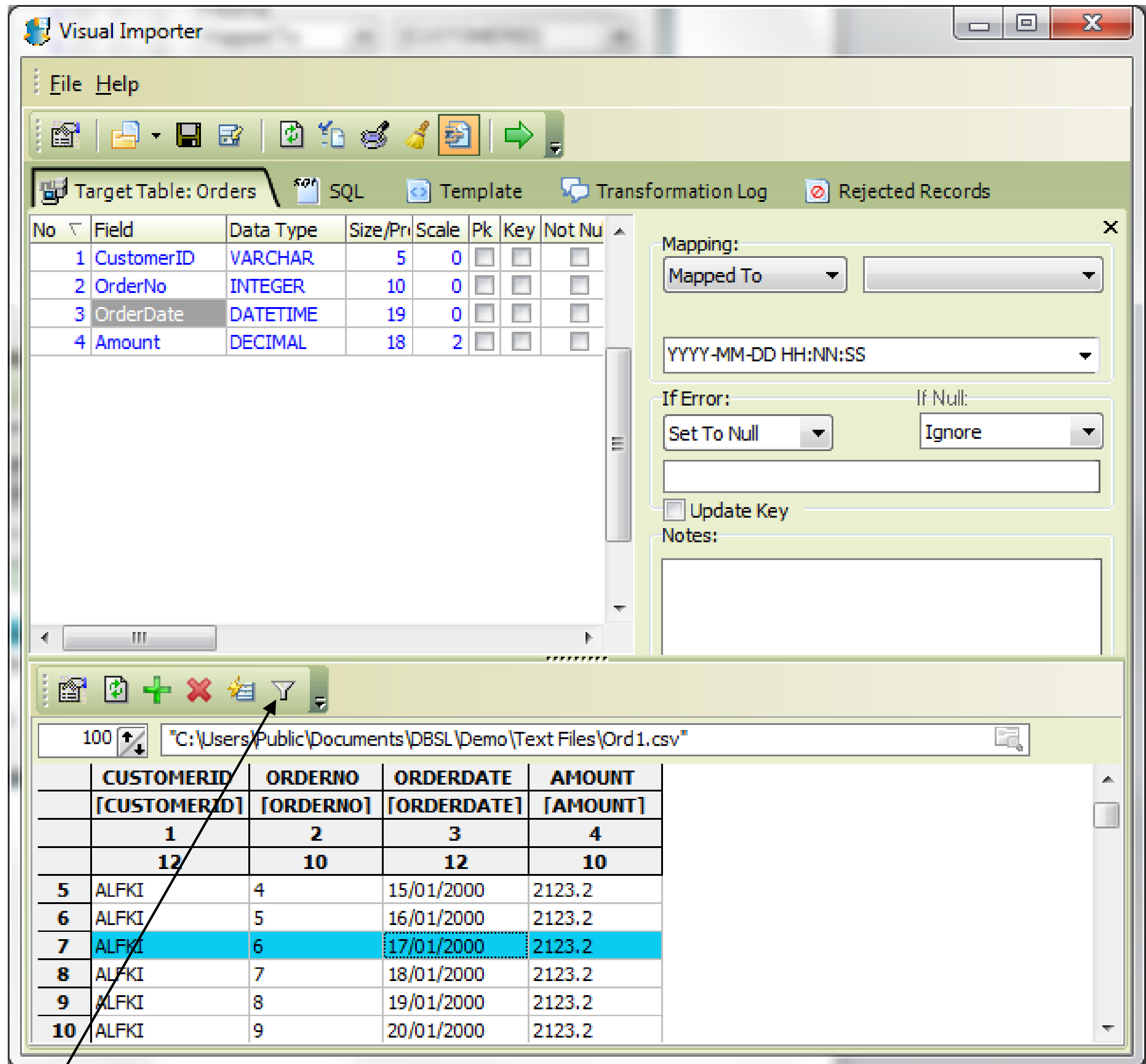
More complicated examples
Iif(StrToFloat([FLOAT_F])> StrToFloat([INTEGER_F]),1,2)
Trim('[CHAR_F]')

You may also use an Expression Editor.

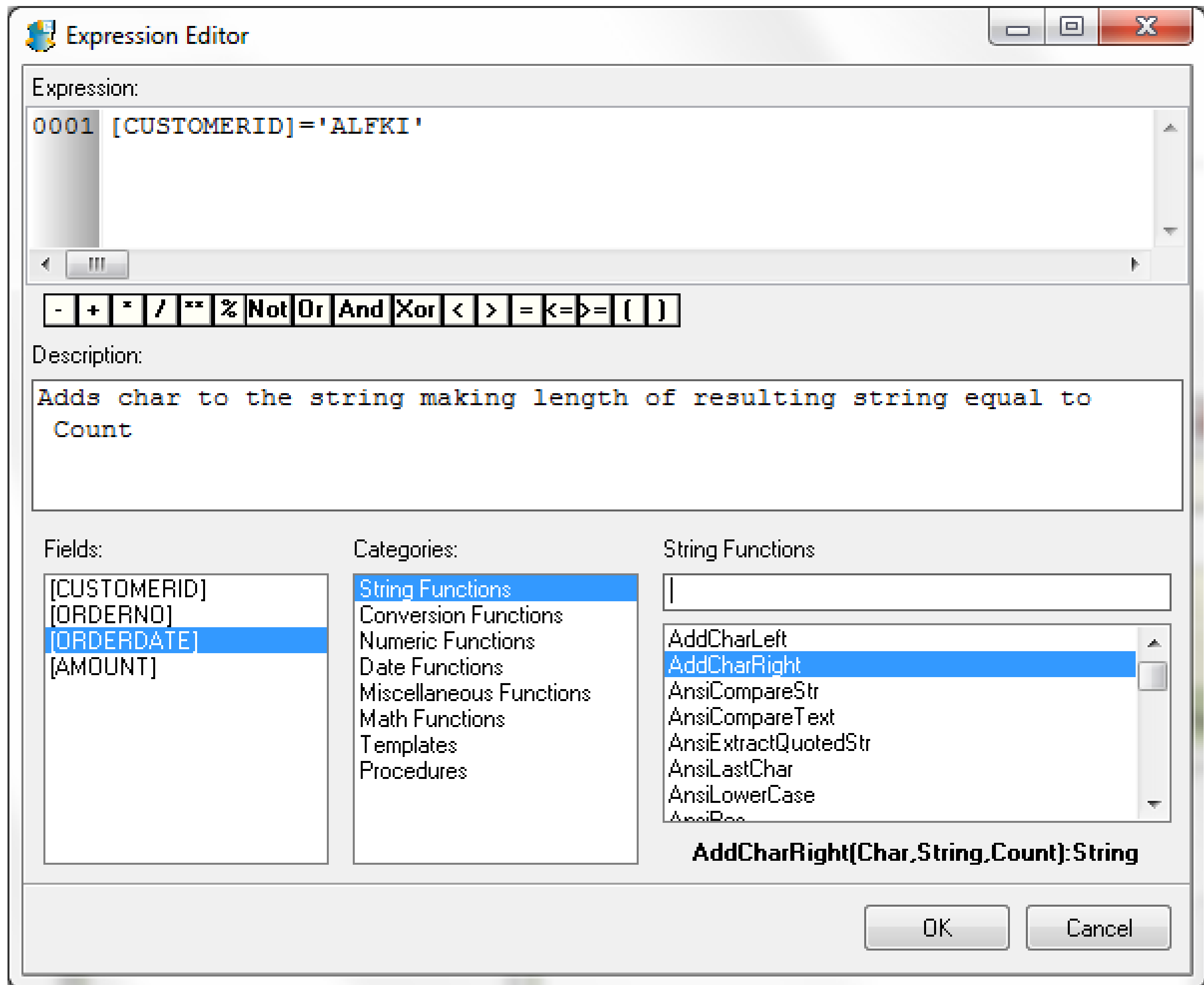


5.17 How to filter data

User may want to load data for specific customer
CUSTOMERID='ALFKI' Customer information



Filter



If you want to use multiple criteria use following example

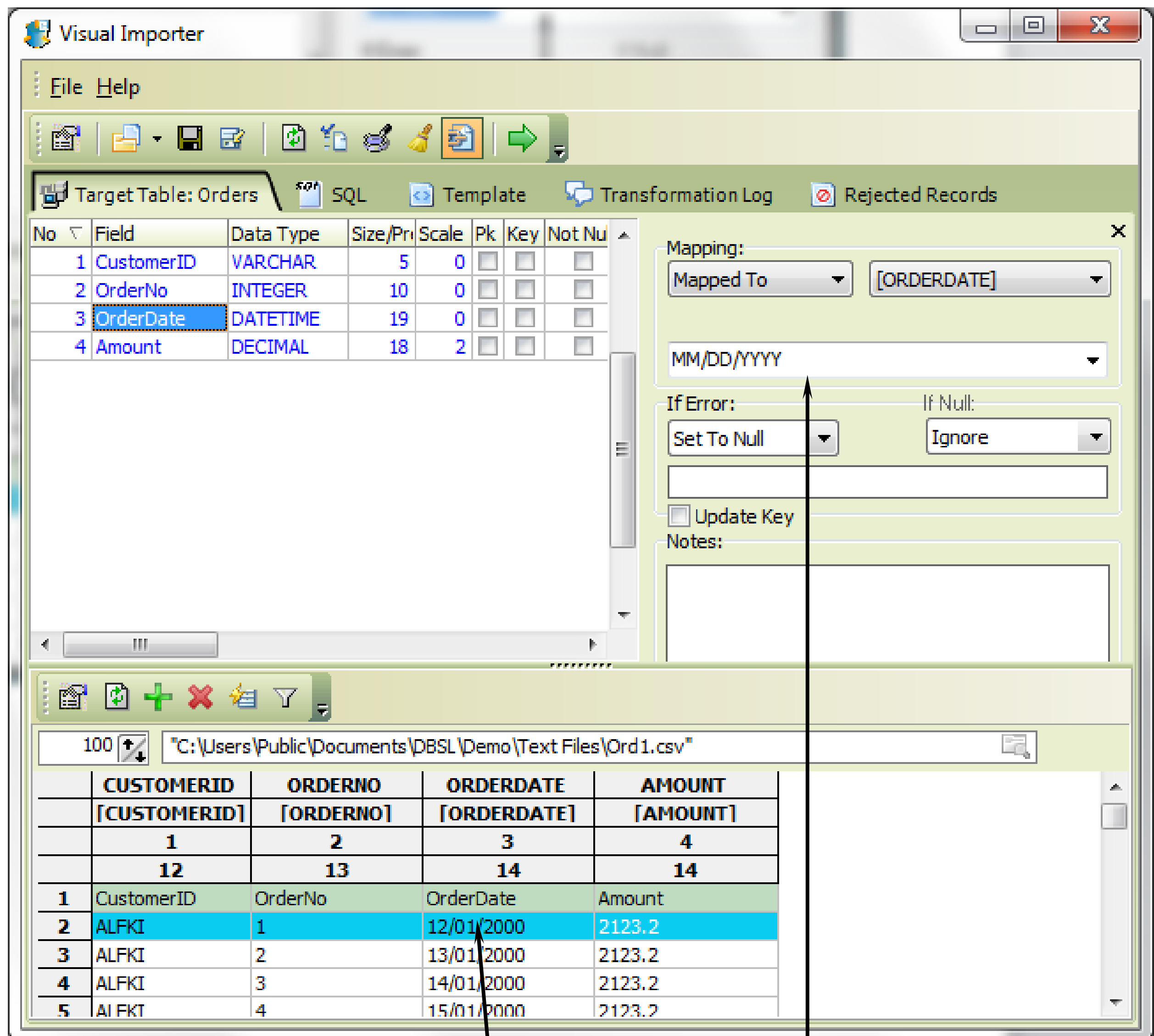
([RECORDTYPE]=1) or ([RECORDTYPE]=56)

Note:

Add brackets for complex expressions

5.18 Working with Date fields

In order to load data into date or time fields date format must be provided for source field
Visual Importer ETL automatically converts data into format required for the target database
Full list of date formats can be found in chapter 7

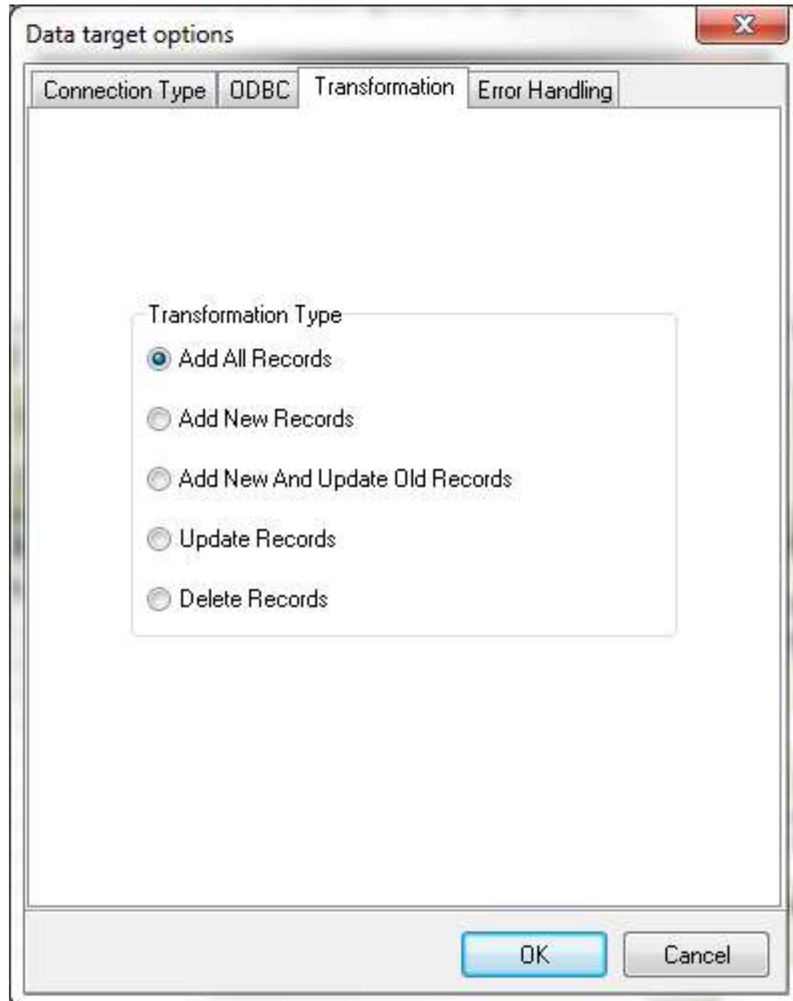


Date Format

Source Date Field

5.19 How to Update/Delete Records

In order to Update/Delete records user must specify an update key.



For the example provided below, **Visual Importer ETL** will execute the following SQL (Update key is CustomerId, OrderNo)

Add New And Update Old Records

```
Select count(*)
from [DEMO].[dbo].[orders]
where CustomerId=? And OrderNo=?
```

If any records found **Visual Importer ETL** will update them by executing

```
Update [DEMO].[dbo].[orders]
set orderdate=?,
    amount=?
where customerid=? And OrderNo=?
```

Copyright © 2011 DB Software Laboratory

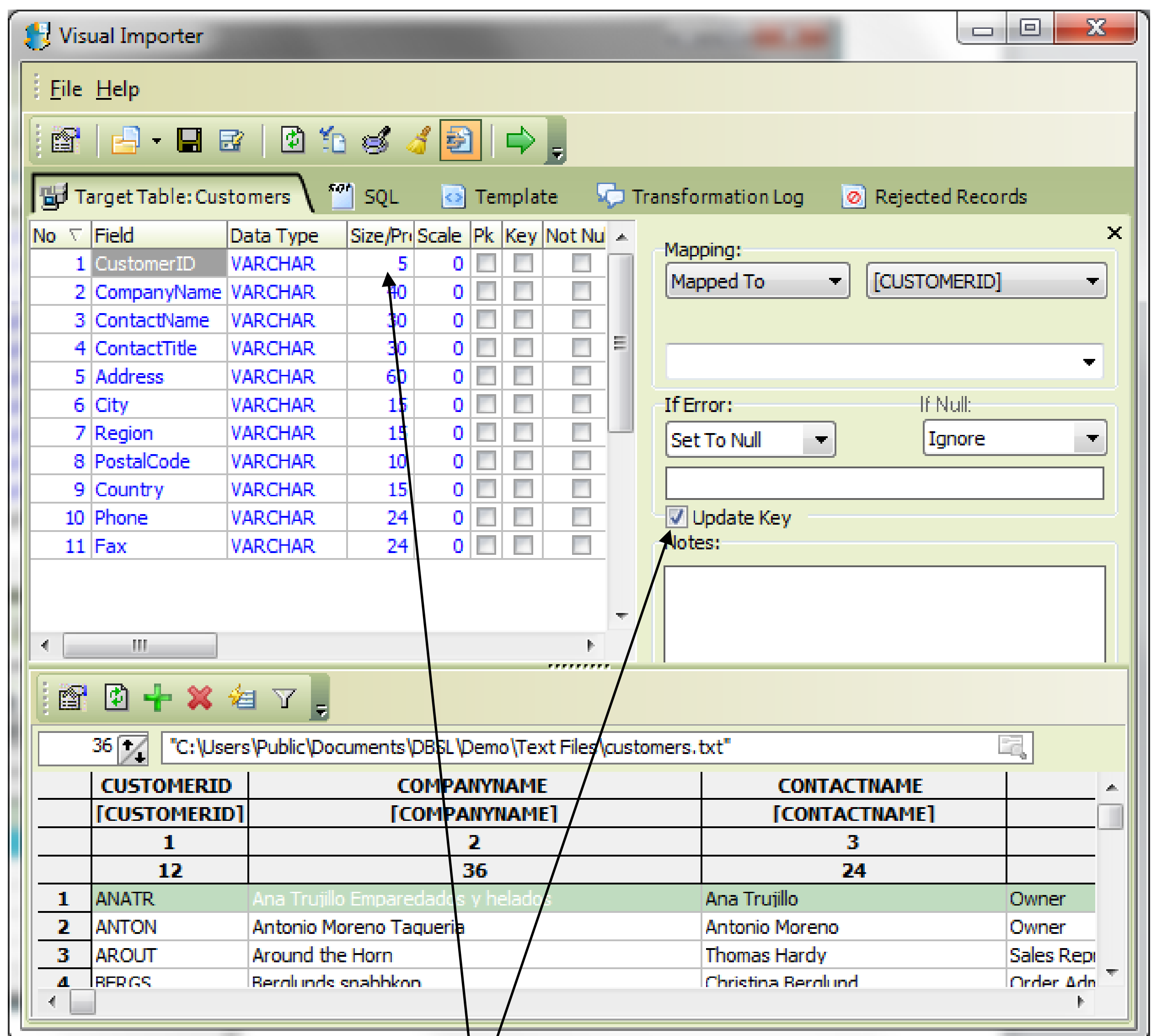
If no records found **Visual Importer ETL** will add new records

Update Records

```
Update [DEMO].[dbo].[orders]
set OrderDate=?,
    Amount=?
where CustomerId=? And OrderNo=?
```

Delete Records

```
Delete from [DEMO].[dbo].[orders]
Where CustomerId=? And OrderNo=?
```



Update key

5.20 MS SQL Server specific parameters

Note:

These options are only available when transformation type is Add All Records. When transformation type is different from Add All Records they are same as for ODBC connection

Check constraints

Ensure that any constraints on the destination table are checked during the bulk copy operation. By default, constraints are ignored.

Keep identity

Specify that there are values in the data file for an identity column.

Keep NULLS

Specify that any columns containing a null value should be retained as null values, even if a default value was specified for that column in the destination table.

Batch size

Specify the number of rows in a batch. The default is the entire data file.

The following values for the **Batch size** property have these effects:

If you set **Batch size** to zero, the data is loaded in a single batch. The first row that fails will cause the entire load to be cancelled, and the step fails.

If you set **Batch size** to one, the data is loaded a row at a time. Each row that fails is counted as one row failure. Previously loaded rows are committed.

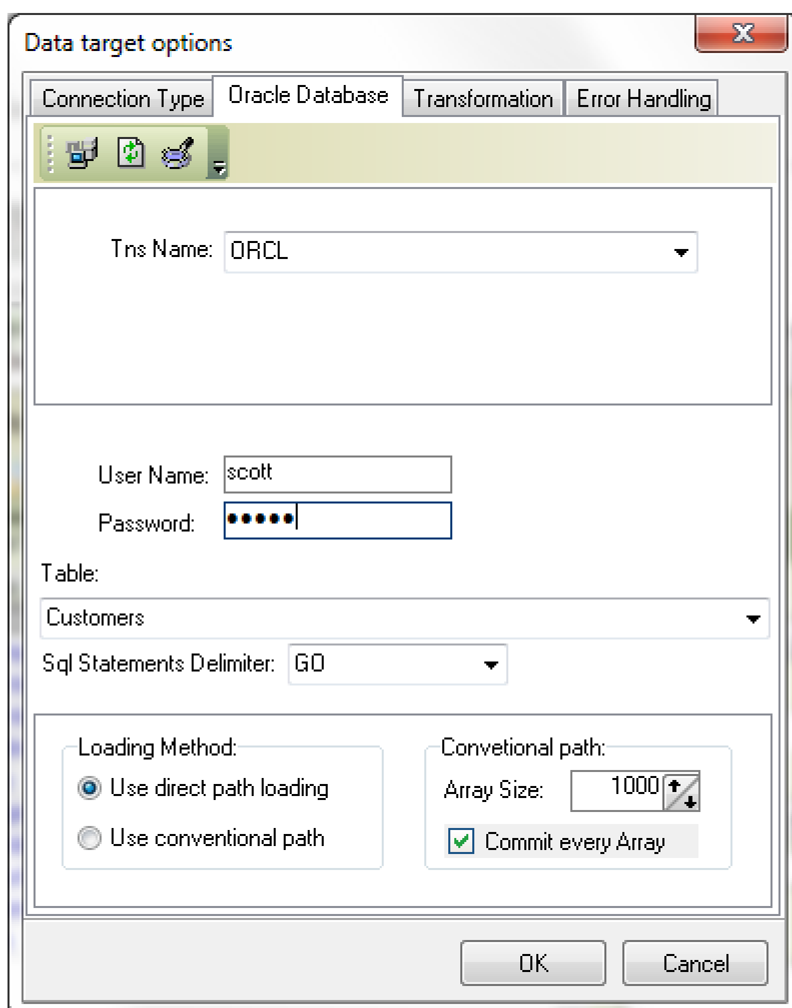
If you set **Batch size** to a value greater than one, the data is loaded one batch at a time. Any row that fails in a batch fails that entire batch; loading stops and the step fails. Rows in previously loaded batches are either committed or, if the step has joined the package transaction, provisionally retained in the transaction, subject to later commitment or rollback.

Copyright © 2011 DB Software Laboratory

5.21 Oracle specific parameters

Visual importer ETL supports three methods of loading data into oracle Direct path load, Conventional path load, Ole DB and ODBC
Direct-path load in Oracle is used when a session is reading buffers from disk directly into the PGA(opposed to the buffer cache in SGA). During direct-path INSERT operations, the database appends the inserted data after existing data in the table. Data is written directly into data files, bypassing the buffer cache. Free space in the existing data is not reused, and referential integrity constraints are ignored. You may prefer to use a direct path load when you have a large amount of data to load quickly and you want to load data in parallel for maximum performance, but there are alternative costs to be aware of.

With the **conventional path load method**, arrays of rows are inserted with standard SQL INSERT statements; integrity constraints and insert triggers are automatically applied. But when you load data with the direct path, **Visual Importer ETL** disables some integrity constraints and all database triggers.



The constraints that remain in force are:

NOT NULL
UNIQUE
PRIMARY KEY (unique-constraints on not-null columns)

The following constraints are automatically disabled by default:

CHECK constraints
Referential constraints (FOREIGN KEY)

For conventional path loading when **Commit every Array** is checked import works as follows:

Execute SQL before statement
Commit
Insert Array of records
Commit
Insert Array of records
Commit
Execute SQL after statement
Commit

When **Commit every Array** is not checked import is executed inside one big transaction:

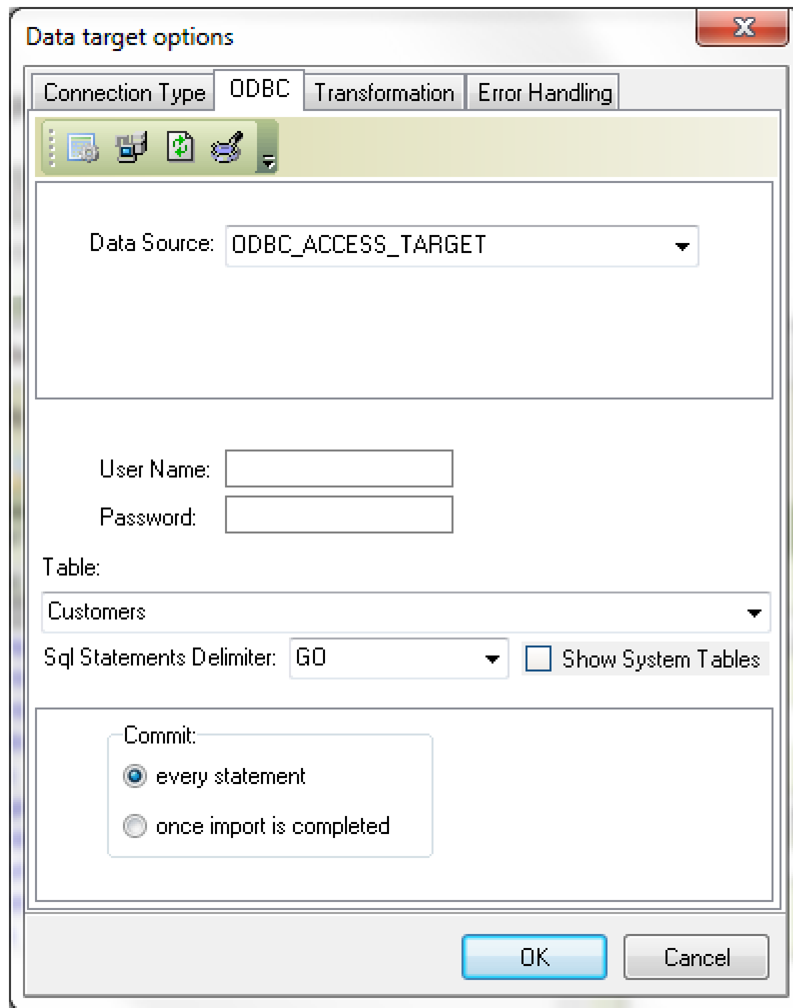
Start transaction
Execute SQL before statement
Insert Array of records
Insert Array of records
More inserts
Execute SQL after statement
Commit transaction

Note:

Loading Unicode data via direct path is not supported. If your database was not created with Unicode support but you have some NCHAR or NVARCHAR2 fields you may need to set array size to 1.

Copyright © 2011 DB Software Laboratory

5.22 ODBC specific parameters



When **Commit** is set to “every statement” import works as follows:

Execute SQL before statement
Commit
Insert one record
Commit
Insert one record
Commit
Execute SQL after statement
Commit

Note:
Most databases support this way of loading data including files

When **Commit** is set to “once import is completed” import is executed inside one big transaction:

Start transaction
Execute SQL before statement
Insert one record
Insert one record
More inserts
Execute SQL after statement
Commit transaction

Note:
Not all databases support this way of loading data.

5.22.1 ODBC connection strings

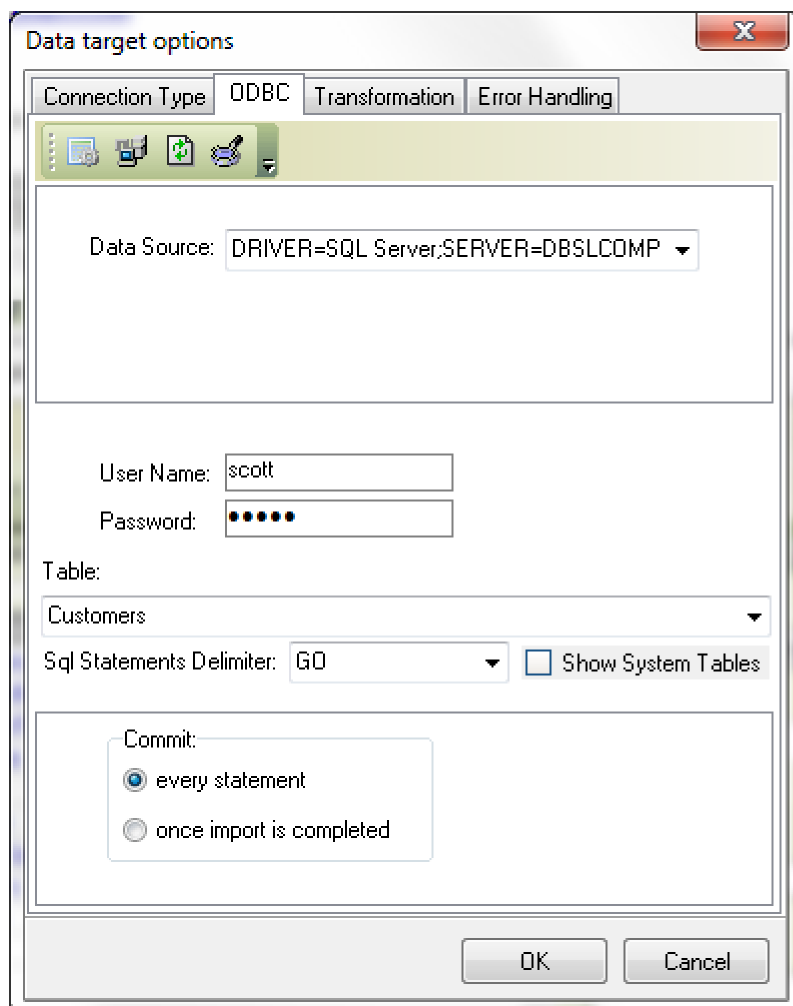
One of the benefits of using **Visual importer ETL** the ability to use ODBC connection strings.
It is no longer necessary to create ODBC DSN on the end user's computers.

For example for MS SQL Server connection string is:

Driver={SQL Native Client};Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPass;

More information about connection strings can be found at
http://www.connectionstrings.com

Note:
Leave user name and password blank and provide it within the connection string

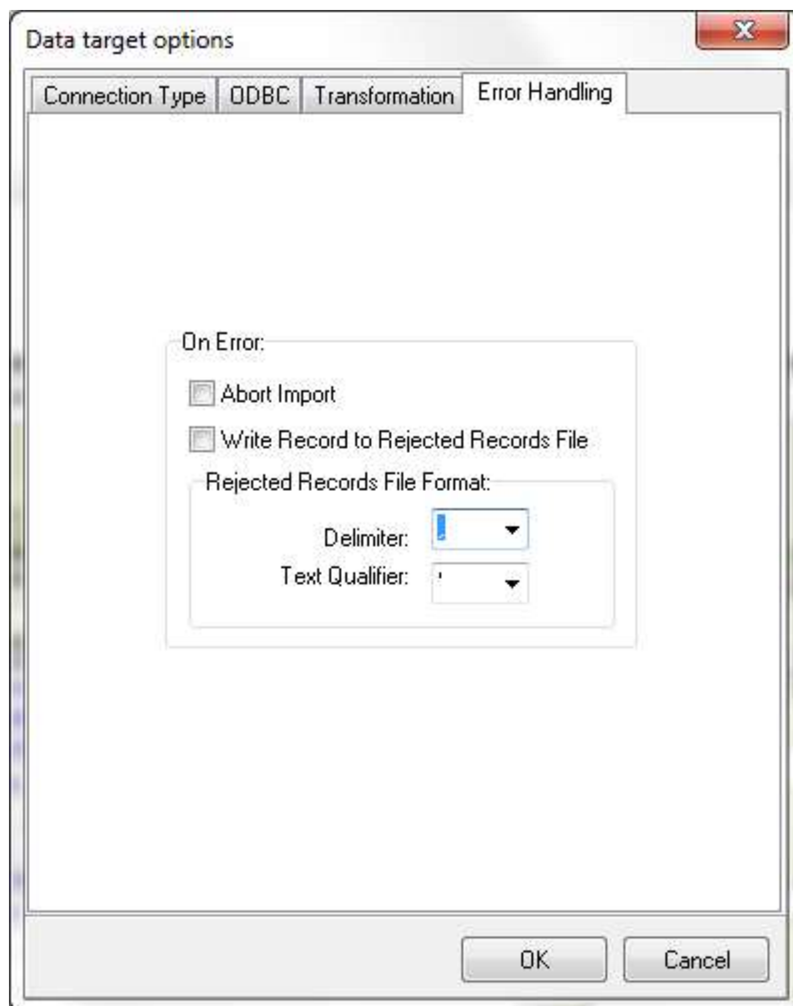


Copyright © 2011 DB Software Laboratory

5.23 Error Handling

Error handling tab defines format of rejected records file and allows user to abort import on error.

Note:
When “Abort Import” is selected import is immediately aborted after an error and **SQL After** script is not executed



Copyright © 2011 DB Software Laboratory

6. Scripting Language

6.1 The Basic Structure

Below is the basic structure that every Script/Calculation transformation must follow:

```
Var
  VariableName : VariableType;
  VariableName : VariableType;
  ...
// Some single line Comment if necessary

{
Multi line
Comment
}

Procedure ProcedureName;
  variables here if necessary
Begin
   Some Code;
End;

Function FunctionName(variableList): VariableType;
  variables here if necessary
Begin
  Some Code if necessary;
  Result := some expression
  More Code if necessary;
End;

... some more functions and procedures if necessary ...

Begin
  the main program block.
  Result:= some calculation
End.
```

Note:

The functions and procedures can appear in any order. The only requirement is that if one procedure or function uses another one, that latter one must have been defined already.

6.2 Variables

Declaring Variables

```
var                          // This starts a section of variables
  LineTotal : Integer;   // This defines an Integer variable called LineTotal
  First,Second : String; // This defines two variables to hold strings of text
```

Assigning Values to Variables

Variables are simply a name for a block of memory cells in main memory. If a value is assigned to a variable, that value must be of the same type as the variable, and will be stored in the memory address designated by the variable name. The assignment statement is the semicolon-equal **:=.**

- Variables must be declared at the beginning of the program, a procedure, or a function
- Variables must be initialized before they can be used.
- Variables can be reused as often as necessary. Their old value is simply overwritten by a new assignment.

Example:

```
Var
  i : Integer:     { variable name is i, type is integer)
 Begin
   i := 10;       { valid integer number assigned to variable i }
 End.
```

Variable Types

Numeric Data Types

Type	Storage size	Range
Byte	1	0 to 255
ShortInt	1	-127 to 127
Word	2	0 to 65,535
SmallInt	2	-32,768 to 32,767
LongWord	4	0 to 4,294,967,295
Cardinal	4*	0 to 4,294,967,295
LongInt	4	-2,147,483,648 to 2,147,483,647
Integer	4*	-2,147,483,648 to 2,147,483,647
Int64	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Single	4	7 significant digits, exponent -38 to +38
Currency	8	50+ significant digits, fixed 4 decimal places
Double	8	15 significant digits, exponent -308 to +308
Extended	10	19 significant digits, exponent -4932 to +4932

Assigning to and from number variables

Number variables can be assigned from other numeric variables, and expressions:

```
var
  Age      : Byte;      // Smallest positive integer type
  Books    : SmallInt;  // Bigger signed integer
  Salary      : Currency;  // Decimal used to hold financial amounts
  Expenses    : Currency;
  TakeHome : Currency;

begin
  Expenses     := 12345.67;  // Assign from a literal constant
  TakeHome   := Salary;    // Assign from another variable
  TakeHome   := TakeHome - Expenses;  // Assign from an expression
end;
```

Numerical operators

Number calculations, or expressions, have a number of primitive operators available:

+	Add one number to another
-	Subtract one number from another
*	Multiply two numbers
/	Divide one decimal number by another
div	Divide one integer number by another
mod	Remainder from dividing one integer by another

Copyright © 2011 DB Software Laboratory

When using these multiple operators in one expression, you should use round brackets to wrap around sub-expressions to ensure that the result is obtained. This is illustrated in the examples below:

```
 var
  myInt : Integer;  // Define integer and decimal variables
  myDec : Single;
begin
  myInt := 20;          // myInt is now 20
  myInt := myInt + 10;   // myInt is now 30
  myInt := myInt - 5;    // myInt is now 25
  myInt := myInt * 4;    // myInt is now 100
  myInt := 14 div 3;     // myInt is now 4   (14 / 3 = 4 remainder 2)
  myInt := 14 mod 3;     // myInt is now 2   (14 / 3 = 4 remainder 2)
  myInt := 12 * 3 - 4;   // myInt is now 32  (* comes before -)
  myInt := 12 * (3 - 4); // myInt is now -12 (brackets come before *)
  myDec := 2.222 / 2.0;  // myDec is now 1.111
end;
```

Character Types

```
var
  Str1 : Char;           // Holds a single character, small alphabet
  Str2 : WideChar;       // Holds a single character, International alphabet
  Str3 : AnsiChar;       // Holds a single character, small alphabet
  Str4 : ShortString;    // Holds a string of up to 255 Char's
  Str5 : String;         // Holds strings of Char's of any size desired
  Str6 : AnsiString;     // Holds strings of AnsiChar's any size desired
  Str7 : WideString;     // Holds strings of WideChar's of any size desired
```

Some simple text variable usage examples are given below:

+	Concatenates two strings together
=	Compares for string equality
<	Is one string lower in sequence than another
<=	Is one string lower or equal in sequence with another
>	Is one string greater in sequence than another
>=	Is one string greater or equal in sequence with another
<>	Compares for string inequality

Copyright © 2011 DB Software Laboratory

Variants

The Variant data type provides a flexible general purpose data type.

It can hold anything but structured data and pointers.

Variants are useful in very specific circumstances, where data types and their content are determined at run time rather than at compile time.

Example:

```
var
  myVar : Variant;
```

Date Variables

TDateTime

Description

The TDateTime type holds a date and time value.

It is stored as a Double variable, with the date as the integral part, and time as fractional part. The date is stored as the number of days since 30 Dec 1899. Quite why it is not 31 Dec is not clear. 01 Jan 1900 has a days value of 2.

Because TDateTime is actually a double, you can perform calculations on it as if it were a number. This is useful for calculations such as the difference between two dates.

Note:
No local time information is held with TDateTime - just the day and time values.

Example:
Finding the difference between two dates

```
var
  day1, day2 : TDateTime;
  diff : Double;
begin
  day1 := StrToDate('12/06/2002');
  day2 := StrToDate('12/07/2002');
  diff := day2 - day1;
  Result:='day2 - day1 = '+FloatToStr(diff)+' days';
end;
```

day2 - day1 = 30 days

Logical data types

These are used in conjunction with programming logic. They are very simple:

```
Var
  Log1 : Boolean;     // Can be 'True' or 'False'
```

Boolean variables are a form of enumerated type. This means that they can hold one of a fixed number of values, designated by name. Here, the values can be True or False.

Copyright © 2011 DB Software Laboratory

6.3 Logical Operations

Simple if then else

Here is an example of how the if statement works:

```
var
  number : Integer;
  text   : String;
begin
  number := Sqr(17);         // Calculate the square of 17
  if number > 400
  then text := '17 squared > 400'    // Action when if condition is true
  else text := '17 squared <= 400';  // Action when if condition is false
  result:=text;
end;
```

text is set to : '17 squared <= 400'

There are a number of things to note about the if statement. First that it spans a few lines - remember that statements can span lines - this is why it insists on a terminating **;**
Second, that the then statement does not have a terminating **;** -this is because it is part of the if statement, which is finished at the end of the else clause.

Third, that we have set the value of a text string when the If condition is successful - the Then clause - and when unsuccessful - the Else clause. We could have just done a then assignment:

```
if number > 400
then text := '17 squared > 400';
```

Note that here, the then condition is not executed (because 17 squared is not > 400), but there is no else clause. This means that the if statement simply finishes without doing anything.

Note also that the then clause now has a terminating **;** to signify the end of the if statement.

Compound if conditions, and multiple statements
We can have multiple conditions for the if condition. And we can have more than one statement for the then and else clauses. Here are some examples:

```
If (condition1) And (condition2)   // Both conditions must be satisfied
  then
    begin
      statement1;
      statement2;
      ...
    end             // Notice no terminating ';' - still part of 'if'
  else
```

```
begin
  statement3;
  statement4;
  ...
end;
```

We used And to join the if conditions together - both must be satisfied for the then clause to execute. Otherwise, the else clause will execute. We could have used a number of different logical primitives, of which And is one, covered under logical primitives below.

Nested if statements:

There is nothing to stop you using if statements as the statement of an if statement. Nesting can be useful, and is often used like this:

```
 if condition1
then statement1
else if condition2
     then statement2
     else statement3;
```

However, too many nested if statements can make the code confusing. The Case statement, discussed below, can be used to overcome a lot of these problems.

Logical primitives

Before we introduce these, it is appropriate to introduce the Boolean data type. It is an enumerated type that can have one of only two values: True or False. We will use it in place of a condition in the if clauses below to clarify how they work:

```
begin
 if false And false
 then Result:='false and false = true';

 if true And false
 then Result:= 'true  and false = true';

 if false And true
 then Result:= 'false and true  = true';

 if true And true
 then Result:= 'true  and true  = true';

 if false Or false
 then Result:= 'false or  false = true';
```

```
  if true Or false
  then Result:= 'true  or  false = true';

  if false Or true
  then Result:= 'false or  true  = true';

  if true Or true
  then Result:= 'true  or  true  = true';

  if false Xor false
  then Result:= 'false xor false = true';

  if true Xor false
  then Result:= 'true  xor false = true';

  if false Xor true
  then Result:= 'false xor true  = true';

  if true Xor true
  then Result:= 'true  xor true  = true';

  if Not false
  then Result:= 'not false  = true';

  if Not true
  then Result:= 'not true  = true';
end;
```

```
true  and true  = true
false or  true   = true
true  or  false  = true
true  or  true   = true
false xor true  = true
true  xor false = true
not false = true
```

Note that the Xor primitive returns true when one, but not both of the conditions are true.

Case statements

The If statement is useful when you have a simple two way decision. Ether you go one way or another way. Case statements are used when you have a set of 3 or more alternatives.

A simple numerical case statement:

```
var
  i : Integer;
```

Copyright © 2011 DB Software Laboratory

```
begin
  i := [F1];
  Case i of
    15 : Resut := ('Number was fifteen');
    16 : Resut := ('Number was sixteen');
    17 : Resut := ('Number was seventeen');
    18 : Resut := ('Number was eighteen');
    19 : Resut := ('Number was nineteen');
    20 : Resut := ('Number was twenty');
  end;
end;
```

Number was fifteen

The case statement above routes the processing to just one of the statements. OK, the code is a bit silly, but it is used to illustrate the point.

Using the otherwise clause

Supposing we were not entirely sure what value our case statement was processing? Or we wanted to cover a known set of values in one fell swoop? The Else clause allows us to do that:

```
var
  i : Integer;
begin
  i := [F1];
  Case i of
    15 : Resut := ‘Number was fifteen';
    16 : Resut := 'Number was sixteen';
    17 : Resut := 'Number was seventeen';
    18 : Resut := 'Number was eighteen';
    19 : Resut := 'Number was nineteen';
    20 : Resut := 'Nuumber was twenty';
  else
    Resut := 'Unexpected number‘;
  end;
end;
```

Unexpected number : 10

Copyright © 2011 DB Software Laboratory

6.4 Repeating sets of commands

Why loops are used in programming

One of the main reasons for using computers is to save the tedium of many repetitive tasks. One of the main uses of loops in programs is to carrry out such repetitive tasks. A loop will execute one or more lines of code (statements) as many times as you want.

Your choice of loop type depends on how you want to control and terminate the looping.

The For loop

This is the most common loop type. For loops are executed a fixed number of times, determined by a count. They terminate when the count is exhausted. The count (loop) is held in a variable that can be used in the loop. The count can proceed upwards or downwards, but always does so by a value of 1 unit. This count variable can be a number or even an enumeration.

Counting up

Here is a simple example counting up using numeric values:

```
var
  count : Integer;
begin
  For count := 1 to 5 do
   Result:= 'Count is now '+IntToStr(count);
end;
```

Counting down

Here is a simple example counting up using numeric values:

```
var
  count : Integer;
begin
  For count := 5 downto 1 do
   Result:= 'Count is now '+IntToStr(count);
end;
```

The For statements in the examples above have all executed one statement. If you want to execute more than one, you must enclose these in a Begin and End pair.

Copyright © 2011 DB Software Laboratory

The Repeat loop

The Repeat loop type is used for loops where we do not know in advance how many times we will execute. For example, when we keep asking a user for a value until one is provided, or the user aborts. Here, we are more concerned with the loop termination condition.

Repeat loops always execute at least once. At the end, the Until condition is checked, and the loop aborts of condition works out as true.

A simple example

```
var
  stop : Boolean;         // Our exit condition flag
  i : Integer;
begin
  i := 1;
  exit := False;          // do not exit until we are ready
  repeat
    i := i+1;             // Increment a count
    if Sqr(i) > 99
    then stop:= true;     // Exit if the square of our number exceeds 99
  until stop;             // Shorthand for 'until exit := true'
 result:=I;
end;
```

Upon exit, i will be 10 (since Sqr(10) > 99)

Here we exit the repeat loop when a Boolean variable is true. Notice that we use a shorthand - just specifying the variable as the condition is sufficient since the variable value is either true or false.

Using a compound condition

```
 var
  i : Integer;
 begin
  i := 1;
  repeat
     i := i+1;          // Increment a count
  until (Sqr(i) > 99) or (Sqrt(i) > 2.5);
  result:=i;
end;
```

Upon exit, i will be 7 (since Sqrt(7) > 2.5)

Notice that compound statements require separating brackets. Notice also that Repeat statements can accommodate multiple statements without the need for a begin/end pair. The repeat and until clauses form a natural pairing.

Copyright © 2011 DB Software Laboratory

While loops

While loops are very similar to Repeat loops except that they have the exit condition at the start. This means that we use them when we wish to avoid loop execution altogether if the condition for exit is satisfied at the start.

```
Var
  i : Integer;
 begin
  i := 1;
  while (Sqr(i) <= 99) and (Sqrt(i) <= 2.5) do
     i := i+1;              // Increment a count
  result:=i;
end;
```

Upon exit, i will be 7 (since Sqrt(7) > 2.5)

Notice that our original Repeat Until condition used Or as the compound condition joiner - we continued until either condition was met. With our While condition, we use And as the joiner - we continue whilst neither condition is met. Have a closer look to see why we do this. The difference is that we repeat an action until something or something else happens. Whereas we keep doing an action while neither something nor something else have happened.

Copyright © 2011 DB Software Laboratory

6.5 Functions

Functions provide a flexible method to apply one formula many times to possibly different values. They are comparable to procedures but

- functions are of always of a certain type
- functions usually have one or more input variable(s)
- the function name must appear at least once inside the definition

The general form of the function statement looks like this:

Function FunctionName(VariableName: VariableType): VariableType;
Begin
some code, if necessary;
Result := some computation;
more code if necessary;
End;

6.6 Script Examples

1. Strings Concatenation

```
Begin
 Result:= [F001] + [F002];
End;
```

2. Adding one value to another

```
Begin
 Result:= StrToFloat([F001]) + StrToFloat([F002]);
End;
```

2. If Statement

```
Begin
  If  [F002]='' then
    Result:= 0
  else If  [F002]=0 then
    Result:= 0
  Else
    Result:= StrToFloat([F001]) mod StrToFloat([F002]);
End;
```

3. Variables

```
Var MyVariable : integer;
Begin
  MyVariable:=10;
  Result :=StrToFloat([F001]) mod MyVariable;
end;
```

6.7 Passing variables between objects

There are two calculation functions which can be used to pass data from one object into another:

1. SetVariable
2. GetVariable

Example:

```
begin
  SetVariable('VariableName',[F0001]);
end;

begin
  Result:= GetVariable('VariableName');
end;
```

Sequence:

```
Var I : Integer;
    s: string;
begin
  I:=1;
  S:=GetVariable('I');
  if s='' then
    SetVariable('I',I)
  else
   begin
    I:=GetVariable('I');
    I:=I+1;
    SetVariable('I',I);
   end;
  Result:=I;
end;
```

Copyright © 2011 DB Software Laboratory

6.7 Supported Functions List

6.7.1 String Functions

AddCharLeft(Char,S,Count)

AddCharLeft returns a string left-padded to Length with characters Char

AddCharRight(Char,S,Count)

AddCharRight returns a string right-padded to Length with characters Char

RightString(S,Count)

RightString returns the trailing characters of String up to a length of Count characters

Replace(S,OldPattern,NewPattern)

Replace replaces all occurrences of the OldPattern by NewPattern within the String

SubString(S,Index,Count)

SubString returns a substring containing Count Characters or elements starting from Index.

LeftString(S,Count)

LeftString returns the leading characters of String up to a length of Count characters

MakeString(Char,Count)

MakeString returns a string of Count filled with character Char.

DelSpaces(S)

DelSpaces returns string with all spaces deleted except one.
"two spaces"->"two spaces

Delete(S,Index,Count)

Delete removes a substring of Count characters from string S starting with S[Index]. S is a string-type variable. Index and Count are integer-type expressions. If index is larger than the length of the string or less than 1, no characters are deleted. If count specifies more characters than remain starting at the index, Delete removes the rest of the string. If count is less than or equal to 0, no characters are deleted

Insert(Substr,Dest,Index)

Insert merges Source into S at the position S[index]. Source is a string-type expression. S is a string-type variable of any length. Index is an integer-type expression. It is a character index and not a byte index. If Index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to the length of the string, turning the operation into an append. If the Source parameter is an empty string, Insert does nothing

ProperCase(S)

ProperCase returns string, with the first letter of each word in uppercase and all other letters in lowercase "proper case"->"Proper Case"

UpperCase(S)

UpperCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiUpperCase.

LowerCase(S)

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase.

AnsiUpperCase(S)

AnsiUpperCase returns a string that is a copy of S, converted to upper case.

AnsiLowerCase(S)

AnsiLowerCase returns a string that is a copy of the given string converted to lower case.

AnsiCompareStr(S1,S2)

AnsiCompareStr compares S1 to S2, with case sensitivity.
The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

AnsiCompareText(S1,S2)

AnsiCompareText compares S1 to S2, without case sensitivity. AnsiCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

Copyright © 2011 DB Software Laboratory

AnsiStrLIComp (S1,S2,MaxLen)

AnsiStrLIComp compares S1 to S2, without case sensitivity. If S1 or S2 is longer than MaxLen characters, AnsiStrLIComp only compares up to the first MaxLen characters.
The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

AnsiLastChar(S)

Call AnsiLastChar to obtain the last character in a string.

Trim(S)

Trim removes leading and trailing spaces and control characters from the given string S.

TrimLeft(S)

TrimLeft returns a copy of the string S with leading spaces and control characters removed.

TrimRight(S)

TrimRight returns a copy of the string S with trailing spaces and control characters removed.

QuotedStr(S)

Use QuotedStr to convert the string S to a quoted string. A single quote character (') is inserted at the beginning and end of S, and each single quote character in the string is repeated.

AnsiQuotedStr(S,Quote)

Use AnsiQuotedStr to convert a string (S) to a quoted string, using the provided Quote character. A Quote character is inserted at the beginning and end of S, and each Quote character in the string is doubled.

AnsiExtractQuotedStr(S,Quote)

AnsiExtractQuotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The Quote parameter defines what character to use as a quote character. If the first character in S is not the value of the Quote parameter, AnsiExtractQuotedStr returns an empty string.

Copyright © 2011 DB Software Laboratory

Copy(S,Index,Count)

S is an expression of a string or dynamic-array type. Index and Count are integer-type expressions. Copy returns a substring or sub array containing Count characters or elements starting at S[Index]. The substring or sub array is a unique copy (that is, it does not share memory with S, although if the elements of the array are pointers or objects, these are not copied as well.) If Index is larger than the length of S, Copy returns an empty string or array. If Count specifies more characters or array elements than are available, only the characters or elements from S[Index] to the end of S are returned.

CRC16(String,Format)

Returns CRC16 checksum of a String, Format 0=HEX ,1=Base64')

CRC24(String,Format)

Returns CRC24 checksum of a String, Format 0=HEX ,1=Base64')

CRC32(String,Format)

Returns CRC32 checksum of a String, Format 0=HEX ,1=Base64')

Adler32(String,Format)

Returns Adler32 checksum of a String, Format 0=HEX ,1=Base64')

CRC64(String,Format)

Returns CRC64 checksum of a String, Format 0=HEX ,1=Base64')

eDonkey(String,Format)

Returns eDonkey checksum of a String, Format 0=HEX ,1=Base64')

eMule(String,Format)

Returns eMule checksum of a String, Format 0=HEX ,1=Base64')

MD4(String,Format)

Returns MD4 checksum of a String, Format 0=HEX ,1=Base64')

MD5(String,Format)

Returns MD5 checksum of a String, Format 0=HEX ,1=Base64')

Copyright © 2011 DB Software Laboratory

RIPEMD160(String,Format)

Returns RIPEMD160 checksum of a String, Format 0=HEX ,1=Base64')

SHA1(String,Format)

Returns SHA1 checksum of a String, Format 0=HEX ,1=Base64')

SHA224(String,Format)

Returns SHA224 checksum of a String, Format 0=HEX ,1=Base64')

SHA256(String,Format)

Returns SHA256 checksum of a String, Format 0=HEX ,1=Base64')

SHA384(String,Format)

Returns SHA384 checksum of a String, Format 0=HEX ,1=Base64')

SHA512(String,Format)

Returns SHA512 checksum of a String, Format 0=HEX ,1=Base64')

Whirlpool(String,Format)

Returns Whirlpool checksum of a String, Format 0=HEX ,1=Base64')

Copyright © 2011 DB Software Laboratory

6.7.2 Conversion Functions

IntToStr(S)

IntToStr converts an integer into a string containing the decimal representation of that number.

IntToHex(I,Digits)

IntToHex converts a number into a string containing the number's hexadecimal (base 16) representation. Value is the number to convert. Digits indicates the minimum number of hexadecimal digits to return.

StrToInt(S)

StrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt raises an exception.

StrToIntDef(S,Default)

StrToIntDef converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToIntDef returns Default.

FloatToStr(F)

FloatToStr converts the floating-point value given by Value to its string representation. The conversion uses general number format with 15 significant digits.

StrToFloat(S)

Use StrToFloat to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

IntegerToString(Integer)

IntegerToString converts integer value to a string value.

NumberToString(Float)

NumberToString converts float value to an string value.

StringToInteger(S)

StringToInteger converts string value to an integer value.

StringToNumber(S)

StringToNumber converts string value to a float value.

6.7.3 File System Functions

FileAge(FileName)

Call FileAge to obtain the OS timestamp of the file specified by FileName. The return value can be converted to a TDateTime object using the FileDateToDateTime function. The return value is -1 if the file does not exist.

FileExists(FileName)

FileExists returns true if the file specified by FileName exists. If the file does not exist, FileExists returns false.

DeleteFile(FileName)

DeleteFile deletes the file named by FileName from the disk. If the file cannot be deleted or does not exist, the function returns false.

RenameFile(OldFile,NewFile)

RenameFile attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns true. If RenameFile cannot rename the file (for example, if the application does not have permission to modify the file), it returns false.

ChangeFileExt(FileName,EXT)

ChangeFileExt takes the file name passed in FileName and changes the extension of the file name to the extension passed in Extension. Extension specifies the new extension, including the initial dot character.

ChangeFileExt does not rename the actual file, it just creates a new file name string.

ExtractFilePath(FileName)

The resulting string is the leftmost characters of FileName, up to and including the colon or backslash that separates the path information from the name and extension. The resulting string is empty if FileName contains no drive and directory parts.

ExtractFileDir(FileName)

Extracts Directory part from the File Name provided

ExtractFileDrive(FileName)

ExtractFileDrive returns a string containing the drive portion of a fully qualified path name for the file passed in the FileName. For file names with drive letters, the result is in the form

Copyright © 2011 DB Software Laboratory

"drive". For file names with a UNC path the result is in the form "\servername\sharename". If the given path contains neither style of path prefix, the result is an empty string.

ExtractFileName(FileName)

The resulting string is the rightmost characters of FileName, starting with the first character after the colon or backslash that separates the path information from the name and extension. The resulting string is equal to FileName if FileName contains no drive and directory parts.

ExtractFileExt(FileName)

Use ExtractFileExt to obtain the extension from a file name.

ExpandFileName(FileName)

ExpandFileName converts the relative file name into a fully qualified path name. ExpandFileName does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

ExpandUNCFileName(FileName)

ExpandUNCFileName returns the fully-qualified file name for a specified file name.

ExtractRelativePath(FileName)

Call ExtractRelativePath to convert a fully qualified path name into a relative path name. The DestName parameter specifies file name (including path) to be converted. BaseName is the fully qualified name of the base directory to which the returned path name should be relative. BaseName may or may not include a file name, but it must include the final path delimiter.

DiskFree(Drive)

DiskFree returns the number of free bytes on the specified drive, where 0 = Current, 1 = A, 2 = B, and so on.

DiskSize(Drive)

DiskSize returns the size in bytes of the specified drive, where 0 = Current, 1 = A, 2 = B, etc. DiskSize returns -1 if the drive number is invalid.

GetCurrentDir(Directory)

GetCurrentDir returns the fully qualified name of the current directory.

SetCurrentDir(Directory)

The SetCurrentDir function sets the current directory. The return value is true if the current directory was successfully changed, or false if an error occurred.

CreateDir(Directory)
CreateDir creates a new directory. The return value is true if a new directory was successfully created, or false if an error occurred.

RemoveDir(Directory)

Call RemoveDir to remove the directory specified by the Dir parameter. The return value is true if a new directory was successfully deleted, false if an error occurred. The directory must be empty before it can be successfully deleted.

FileCRC16(String,Format)

Returns CRC16 checksum of a File, Format 0=HEX ,1=Base64')

FileCRC24(String,Format)

Returns CRC24 checksum of a File, Format 0=HEX ,1=Base64')

FileCRC32(String,Format)

Returns CRC32 checksum of a File, Format 0=HEX ,1=Base64')

FileAdler32(String,Format)

Returns Adler32 checksum of a File, Format 0=HEX ,1=Base64')

FileCRC64(String,Format)

Returns CRC64 checksum of a File, Format 0=HEX ,1=Base64')

FileeDonkey(String,Format)

Returns eDonkey checksum of a File, Format 0=HEX ,1=Base64')

FileeMule(String,Format)

Returns eMule checksum of a File, Format 0=HEX ,1=Base64')

FileMD4(String,Format)

Returns MD4 checksum of a File, Format 0=HEX ,1=Base64')

FileMD5(String,Format)

Returns MD5 checksum of a File, Format 0=HEX ,1=Base64')

Copyright © 2011 DB Software Laboratory

FileRIPEMD160(String,Format)

Returns RIPEMD160 checksum of a File, Format 0=HEX ,1=Base64')

FileSHA1(String,Format)

Returns SHA1 checksum of a File, Format 0=HEX ,1=Base64')

FileSHA224(String,Format)

Returns SHA224 checksum of a File, Format 0=HEX ,1=Base64')

FileSHA256(String,Format)

Returns SHA256 checksum of a File, Format 0=HEX ,1=Base64')

FileSHA384(String,Format)

Returns SHA384 checksum of a File, Format 0=HEX ,1=Base64')

FileSHA512(String,Format)

Returns SHA512 checksum of a File, Format 0=HEX ,1=Base64')

FileWhirlpool(String,Format)

Returns Whirlpool checksum of a File, Format 0=HEX ,1=Base64')

Copyright © 2011 DB Software Laboratory

6.7.4 Date and Time Functions

Day(Date,Format)

Use Day to get the day part of a date value.

Day('01012003','DDMMYYYY')

Hour(Date,Format)

Use Hour to get the hour part of a date value.

Hour('01012003','DDMMYYYY')

Minute(Date,Format)

Use Minute to get the minute part of a date value.

Minute('01012003','DDMMYYYY')

Month(Date,Format)

Use Month to get the month part of a date value.

Month('01012003','DDMMYYYY')

Second(Date,Format)

Use Second to get the second part of a date value.

Second('01012003','DDMMYYYY')

Year(Date,Format)

Use Year to get the year part of a date value.

Year('01012003','DDMMYYYY')

DayS(Date,Format)

Use DayS to get the day part of a date value as string.

DayS('01012003','DDMMYYYY')

HourS(Date,Format)

Use HourS to get the hour part of a date value as string.

Copyright © 2011 DB Software Laboratory

HourS('01012003','DDMMYYYY')

MinuteS(Date,Format)

Use MinuteS to get the minute part of a date value as string.

MinuteS('01012003','DDMMYYYY')

MonthS(Date,Format)

Use MonthS to get the month part of a date value as string.

MonthS('01012003','DDMMYYYY')

SecondS(Date,Format)

Use SecondS to get the second part of a date value as string.

SecondS('01012003','DDMMYYYY')

YearS(Date,Format)

Use YearS to get the year part of a date value as string.

YearS('01012003','DDMMYYYY')

IncDateS (Date,Format,ChangeType,Increment)

ChangeType: YEAR,MONTH,WEEK,DAY,HOUR,MINUTE,SECOND

Use IncDateS to Increase ChangeType part of a date value by an Increment.

IncDateS ('01012003','DDMMYYYY', 'YEAR',1)

DecDateS(Date,Format,ChangeType,Decrement)

ChangeType: YEAR,MONTH,WEEK,DAY,HOUR,MINUTE,SECOND

Use DecDateS to Decrease ChangeType part of a date value by an Decrement.

DecDateS ('01012003','DDMMYYYY', 'YEAR',1)

EncodeDate(Year,Month,Day)

EncodeDate returns a TDateTime value from the values specified as the Year, Month, and Day parameters. The year must be between 1 and 9999. Valid Month values are 1 through 12. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

EncodeTime(Hour,Min,Sec,MSec)

EncodeTime encodes the given hour, minute, second, and millisecond into a TDateTime value. Valid Hour values are 0 through 23. Valid Min and Sec values are 0 through 59. Valid MSec values are 0 through 999. If the specified values are not within range, EncodeTime raises an EConvertError exception. The resulting value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

DayOfWeek(D)

DayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

Date

Use Date to obtain the current local date as a TDateTime value. The time portion of the value is 0 (midnight).

Time

Use Time to return the current time as a TDateTime value. The two functions are completely equivalent.

Now

Returns the current date and time.

IncMonth(D)

IncMonth returns the value of the Date parameter, incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous. If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month. The time of day specified by the Date parameter is copied to the result.

IsLeapYear(D)

Call IsLeapYear to determine whether the year specified by the Year parameter is a leap year. Year specifies the calendar year. Use YearOf to obtain the value of Year for IsLeapYear from a TDateTime value.

DateToStr(D)

Use DateToStr to obtain a string representation of a date value that can be used for display purposes.

TimeToStr(D)

TimeToStr converts the Time parameter, a TDateTime value, to a string.

DateTimeToStr(D)

Converts a TDateTime value to a string.

StrToDate(S)

Call StrToDate to parse a string that specifies a date. If S does not contain a valid date, StrToDate raises an exception.

StrToTime(S)

Call StrToTime to parse a string that specifies a time value. If S does not contain a valid time, StrToTime raises an exception.

StrToDateTime(S)

Call StrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTime raises an exception.

FormatDateTime(Format,DateTime)

FormatDateTime formats the TDateTime value given by DateTime using the format given by Format. See the table below for information about the supported format strings.

6.7.5 Numeric Functions

Round(Float,Integer)

Use Round to round Value to a specified power of ten.
The following examples illustrate the use of Round:

Expression	Value
Round(1234567, 3)	1234000
Round(1.234, -2)	1.23
Round(1.235, -2)	1.24
Round(1.245, -2)	1.24

Sign(I)

Use Sign to test the sign of a numeric value.
Sign returns
0 if AValue is zero.
1 if AValue is greater than zero.
-1 if AValue is less than zero.

Abs(X)

Returns an absolute value.

Trunc(X)

Truncates a real number to an integer.

Ceil(X)

Call Ceil to obtain the lowest integer greater than or equal to X. The absolute value of X must be less than MaxInt. For example: Ceil(-2.8) = -2 Ceil(2.8) = 3 Ceil(-1.0) = -1

Floor(X)

Call Floor to obtain the highest integer less than or equal to X. For example: Floor(-2.8) = -3 Floor(2.8) = 2 Floor(-1.0) = -1

RandomRange(AFrom,ATo)

RandomRange returns a random integer from the range that extends between AFrom and ATo (non-inclusive). RandomRange can handle negative ranges (where AFrom is greater than ATo). To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomRange.

Max(A,B)

Call Max to compare two numeric values. Max returns the greater value of the two.

Min(A,B)

Call Min to compare two numeric values in Delphi. Min returns the smaller value of the two.

Copyright © 2011 DB Software Laboratory

6.7.5 Miscellaneous Functions

Iif(expr1=expr2;expr3;expr4)

Iif function returns expr3 or expr4 depending on expr1=expr2

GetSystemVariable(VariableName)

GetSystemVariable returns value of 'VARIABLENAME'.

Possible values for 'VARIABLENAME' are:
- COMPUTERNAME
- OSUSERNAME
- DBUSERNAME
- BLOCKNUMBER
- LINENUMBER
- RECORDNUMBER
- SYSTEM_DATE
- SOURCE_FILE_NAME
- SOURCE_TABLE_NAME

GetSystemDate(Format)

Returns Current system date/time in format specified
GetSystemDate('MMDDYYYY')

Pos(Substr,S)

Pos searches for Substr within String and returns an integer value that is the index of the first character of Substr within String. Pos is case-sensitive. If Substr is not found, Pos returns zero.

AnsiPos(Substr,S)

Call AnsiPos to obtain the byte offset of the Substr parameter, as it appears in the string S. For example, if Substr is the string "AB", and S is the string "ABCDE", AnsiPos returns 1. If Substr does not appear in S, AnsiPos returns 0.

Chr(X)

Returns the character for a specified ASCII value.

Length(X)

Returns the number of characters in a string or elements in an array.

Copyright © 2011 DB Software Laboratory

Pos(Substr,Str)

Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero

SetLength(S,Length)

Set Length of dynamic array or string

H**igh(X)**

Call High to obtain the upper limit of an Array

Low(X)

Call Low to obtain the lowest value or first element of an Array.

6.7.6 Procedures

Abort

Use Abort to escape from an execution path without reporting an error.

Beep

Beep generates a conventional message beep.

Break

The Break procedure forces a jump out of the set of statements within a loop

Continue

The Continue procedure forces a jump past the remaining statements within a loop, back to the next loop iteration

Exit

The Exit procedure abruptly terminates the current function or procedure.

Copyright © 2011 DB Software Laboratory

6.7.7 Math functions

Sqr(X)

the Sqr function returns the square of the argument. X is a floating-point expression. The result, of the same type as X, is the square of X, or X*X.

Sqrt(X)

The result is the square root of X.

Exp(X)

Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms

Ln(X)

Ln returns the natural logarithm (Ln(e) = 1) of the real-type expression X.

Sin(X)

Sin returns the sine of the angle X in radians.

Cos(X)

Cos returns the cosine of the angle X. X expression that represents an angle in radians

Tan(X)

Tan returns the tangent of X. Tan(X) = Sin(X) / Cos(X).

ArcTan(X)

ArcTan returns the arctangent of X. X is a real-type expression that gives an angle in radians

PI

Represents the mathematical value pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

ArcCos(X)

ArcCos returns the inverse cosine of X. X must be between -1 and 1. The return value is in the range [0..Pi], in radians.

Copyright © 2011 DB Software Laboratory

ArcCosh(X)

ArcCosh returns the inverse hyperbolic cosine of X. The value of X must be greater than or equal to 1.

ArcCot(X)

ArcCot returns the inverse cotangent of X.

ArcCotH(X)

ArcCot returns the inverse hyperbolic cotangent of X.

ArcCsc(X)

ArcCsc returns the inverse cosecant of X.

ArcCscH(X)

ArcCsc returns the inverse hyperbolic cosecant of X.

ArcSec(X)

ArcSec returns the inverse secant of X.

ArcSecH(X)

ArcSec returns the inverse hyperbolic secant of X.

ArcSin(X)

ArcSin returns the inverse sine of X. X must be between -1 and 1. The return value will be in the range [-Pi/2..Pi/2], in radians.

ArcSinh(X)

ArcSinh returns the inverse hyperbolic sine of X.

ArcTan(X)

ArcTan returns the arctangent of X. X is a real-type expression that gives an angle in radians.

ArcTanh(X)

ArcTanh returns the inverse hyperbolic tangent of X. The value of X must be between -1 and 1 (inclusive).

Copyright © 2011 DB Software Laboratory

Cosecant(X)

Use the Cosecant to calculate the cosecant of X, where X is an angle in radians. The cosecant is calculated as 1/ Sin(X).

Cosh(X)

Use the Cosh to calculate the hyperbolic cosine of X.

Cot(X)

Call Cot to obtain the cotangent of X. The cotangent is calculated using the formula 1 / Tan (X).

Cotan(X)

Call Cotan to obtain the cotangent of X. The cotangent is calculated using the formula 1 / Tan (X)
Do not call Cotan with X = 0

CotH(X)

Call CotH to obtain the hyperbolic cotangent of X, where X is an angle in Radians.

Csc(X)

Use the Csc to calculate the cosecant of X, where X is an angle in radians.

CscH(X)

Use the CscH to calculate the hyperbolic cosecant of X, where X is an angle in radians.

CycleToDeg(X)

CycleToDeg converts angles measured in cycles into degrees, where degrees = cycles * 360.

CycleToGrad(X)

CycleToGrad converts angles measured in cycles into grads.

CycleToRad(X)

CycleToRad converts angles measured in cycles into radians, where radians = 2pi * cycles.

DegToCycle(X)

Use DegToCycle to convert angles expressed in degrees to the corresponding value in cycles.

Copyright © 2011 DB Software Laboratory

DegToGrad(X)

Use DegToGrad to convert angles expressed in degrees to the corresponding value in grads.

DegToRad(X)

Use DegToRad to convert angles expressed in degrees to the corresponding value in radians, where radians = degrees(pi/180).

GradToCycle(X)

GradToCycle converts angles measured in grads into cycles.

GradToDeg(X)

GradToDeg converts angles measured in grads into degrees.

GradToRad(X)

GradToRad converts angles measured in grads into radians, where radians = grads(pi/200).

Hypot(X,Y)

Hypot returns the length of the hypotenuse of a right triangle. Specify the lengths of the sides adjacent to the right angle in X and Y. Hypot uses the formula Sqrt(X**2 + Y**2)

IntPower(Base,Exponent)

IntPower raises Base to the power specified by Exponent.

Ldexp(X)

Ldexp returns X times (2 to the power of P).

LnXP1(X)

LnXP1 returns the natural logarithm of (X+1). Use LnXP1 when X is a value near 0.

Log10(X)

Log10 returns the log base 10 of X.

Log2(X)

Log2 returns the log base 2 of X.

LogN(Base,X)

LogN returns the log base Base of X.

Power(Base,Exponent)

Power raises Base to any power. For fractional exponents or exponents greater than MaxInt, Base must be greater than 0.

RadToCycle(X)

Use RadToCycle to convert angles measured in radians into cycles, where cycles = radians/(2pi).

RadToDeg(X)

Use RadToDeg to convert angles measured in radians to degrees, where degrees = radians(180/pi).

RadToGrad(X)

Use RadToGrad to convert angles measured in radians to grads, where grads = radians(200/pi).

RandG(Mean,StdDev)

RandG produces random numbers with Gaussian distribution about the Mean. This is useful for simulating data with sampling errors and expected deviations from the Mean.

Sec(X)

Call Sec to obtain the secant of X, where X is an angle in radians. The secant is calculated using the formula 1 / Cos(X).

SecH(X)

Call SecH to obtain the hyperbolic secant of X, where X is an angle in Radians.

Sinh(X)

Sinh calculates the hyperbolic sine of X.

Tan(X)

Tan returns the tangent of X. Tan(X) = Sin(X) / Cos(X).

Tanh(X)

Tanh calculates the hyperbolic tangent of X.

7. Date formats

Date/Time format strings control the conversion of strings into date time type.

Date/Time format strings are composed from specifiers which describe values to be converted into the date time value.

In the following table, specifiers are given in lower cases. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Description
d	Day as a number without a leading zero (1-31).
dd	Day as a number with a leading zero (01-31).
m	Month as a number without a leading zero (1-12).
mm	Month as a number with a leading zero (01-12).
mmm	Month as an abbreviation (Jan-Dec).
mmmm	Month as a full name (January-December).
yy	Year as a two-digit number (00-99).
yyyy	Year as a four-digit number (0000-9999).
h	Hour without a leading zero (0-23).
hh	Hour with a leading zero (00-23).
n	Minute without a leading zero (0-59).
nn	Minute with a leading zero (00-59).
s	Second without a leading zero (0-59).
ss	Second with a leading zero (00-59).
fff	Fraction of Second with a leading zero (000-999).
tt	Uses the 12-hour clock for the preceding ***h*** or ***hh*** specifier, 'am' for any hour before noon, and 'pm' for any hour after noon.

Important thing is to understand that this format has nothing to do with your target database. This is the format of the source data. It is there to help to covert string into date time type inside of the software, so it can be loaded later into date or timestamp field

So if source data is:

16/08/2009 than the format is DD/MM/YYYY
1/31/2009 than the format is M/D/YYYY
2006-05-23 22:34:42.096 than the format is YYYY-MM-DD HH:NN:SS.FFF
1992/mar/12 00:00 than the format is YYYY/MMM/DD HH:NN

Copyright © 2011 DB Software Laboratory

8. Command Line

To run an import from the command line type VImp.exe importscript.vis.

Copyright © 2011 DB Software Laboratory

9. Support Procedure

Solving problems today

We respect your time and hope you respect ours.

People ask us same questions again and again,

Please do not be selfish use our support forum, save your time and help the other users.
We are here to help you.

A lot of issues can be resolved in minutes provided that you supply all necessary information:

Version Number
Repository type
Database type you are working with.
Useful screenshots so we can see your entire screen not just error message
Steps to reproduce the problem eg click this click that => got error message
Description of the problem: Want to do this.. Do not know how...
Suggestions or ideas. (We do implement most of them)

If you do not provide this information, we will ask for it anyway and you will waste time.

We intend to resolve 90 percent of the reported problems within 48 hours.
Most difficult issues or functionality extension are resolved within one week

For General Sales and License Queries

Email to: sales@dbsoftlab.com

For Technical Help

Support Forum: http://www.etl-tools.com/etl-forum.html
Email To: support@etl-tools.com

Online Contact Form:

http://www.etl-tools.com/contact-db-software-laboratory.html

Copyright © 2011 DB Software Laboratory

10. License Agreement

Visual Importer ETL by
DB Software Laboratory
www.dbsoftlab.com
info@dbsoftlab.com

END-USER LICENSE AGREEMENT FOR THIS SOFTWARE
IMPORTANT - READ CAREFULLY:

This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and DB Software Laboratory for the SOFTWARE PRODUCT identified above, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation. By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, you may be subject to civil liability if you install and use this SOFTWARE PRODUCT.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties.

The SOFTWARE PRODUCT is licensed, not sold.

INSTALLATION AND USAGE

Once SOFTWARE PRODUCT is installed you may use it for 30 days. After evaluation period ends, you have to purchase a license or stop using the SOFTWARE PRODUCT.

If this is an EVALUATION VERSION of the SOFTWARE PRODUCT, you may copy and distribute an unlimited number of copies of the SOFTWARE PRODUCT; provided that each copy shall be a true and complete copy, including all copyright and trademark notices, and shall be accompanied by a copy of this EULA.

If this is a REGISTERED VERSION of the SOFTWARE PRODUCT, you may install and use it for your personal use only. You may not reproduce or distribute the SOFTWARE PRODUCT for use by others.

LICENSING

There are two types of licenses available

1. A single computer usage license. The user purchases one license to use the SOFTWARE PRODUCT on one computer.

2. A SITE usage license. The user purchases a single usage license, authorising the use of SOFTWARE PRODUCT, by the purchaser, the purchaser's employees or accredited agents, on an unlimited number of computers at the same physical site location. This site location would normally be defined as a single building, but could be considered to be a number of buildings within the same, general, geographical location, such as an industrial estate or small town.

OTHER RIGHTS AND LIMITATIONS

You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. Any such unauthorized use shall result in immediate and automatic termination of this license and may result in criminal and/or civil prosecution. All rights not expressly granted here are reserved by DB Software Laboratory.

The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

You may permanently transfer all of your rights under this EULA, provided the recipient agrees to the terms of this EULA.

SEVERABILITY

In the event of invalidity of any provision of this license, the parties agree that such invalidity shall not affect the validity of the remaining portions of this license.

COPYRIGHT

The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. All title and copyrights related to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by DB Software Laboratory.

MISCELLANEOUS

Should you have any questions concerning this EULA, or if you desire to contact the author of this Software for any reason, please contact DB Software Laboratory (see contact information at the top of this EULA).

LIMITED WARRANTY

DB Software Laboratory expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT and any related documentation is provided "as is" without warranty of any kind, either express or implied, including, without limitation, the implied warranties or merchantability, fitness for a particular purpose, or no infringement. The entire risk arising out of use or performance of the SOFTWARE PRODUCT remains with you.

In no event shall DB Software Laboratory be liable for any damages whatsoever or refund any money (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if DB Software Laboratory has been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

SUPPORT AND MAINTENACE

The annual maintenance fee is 20 percent of initial software license cost.

Users with a fully paid annual maintenance fee get the following benefits:

Priority Support
Free software enhancements, updates and upgrades during the maintenance period
Advanced and exclusive notification of software promotions
"Maintenance Owner ONLY" product promotions

ENTIRE AGREEMENT

This is the entire agreement between you and DB Software Laboratory which supersedes any prior agreement or understanding, whether written or oral, relating to the subject matter of this license.

Thank you for using the **Visual Importer ETL.**
DB Software Laboratory

Copyright © 2011 DB Software Laboratory