TARGIT®

TARGIT InMemory

Reference Guide

Contents

TARGIT InMemory

What is TARGIT InMemory Database?

TARGIT InMemory Database, offered for use with TARGIT Decision Suite, is similar to Microsoft's tabular engine. Both are column-store databases.

Column-store databases allow for large data set storage in a small space when loaded InMemory. Rather than repeating all the data values for each row like a traditional relational database would do, a column-store database only keeps a distinct list of values for each field and replaces the values of the transactions with highly indexed integer values so it can quickly look up the values and present the data set. This allows a 50GB+ database to fit into a few GB of memory space.

However, unlike Microsoft's tabular engine, the TARGIT InMemory Database can be used with standard edition of SQL Server and most other databases. Data can even be loaded directly from the source systems into the TARGIT InMemory Database using the TARGIT InMemory ETL Studio. This means there is no longer a reliance on SQL Server.

What is an InMemory Database?

An InMemory database primarily relies on main memory for data storage in contrast to more traditional database management systems that employ a disk storage mechanism. Once a database is loaded, data is "in memory" and ready-to-go, therefore it is able to retrieve and present information far more quickly than previously possible.

Following importation the TARGIT InMemory Database can be used as a data warehouse to be refreshed only when an update is required. You do not need to reload from a third-party source each time your pc is reset.

The benefits of an InMemory Database

In our experience most data sets loaded into analysis services cubes will benefit from InMemory loading for several reasons.

1. Query performance will be significantly improved as there will no longer be a reliance on disk IO to deliver the data. Instead, it will go straight to memory, which is significantly faster.
2. Combining data from across several dimensions is faster.
3. There is no need to cross analyze data files from the file system (to find non-empty data, like in cubes).
4. Data can be joined directly InMemory (similar to a relational database).

Thus, the TARGIT InMemory Database can be used to solve the performance bottlenecks common with traditional analysis services cubes.

Additionally, a set of BI functions have been added to the engine to replace calculations we are currently executing in MDX. For example, the balance calculation for Finance Balance and Inventory Balance can be made using the built-in balance function in the TARGIT InMemory Database. This is because it is a built-in function and data is sitting in memory, the performance is significantly better than the MDX calculated members done today.

Typical import speed

Import speed into TARGIT InMemory Database varies depending on connection type to the source database. Data can be imported via ODBC, OLEDB or Native Dot Net Providers.

In tests, we have found the following to be speeds typical of a Sql Server import:

- ODBC link 20,000 records per second
- OleDB 40,000 records per second
- Native Dot Net Provider 170,000 records per second

Database Configuration

The TARGIT InMemory Database Server (tiServer) is run as a Windows Service, installed by the installer.

By Default, TARGIT InMemory Database is suggested to be installed c:\Program Files\TARGIT in the folder TARGIT InMemoryDB for the binaries and c:\ProgramData\TARGIT\TARGIT InMemoryDB for the data and log files.

The TARGIT InMemory Database Server utilizes port tcp/5060. This port can manually be edited in the file targitdb.ini which also contains other system configurations.

If you change the targitdb.ini file, you will need to restart the service for the changes to take effect. TARGIT InMemory Database is comprised of a number of applications, namely:

- tiServer TARGIT InMemory Database Server.
- tiImport Import data from another source, in effect, to build a database.
- tiQuery Run SQL queries/view data structure on the TARGIT InMemory Database Server.
- tiLoad Load/Reload an TARGITDB file into memory, ready for data-access.

Database Server (tiServer.exe)

TARGIT InMemory Database Server is run automatically on installation and subsequently on system reboot. It is possible to start and stop TARGIT InMemory Database Server with the following command-line commands:

net stop “tiServer”
net start “tiServer”

TARGITDB.INI

This is the main configuration file for the TARGIT InMemory Database Server service. It is placed in the install directory for the tiServer executable. All configuration parameters currently go into the general section. The main required ones are set by the installer:

- **data_dir**
  Directory of the data files.
- **Port**
  Port to run the TARGIT InMemory Database on. Default value is 5060.

- **Password**
  Password to validate against when clients connect.

Optional parameters are:

- **Nocache**
  Setting this to true, will cause subselect inner queries not to be cached. This causes nocache to become the default behaviour.
- **Maxcpus**
  Specify an integer here to limit the maximum number of cores that that TARGIT InMemory Database uses to process a query. When running on computers with large number of cores > 16, setting a value of 50% of the number of cores, can help performance.
- **unload_timer**
  Specify an integer here that causes inactive databases to be unloaded after that amount of time in minutes. This helps reduce memory consumption, when you have large numbers of databases, of which only a % are being used concurrently. Default is 0, which causes inactive databases not to be unloaded.
- **unload_temptables**
  Specify an integer here that causes inactive temp to be released automatically after that amount of time in minutes. The system default is 30 minutes.
- **log_dir**
  Lets you specify a directory to place the log file for the TARGIT InMemory Database.
- **Logall**
  For systems with a high amount of queries it is beneficial to turn off logging all queries. This can be done by setting the value in the TARGITDB.INI file to logall=false
- **Querycache**
  Query cache controls whether the database itself should provide a layer of caching for TARGIT. By default the query cache holds the cached data in memory for 10 minutes. This can be controlled by the querycache_timeout parameter. Please note that resultsets with with more than 10 million rows or where rows * columns > 100 million will also not be cached.
- **querycache_timeout**
  This parameter controls the amount of time resultsets are kept in cache. The value specified is in minutes.
- **LazyLoad**
  Setting this to **true** will enable the LazyLoad mode. This loads columns on demand, and reduces memory consumption.
- **lazyload_unload_column_timer**
  Setting this to a number e.g. 15 will cause database columns to be unloaded after 15 minutes of inactivity, when lazyload is set.
- **Windowsauth**
  Setting this to **true,** makes the server run in Windows Authentication mode. **windowsauth_role** When windowsauth is set, this specifies a role that is allowed to connect. Default value is Administrators.
- **Encrypt**
  Setting this to **true**, will make the server run in encrypted mode.
- **encrypt_cert**
  Specifies the file location of the server certificate for encryption. (PFX File)
- **encrypt_password**
  Specifies the encryption password for the cert specified by encrypt_cert

- **Skiptables**
  Specifies a comma separate list of tables / or table patterns not to load. Patterns use sql like syntax. E.g. skiptables=stage% would cause any table starting with stage not to load. Designed to help reduce memory consumption by specifying certain tables not to load.
- **query_log_size**
  Setting this to a non zero number e.g. 10000 will cause the server to remember the last 10,000 queries for each database. This information can then be retrieved with select * from information_schema.query_log

Import Tool

Importing into TARGIT InMemory Database

tiImport is a command line utility used to import data into TARGIT InMemory Database (build a database).

The tiImport.exe takes one main parameter which is the name of the import file to read in

e.g. From the command line, type
tiImport northwind.imp

This executes an import file called northwind.imp and stores the data as northwind.targitdb in the same folder.
Additional parameters can be referenced as variables inside the script.

Note 1: *If you would like to run the example imports (supplied with TARGIT InMemory Database) based on the Northwind database, ensure you have a Northwind database available.*
–e.g. MS Access file "northwind.mdb" can be downloaded from multiple sources.
Next create a data connection to this source.
–e.g. 32bit or 64bit ODBC driver, called "targit_northwind" (as used in northwind.imp), pointing to this database.

Note 2: *You can edit northwind.imp with a text editor to see what it is doing.*

In this case, assuming TARGIT InMemory Database was installed to c:\program files\TARGIT\TARGIT InMemoryDB you can run this import file by opening a command prompt, going to the C:\ProgramData\TARGIT\TARGIT InMemoryDB\ data folder, then entering:

"c:\program files\targit\TARGIT InMemoryDB\tiImport" "c:\programdata\targit\TARGIT InMemoryDB\data\northwind.imp"

This calls on tiImport.exe to run the .imp file.

Note 3: .imp is the standard extension for TARGIT InMemory import scripts. It generates a .targitdb data file of the same name in the same directory.

Note 4: tiImport.exe runs in 64bit mode in a 64bit environment and in 32bit mode in a 32bit environment.

To connect to a 32bit data-source (e.g. 32bit ODBC) whilst running in a 64bit environment, tiImport32.exe should be used instead.

tiImport32 does exactly the same as tiImport but will always run in 32 bit mode. This can be necessary if e.g. connection using a Dynamics NAV Native driver that only is deliver as 32bit.

To reload an update data file on the server, use tiLoad with the /reload parameter.

tiImport Reference Guide

The import file (.imp) contains a list of commands that imports tables into a database. Click the links to the left to learn more about them.

ALTER TABLE ADD COLUMN

Modifies the table structure by adding a new column.

Syntax:
ALTER TABLE table_name ADD COLUMN new_column_name = { sql_expression}

Example:

```
ALTER TABLE customer ADD COLUMN customerType = {CASE WHEN "Source Type"=1 THEN 'Retail' ELSE
'Wholesale' END}
```

```
ALTER TABLE tStageCustLedgerEntry ADD COLUMN wAgingBucket = {IIF([Open]=0 AND
[wRemainingAmountLCY]<>0,
        CASE WHEN [wAgingDaysOverdue] <= 0  THEN 0
                WHEN [wAgingDaysOverdue] > 0  AND [wAgingDaysOverdue] <= 30 THEN 30
                WHEN [wAgingDaysOverdue] > 30 AND [wAgingDaysOverdue] <= 60 THEN 60
                WHEN [wAgingDaysOverdue] > 61 AND [wAgingDaysOverdue] <= 90 THEN 90
                WHEN [wAgingDaysOverdue] > 90 THEN 120
        ELSE    0               END, 0)}
```

CASE

Evaluates the expression and returns the specified value if a condition is met.

The CASE expression can be used in two ways: simple or list:

Simple expression

The simple expression evaluates the expression and returns the value specified, optionally a default value can be specified

CASE WHEN expression operator expression THEN return_expression [ELSE return_expression] END

List Expression

The list expression evaluates the value of an expression and measures it against another expression in the form seen below:

```
CASE expression
   WHEN eval_expression_1 THEN return_expression
   WHEN eval_expression_2 THEN return_expression
ELSE
```

```
   return_expression
END
```

Example:

Simple Usage

```
DECLARE @myvar as int

set @myvar=100

PRINT 'Size of myvar:'+CASE
      WHEN @myvar<100 THEN 'Small'
      WHEN @myvar>=100  AND @myvar<500 THEN 'Medium'
      WHEN @myvar>=500  AND @myvar<1500 THEN 'Large'
      ELSE 'Enterprise' END
```

List Usage

```
DECLARE @myvar as int
set @myvar=1

PRINT 'MyVar :'+ CASE @myvar

      WHEN 1 THEN 'Step 1'
      WHEN 2 THEN 'Step 2'
      WHEN 3 THEN 'Step 3'
      ELSE  'Step '+@myvar
END
```

CLUSTERED INDEX

For even more speed, it is possible to create a clustered index on one field in a table. The table will be sorted by that index, the index will be used on Range Queries or queries with multiple qualities.

Syntax:

```
CREATE CLUSTERED INDEX ON table_name ( field_name )
```

Example:

```
CREATE CLUSTERED INDEX ON [combined_sales] ( OrderDate )
```

CREATE TABLE

The CREATE TABLE command create a high-performance un-columnized table for inserting scripted data using the INSERT INTO command.

For the table to be available in the InMemory data store the command COLUMNIZE must be used after completing insertions.

Example:

```
/* Example of creating a table with the name periodtable that will contain all dates from 2015-2016 */

DECLARE @mindate as datetime
DECLARE @maxdate as datetime
SET @mindate='2015-01-01'
SET @maxdate='2016-01-01'

/* Create a new un-columnized table structure for mass insertions - much quicker than running multiple
IMPORTs */
CREATE TABLE periodtable ([Posting Date] datetime, [year] int, [quarter] int, [month] int, [DateName] string)
WHILE @mindate<@maxdate

/* inserts like this only works for un-columnized tables. */
/*Please note that not all sql functions are supported, in the example below calculations */
/* are done for quarter and date instead of using the appropriate functions as of now this is by design. */
INSERT INTO periodtable ([Posting Date], [Year], [quarter], [month], [datename])
VALUES (@mindate,year(@mindate),(1+month(@mindate)-1)/3, month(@mindate),
right('00'+month(@mindate),2)+'/'+right('00'+day(@mindate),2)+'/'+year(@mindate))
SET @mindate=dateadd('d',1,@mindate)
LOOP

/* columnize it so it gets to the data warehouse and we can perform normal operations on the table */
COLUMNIZE periodtable
```

CREATE VIEW / DROP VIEW

Use CREATE VIEW for creating views and DROP VIEW for dropping them.

By default, views act as virtual tables that are persisted at time of first use. Addding NOCACHE after the first SELECT will cause the view to be evaluated and run on demand and not persisted. Otherwise, it will follow the default caching behavior.

Example:

```
create view MyView as select * from Orders union all select * from orders

create view MyView as select NOCACHE * from Orders union all select * from orders
```

COLUMNIZE

After creating a table with the CREATE TABLE statement and using INSERT INTO to add data to the table the command COLUMNIZE must be performed in order to make the table usable with standard InMemory commands.

Syntax:
COLUMNIZE *tablename*

DECOLUMNIZE Command

If you want to decolumnize one of your tables, you can use the DECOLUMNIZE command. This will take a table, that has been columnized into the a compressed format, and expand it back out, so you can insert data into it.

Syntax:

DECOLUMNIZE *tablename*

SET PRESERVE DECIMAL

By default, decimal data is imported into Double format to speed up calculations. Double types can execute up to 5 times faster than Decimal. This may cause rounding issues when dealing with large amount of records.

To disable this behavior, use the command

SET PRESERVE DECIMAL TRUE

To re-enable Decimal to Double conversion, use

SET PRESERVE DECIMAL FALSE

DATASOURCE

The very first thing that should be done at the top of the import file (.imp) is to declare the Data sources you want to import-from by using the DATASOURCE command.

Note: *You can declare as many datasources as you need in order to create your TARGITDB file.*
Note: *DOTNETDataSouce and MEDDataSources are covered seperately*

Sources can be ODBC, OLEDB, DOTNET or several other sources listed below.

The following can be used in a single .imp file importation.

Syntax:
DATASOURCE Source1 = ODBC 'odbc_connection_X'
DATASOURCE Source2 = ODBC 'odbc_connection_Y'
DATASOURCE Source3 = SQLSERVER 'connection_string'
DATASOURCE Source4 = DOTNET PROVIDER 'System.Data.SqlClient' 'Data Source=localhost;Initial Catalog=yourDB;User Id=your_username; Password=your_password;'

You can open other locally stored .TARGITDB Data files and import tables or execute sql against them like any other Data Source

Syntax:
DATASOURCE yourname=LOCAL INMEMORYDB 'LOCAL_FILE_Name'

Example:

/* Connection_string is an ODBC Connection string: */
/* dsn=ir_northwind for a system odbc datasource called ir_northind. */
/* yourname is the name that you use as an alias for the datasource in IMPORT commands */
/* Refer to www.connectionstrings.com for a full reference on odbc connection strings */
DATASOURCE yourname=ODBC 'connection_string'

/* Connection_string is an OLE DB Connection string */
/* Refer to www.connectionstrings.com for a full reference on oledb connection strings */
DATASOURCE yourname=OLEDB 'connection_string'

/* Connection_string in this case is the connection string for the Sql Server Dot Net Provider for Standard Authentication */
DATASOURCE yourname=SQLSERVER 'connection_string'

/* **Note**: The Dot Net Provider is the fastest way of connecting to sql server */
.DATASOURCE mySQLSource=SQLSERVER 'Data Source=localhost; Initial Catalog=myDBName;User Id=myUser; Password=myPwd;'

/* This opens the local data contoso.targitdb as datasource A1 */
DATASOURCE A1=LOCAL INMEMORYDB 'contoso.targitdb'

DOT NET Datasource

A Dot Net Data provider source can be set with the keywords DOTNET PROVIDER after the equals symbol. It takes two parameters.

Param 1 is the dot net name of the class
Param 2 is the connection string for the provider

Example:

DATASOURCE a1=DOTNET PROVIDER 'System.Data.SqlClient' 'Data Source=localhost; Initial Catalog=yourDB; User Id=your_username; Password=your_password;'

The Dot Net provider Class needs to be accessible from tiImport/tiImport32

Example:

LOAD ASSEMBLY 'Npgsql.dll'
DATASOURCE a1 = DOTNET CONNECTION 'Npgsql.NpgsqlConnection'

User ID=your_user_id;Password=your_password;Host=your_host;Port=5432;Database=your_database'

This example loads the Postgres Dot Net driver with the LOAD ASSEMBLY Command, and creates a connection to a Postgres server.

ME Datasource

ME (case unimportant) is a special “built-in” data source that refers to the current InMemory database you are building. ME is not declared at the top of the import file since it always refers to “this current import process”. ME is used with the IMPORT statement when you want to refer to a table or execute a SQL command against a table that you have already imported in this same .imp file import.

```
DATASOURCE Source1 = ODBC 'odbc_connection_X'

/* Comments can be entered in an impfile like this */
DATASOURCE Source2 = ODBC 'odbc_connection_Y'

/* Comments can also begin and end with these characters which can be useful if they span more than one line */
DATASOURCE Source3 = SQLSERVER 'Provider=SQLNCLI11;
Server=192.168.1.45;Database=ir_dw;Uid=usr1; Pwd=pwd;'
DATASOURCE yesterdays_targitdb = LOCAL INMEMORYDB 'contoso.targitdb'
DATASOURCE A1 = ODBC ‘odbc_connection_Z’
```

Declare

You can declare variables for use inside the tiImport script, which can be used in expressions and other commands. Variables begin with @ and then have an alphabetic character, followed by an alphanumeric character. There are 5 supported types:

- STRING
- DATETIME
- LONG
- INT
- DOUBLE

Note: a numeric character cannot be used immediately after the @ symbol, but can be used in any position subsequent to that.

Syntax:

```
DECLARE @variablename as datatype
```

Example:

```
DECLARE @cvar1 as STRING
DECLARE @nvar1 as INT
```

Any command line parameter is available as a special variable called @param1 ,… @paramN of type STRING.

```
/* For this importation, within the mysample2.imp file, @param1 = "companyX" */
c:\Program Files\TARGIT\TARGIT InMemoryDB\tiImport.exe -> c:\Program Files\TARGIT\TARGIT
InMemoryDB\tiImport.exe
```

DROP

The DROP command is used to drop tables no longer required. This is useful for removing intermediate tables, used to create summary tables in order to reduce the size of the TARGITDB file.

Syntax:

```
DROP table_name
```

Note: The table in question will not be saved in the TARGITDB file.

Example:

```
DROP [order details]
```

EXECUTE

The EXECUTE Command is used to execute a SQL statement against the current TARGITDB, much the same as with 'ME', storing the results back into the TARGITDB.

Syntax:

```
EXECUTE table_to_store_results_as = { some_SQL_command in curly bracks }
```

Example:

```
EXECUTE Combined_Sales = {SELECT 2.0 as currency_factor,
                [order details].UnitPrice AS unit_price,
                [Order Details].Quantity, [Order Details].costprice,
                Orders.CustomerID, Orders.EmployeeID,Orders.ShipVia,
                [Order Details].ProductID,
                Products.CategoryID, Products.SupplierID,
                Orders.OrderID, Orders.OrderDate
FROM [Order Details]
                INNER JOIN Orders ON [Order Details].OrderID = Orders.OrderID
                INNER JOIN Products ON [Order Details].ProductID = Products.ProductID
                }
```

SCRIPTEXEC

It is possible to dynamically generate complete programs and execute them using the scriptexec statement. The following example imports the Orders table from the Northwind odbc data source. The generated code, should be in the tiImport language syntax.

Example:

```
DATASOURCE a1 = odbc ‘dsn=northwind’

DECLARE @tableName as string

SET @tableName = ‘Orders’

ScriptExec 'import [' + @tableName + '] = a1.[ ' + @tableName + '] '

SAVE
```

SQLEXEC

An external sql script file can be run using the SQLEXEC command. The command takes the file name as the parameter.

Syntax:

```
SQLEXEC 'myquery.sql'
```

EXIT

You can have tiImport stop execution and exit with a specific code to the OS.

Syntax:

```
EXIT {exit_code}
```

Note:This exits tiImport and sends an optional code back to the Operating System.

Example:

```
EXIT -1
```

FILEEXISTS

The FILEEXISTS command checks whether a file exists in the filesystem where the tiImport application is executed and returns TRUE or FALSE.

Syntax:

```
FILEEXISTS(path_and_filename_to_check)
```

Example:

```
/* Checks if a file exists and only if found prints the result */
IF FILEEXISTS('c:\budget\mybudget.xlsx')=TRUE THEN
   PRINT 'The file MYBUDGET.XLSX exists'
END IF
```

FLUSH

The FLUSH command writes all tables loaded until this checkpoint to disk and makes the in-operable for joins, updates, delete and other statements. Tables flushed to disk will be available in the final InMemory datastore and can be used in TARGIT.

Syntax:

FLUSH

Example:

```
/* In this example we are using the Oracle Managed Data provider */
/* and retrieving data from an Oracle database */
LOAD ASSEMBLY 'Oracle.ManagedDataAccess.dll'
DATASOURCE oracledb=DOTNET CONNECTION 'Oracle.ManagedDataAccess.Client.OracleConnection' 'data
source=(DESCRIPTION =(ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST =myOracleServer)(PORT =
1521)))(CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = )));User Id = oracleLogin; Password =
readOraclePassword'

/* Read the SalesInvoiceLines table with 400 mill rows */
IMPORT SalesInvoiceLines=oracledb.{SELECT * FROM SALESINVOICELINES}
/* Write the table to disk and release memory */
FLUSH

/* Import other tables where we don't need to worry about memory */
DO PARALLEL
        IMPORT LOCATIONS=oracledb.{SELECT * FROM LOCATIONS}
        IMPORT DEPARTMENTS=oracledb.{SELECT * FROM DEPARTMENTS}
        IMPORT PRODUCTS=oracledb.{SELECT * FROM PRODUCTS}
        IMPORT EMPLOYEES=oracledb.{SELECT * FROM EMPLOYEES}
END PARALLEL

/* Commit data to disk */
SAVE
```

FOR Loop

The FOR loop iterates over a defined variable by incrementing the value with the specified increment.

FOR *@variable=start_value* TO *end_value+increment_value*

  PRINT 'The value of @variable is now:'+*@variable*

NEXT

```
DECLARE @i as int


FOR @i=1 TO 10+1
        PRINT 'Value of variable '' + @i
NEXT
```

FORCEGC

To eliminate overhead memory usage a garbage collection can explicitly be configured to be run after this amount of rows. This is only necessary in case of limited amount of memory on the server.

Syntax:

FORCEGC *count_of_rows*

Example:

```
/* In this example we are using the Oracle Managed Data provider and retrieving data from an Oracle
database */
LOAD ASSEMBLY 'Oracle.ManagedDataAccess.dll'
DATASOURCE oracledb=DOTNET CONNECTION 'Oracle.ManagedDataAccess.Client.OracleConnection' 'data
source=(DESCRIPTION =(ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST =myOracleServer)(PORT =
1521)))(CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = )));User Id = oracleLogin; Password =
readOraclePassword'

/* Force garbage collection to run every 10 million rows in this script – thereby freeing up memory during
the import process*/
FORCEGC 10000000
/* Read the SalesInvoiceLines table with 400 mill rows */
IMPORT SalesInvoiceLines=oracledb.{SELECT * FROM SALESINVOICELINES}
/* Write the table to disk and release memory */
FLUSH
/* Import other tables where we don't need to worry about memory */
DO PARALLEL
        IMPORT LOCATIONS=oracledb.{SELECT * FROM LOCATIONS}
        IMPORT DEPARTMENTS=oracledb.{SELECT * FROM DEPARTMENTS}
        IMPORT PRODUCTS=oracledb.{SELECT * FROM PRODUCTS}
        IMPORT EMPLOYEES=oracledb.{SELECT * FROM EMPLOYEES}
END PARALLEL
/* Commit data to disk */
SAVE
```

If

You can use the IF Command to do control flow in your tiImport script.
AND, OR, NOT and parenthesis are supported in the logic.
Currently only the Equals operator is supported.
ELSE is optional in the IF statement

Syntax:

```
IF expression1 = expression2 THEN
statementList
{ELSE
statementList}
```

END IF

Example:

```
IF @var1=10 THEN
PRINT 'HELLO WORLD'
END IF
```

IMPORT

Once a data-source is declared, data can be imported via the IMPORT command.

table_name_in_targitdb – defines the table the import will store the data in datasource_name is the alias for one datasource table_name is the name of the table to read in the source database.

Syntax:

IMPORT table_name_in_targitdb=datasource_name.table_name

Example:

```
/*This imports the categories table from datasource a1 and stores it as the categories table in designated
.TARGITDB file*/
IMPORT categories=a1.categories
/*You can use Square Brackets [] around the destination table name to save it as is. You can also use CURLY
{} brackets, after the . to read from an arbitrary SQL statement*/
IMPORT [Order Details]=a1.{select * from [Order Details]}
/*You can also import data using the sql contained in another file by putting quotes around the filename. It
is assumed the .sql file is in the same folder.*/
IMPORT sometable=a1.'Name_Of_File_to_read.sql'
```

IMPORT UNION ALL

Multiple UNION ALL operators can also be combined. The number of columns and field datatypes need to match across each table/ query in the Union.

By default the TARGIT InMemory Database insists on strict datatype matches between datasets in a Union All.

For instance, numeric fieldA (which is, say, a Double) in Table1 linking with fieldA (which is datatype Float) in Table2 will cause an error as the InMemory Server expects them to have the same datatype. This assumption improves performance, but may be switched off by use of the USETEMP keyword.

Note: SQL syntax is dependent on the source database syntax rules. For a list of specific SQL Syntax see TARGIT InMemory Database SQL Reference.

Syntax:

IMPORT combined_table=USETEMP A1.table1 UNION ALL a2.table2

Example:

```
/*This creates a temporary table behind the scenes in TARGIT InMemory Database for both tables before
creating the final one. Thus more time and overheads are used.*/
IMPORT decimal_test= source2.decimal_test
IMPORT Products= source2.products
IMPORT large_query = source1.
                {SELECT Orders.OrderID AS int_col,
                Orders.OrderDate AS date_col,
                [Order Details].Quantity AS short_col,
                [Order Details].UnitPrice AS dec_col,
                Orders.ShipName AS text_col,
                Products.Discontinued AS bit_col,
                CDbl([Order Details].[UnitPrice]) AS Dbl_Col,
                CSng([Order Details].[UnitPrice]) AS float_col,
                Categories.Picture AS bitarray_col
FROM Categories
INNER JOIN (Products INNER JOIN (Orders INNER JOIN
                [Order Details] ON Orders.OrderID = [Order
                Details].OrderID) ON Products.ProductID = [Order
                Details].ProductID) ON Categories.CategoryID =
                Products.CategoryID; }
                import datatype_table= me.{
                select *, CAST_NUM_AS_LONG(int_col) as int64_col,
                CAST_NUM_AS_BYTE(int_col%64) as byte_col
                from large_query }

/* Perhaps there are Customers in two data sources (duplicate tables/ different set of customers in each)
and you need to bring them together as one in TARGITDB... */
IMPORT Combined_Custs =USETEMP Source1.customers
UNION ALL
Source3.customers

/* Alternatively, you could bring in both, and then combine...*/
IMPORT Custs1 =Source1.customers
IMPORT Custs2 =Source3.{SELECT * FROM customers}
IMPORT Combined_Custs = ME.Custs1 UNION ALL ME.Custs2
```

IMPORTTIMEOUT

Import timeout affects the amount of time that tiImport will wait for a data source to return data.
If you have a very large SQL query running on e.g. a slow Oracle database you can set the amount of time in seconds.

Example:

```
/* Sets the import timeout to 1800 seconds */
IMPORTTIMEOUT 1800
```

INSERT INTO

Data rows can be inserted into an un-columnized table create with the CREATE TABLE-command.
For supported data types see _datetypes link.

Note: INSERT INTO does NOT work with a table that's created with the IMPORT statement as that table already is columnized.

Syntax
INSERT INTO (*column1, column2, column3, …)* VALUES (*value1,value2,value3, …*)

Example:

CREATE TABLE customer (customerno int, customername string)

INSERT INTO customer (customerno,customername) VALUES (1,'Smith')
INSERT INTO customer (customerno,customername) VALUES (2,'Jones')

COLUMNIZE customer

MOVENEXT

The keyword MOVENEXT is used in conjunction with a WHILE statement iteration over a table. For a more detailed example see WHILE.

```
WHILE table.EOF=false
        PRINT 'We just read the column and got the value '+table.Column_name_in_table
        MOVENEXT table
LOOP
```

PRINT

You can print the output of an expression to the console using this command. This is useful for debugging purposes.

Syntax:

PRINT expression

Example:

PRINT 'Hello World'
PRINT @var2

SAVE

The SAVE command enables committing the loaded TARGIT InMemory Database to the disk and also makes it possible to store a single InMemory table in the file system as either a compressed .targitdb file or as a CSV file.

Note: When building a tiImport script the last command must always SAVE command without any parameters, otherwise the data processed until that point will be discarded.

Syntax:

SAVE [TABLE *tablename* TO {CSV} *'path_in_filesystem'*]

Example:

```
/* Connect to SQL Server */
DATASOURCE sqldb=SQLSERVER 'Server=localhost;Database=StageTCP;Trusted_Connection=True;'


/* Read all tables from the SQL database – SLURP only works on MS SQL, Access and TARGIT InMemory
Database connections */
SLURP sqldb

/* Save a single table to a storage location in TARGITDB format, a valid path must be specified */

SAVE TABLE 'salestransactions' TO  'C:\OUTPUT\salestransactions.targitdb'

/* Save a single table as a CSV to a storage location, a valid path must be specified */

SAVE TABLE 'salestransactions' TO CSV 'C:\OUTPUT\salestransaction.csv'

/* Commit entire database to disk */
SAVE
```

SET

You can use the SET command to set the value of a variable equal to an expression.

Syntax:

SET @variablename = expression

Like:

```
SET @var1 = 5+5
SET @var1 = 5+(20*3)
SET @var2 = @var1+50
```

An important feature of variables is that they can be used inside the SQL of IMPORT statements. When the import statement uses an TARGITDB type datasource, a single Declare and Set can be made at the beginning of the .imp file, otherwise a variable to be used in a query from an external source must be declared and set within the IMPORT brackets.

Example:

```
DATASOURCE Source1 = ODBC 'odbc_connection_X'
IMPORT orders = source1.orders
/* Say we want to create another order table with a specific subset of items only... */
IMPORT orders_some_only = ME.
{
DECLARE @cSupp1 AS STRING
SET @cSupp1 = “Smiths Sweets Inc.”
SELECT * FROM ORDERS AS o1 WHERE o1.Supplier = @cSupp AND o1.Office = @param1
/* Notice we did not have to declare @param1 since it is a system function, in this case created when we
```

called the ‘mysample2.imp’ import script */
}

From the Supported Cultures page, you can find the Culture Value you wish to use.
Syntax:

SET CULTURE '*value*'

Example:

```
SET CULTURE 'en-US'
IMPORT datetable= me.{select SAFECAST ('09/01/2010' as DATETIME ) as thedate}
```

Supported Cultures

The CultureInfo class specifies a unique name for each culture, based on RFC 4646 (Windows Vista and later). The name is a combination of an ISO 639 two-letter lowercase culture code associated with a language and an ISO 3166 two-letter uppercase subculture code associated with a country or region.

Country	Two Letter Country Code	Three Letter Country Code	Language	Two Letter Lang Code	Three Letter Lang Code	CultureInfo Code
Afghanistan	AF	AFG	Pashto	ps	pus	ps-AF
Afghanistan	AF	AFG	Dari	prs	prs	prs-AF
Albania	AL	ALB	Albanian	sq	sqi	sq-AL
Algeria	DZ	DZA	Arabic	ar	ara	ar-DZ
Argentina	AR	ARG	Spanish	es	spa	es-AR
Armenia	AM	ARM	Armenian	hy	hye	hy-AM
Australia	AU	AUS	English	en	eng	en-AU
Austria	AT	AUT	German	de	deu	de-AT
Bahrain	BH	BHR	Arabic	ar	ara	ar-BH
Bangladesh	BD	BGD	Bengali	bn	bng	bn-BD
Basque	ES	ESP	Basque	eu	eus	eu-ES
Belarus	BY	BLR	Belarusian	be	bel	be-BY
Belgium	BE	BEL	French	fr	fra	fr-BE
Belgium	BE	BEL	Dutch	nl	nld	nl-BE
Belize	BZ	BLZ	English	en	eng	en-BZ

Bolivarian Republic of Venezuela	VE	VEN	Spanish	es	spa	es-VE
Bolivia	BO	BOL	Quechua	quz	qub	quz-BO
Bolivia	BO	BOL	Spanish	es	spa	es-BO
Brazil	BR	BRA	Portuguese	pt	por	pt-BR
Brunei Darussalam	BN	BRN	Malay	ms	msa	ms-BN
Bulgaria	BG	BGR	Bulgarian	bg	bul	bg-BG
Cambodia	KH	KHM	Khmer	km	khm	km-KH
Canada	CA	CAN	French	fr	fra	fr-CA
Canada	CA	CAN	English	en	eng	en-CA
Caribbean	29	29	English	en	eng	en-029
Catalan	ES	ESP	Catalan	ca	cat	ca-ES
Chile	CL	CHL	Mapudungun	arn	arn	arn-CL
Chile	CL	CHL	Spanish	es	spa	es-CL
Colombia	CO	COL	Spanish	es	spa	es-CO
Costa Rica	CR	CRI	Spanish	es	spa	es-CR
Croatia	HR	HRV	Croatian	hr	hrv	hr-HR
Cyrillic, Azerbaijan	AZ	AZE	Azeri	az	aze	az-Cyrl-AZ
Cyrillic, Bosnia and Herzegovina	BA	BIH	Serbian	sr	srn	sr-Cyrl-BA
Cyrillic, Bosnia and Herzegovina	BA	BIH	Bosnian	bs	bsc	bs-Cyrl-BA
Cyrillic, Mongolia	MN	MNG	Mongolian	mn	mon	mn-MN
Cyrillic, Montenegro	ME	MNE	Serbian	sr	srp	sr-Cyrl-ME
Cyrillic, Serbia	RS	SRB	Serbian	sr	srp	sr-Cyrl-RS
Cyrillic, Serbia and Montenegro	CS	SCG	Serbian	sr	srp	sr-Cyrl-CS
Cyrillic, Tajikistan	TJ	TAJ	Tajik	tg	tgk	tg-Cyrl-TJ
Cyrillic, Uzbekistan	UZ	UZB	Uzbek	uz	uzb	uz-Cyrl-UZ
Czech Republic	CZ	CZE	Czech	cs	ces	cs-CZ
Denmark	DK	DNK	Danish	da	dan	da-DK
Dominican Republic	DO	DOM	Spanish	es	spa	es-DO

Ecuador	EC	ECU	Quechua	quz	que	quz-EC
Ecuador	EC	ECU	Spanish	es	spa	es-EC
Egypt	EG	EGY	Arabic	ar	ara	ar-EG
El Salvador	SV	SLV	Spanish	es	spa	es-SV
Estonia	EE	EST	Estonian	et	est	et-EE
Ethiopia	ET	ETH	Amharic	am	amh	am-ET
Faroe Islands	FO	FRO	Faroese	fo	fao	fo-FO
Finland	FI	FIN	Finnish	fi	fin	fi-FI
Finland	FI	FIN	Swedish	sv	swe	sv-FI
Finland	FI	FIN	Sami, Northern	se	smg	se-FI
Finland	FI	FIN	Sami, Skolt	sms	sms	sms-FI
Finland	FI	FIN	Sami, Inari	smn	smn	smn-FI
Fmr. Yugoslav Rep.of Macedonia	MK	MKD	Macedonian	mk	mkd	mk-MK
France	FR	FRA	French	fr	fra	fr-FR
France	FR	FRA	Breton	br	bre	br-FR
France	FR	FRA	Occitan	oc	oci	oc-FR
France	FR	FRA	Corsican	co	cos	co-FR
France	FR	FRA	Alsatian	gsw	gsw	gsw-FR
Galician	ES	ESP	Galician	gl	glg	gl-ES
Georgia	GE	GEO	Georgian	ka	kat	ka-GE
Germany	DE	DEU	German	de	deu	de-DE
Germany	DE	DEU	Upper Sorbian	hsb	hsb	hsb-DE
Germany	DE	DEU	Lower Sorbian	dsb	dsb	dsb-DE
Greece	GR	GRC	Greek	el	ell	el-GR
Greenland	GL	GRL	Greenlandic	kl	kal	kl-GL
Guatemala	GT	GTM	K'iche	qut	qut	qut-GT
Guatemala	GT	GTM	Spanish	es	spa	es-GT
Honduras	HN	HND	Spanish	es	spa	es-HN
Hungary	HU	HUN	Hungarian	hu	hun	hu-HU
Iceland	IS	ISL	Icelandic	is	isl	is-IS

India	IN	IND	Hindi	hi	hin	hi-IN
India	IN	IND	Bengali	bn	bng	bn-IN
India	IN	IND	Punjabi	pa	pan	pa-IN
India	IN	IND	Gujarati	gu	guj	gu-IN
India	IN	IND	Oriya	or	ori	or-IN
India	IN	IND	Tamil	ta	tam	ta-IN
India	IN	IND	Telugu	te	tel	te-IN
India	IN	IND	Kannada	kn	kan	kn-IN
India	IN	IND	Malayalam	ml	mym	ml-IN
India	IN	IND	Assamese	as	asm	as-IN
India	IN	IND	Marathi	mr	mar	mr-IN
India	IN	IND	Sanskrit	sa	san	sa-IN
India	IN	IND	Konkani	kok	kok	kok-IN
India	IN	IND	English	en	eng	en-IN
Indonesia	ID	IDN	Indonesian	id	ind	id-ID
Iran	IR	IRN	Persian	fa	fas	fa-IR
Iraq	IQ	IRQ	Arabic	ar	ara	ar-IQ
Ireland	IE	IRL	Irish	ga	gle	ga-IE
Ireland	IE	IRL	English	en	eng	en-IE
Islamic Republic of Pakistan	PK	PAK	Urdu	ur	urd	ur-PK
Israel	IL	ISR	Hebrew	he	heb	he-IL
Italy	IT	ITA	Italian	it	ita	it-IT
Jamaica	JM	JAM	English	en	eng	en-JM
Japan	JP	JPN	Japanese	ja	jpn	ja-JP
Jordan	JO	JOR	Arabic	ar	ara	ar-JO
Kazakhstan	KZ	KAZ	Kazakh	kk	kaz	kk-KZ
Kenya	KE	KEN	Kiswahili	sw	swa	sw-KE
Korea	KR	KOR	Korean	ko	kor	ko-KR
Kuwait	KW	KWT	Arabic	ar	ara	ar-KW
Kyrgyzstan	KG	KGZ	Kyrgyz	ky	kir	ky-KG

Lao P.D.R.	LA	LAO	Lao	lo	lao	lo-LA
Latin, Algeria	DZ	DZA	Tamazight	tzm	tzm	tzm-Latn-DZ
Latin, Azerbaijan	AZ	AZE	Azeri	az	aze	az-Latn-AZ
Latin, Bosnia and Herzegovina	BA	BIH	Croatian	hr	hrb	hr-BA
Latin, Bosnia and Herzegovina	BA	BIH	Bosnian	bs	bsb	bs-Latn-BA
Latin, Bosnia and Herzegovina	BA	BIH	Serbian	sr	srs	sr-Latn-BA
Latin, Canada	CA	CAN	Inuktitut	iu	iku	iu-Latn-CA
Latin, Montenegro	ME	MNE	Serbian	sr	srp	sr-Latn-ME
Latin, Nigeria	NG	NGA	Hausa	ha	hau	ha-Latn-NG
Latin, Serbia	RS	SRB	Serbian	sr	srp	sr-Latn-RS
Latin, Serbia and Montenegro	CS	SCG	Serbian	sr	srp	sr-Latn-CS
Latin, Uzbekistan	UZ	UZB	Uzbek	uz	uzb	uz-Latn-UZ
Latvia	LV	LVA	Latvian	lv	lav	lv-LV
Lebanon	LB	LBN	Arabic	ar	ara	ar-LB
Libya	LY	LBY	Arabic	ar	ara	ar-LY
Liechtenstein	LI	LIE	German	de	deu	de-LI
Lithuania	LT	LTU	Lithuanian	lt	lit	lt-LT
Luxembourg	LU	LUX	Luxembourgish	lb	ltz	lb-LU
Luxembourg	LU	LUX	German	de	deu	de-LU
Luxembourg	LU	LUX	French	fr	fra	fr-LU
Malaysia	MY	MYS	Malay	ms	msa	ms-MY
Malaysia	MY	MYS	English	en	eng	en-MY
Maldives	MV	MDV	Divehi	dv	div	dv-MV
Malta	MT	MLT	Maltese	mt	mlt	mt-MT
Mexico	MX	MEX	Spanish	es	spa	es-MX
Mohawk	CA	CAN	Mohawk	moh	moh	moh-CA
Monaco	MC	MCO	French	fr	fra	fr-MC
Morocco	MA	MAR	Arabic	ar	ara	ar-MA

Nepal	NP	NPL	Nepali	ne	nep	ne-NP
Netherlands	NL	NLD	Dutch	nl	nld	nl-NL
Netherlands	NL	NLD	Frisian	fy	fry	fy-NL
New Zealand	NZ	NZL	Maori	mi	mri	mi-NZ
New Zealand	NZ	NZL	English	en	eng	en-NZ
Nicaragua	NI	NIC	Spanish	es	spa	es-NI
Nigeria	NG	NGA	Yoruba	yo	yor	yo-NG
Nigeria	NG	NGA	Igbo	ig	ibo	ig-NG
Norway	NO	NOR	Norwegian, Bokmål	nb	nob	nb-NO
Norway	NO	NOR	Sami, Northern	se	sme	se-NO
Norway	NO	NOR	Norwegian, Nynorsk	nn	nno	nn-NO
Norway	NO	NOR	Sami, Lule	smj	smj	smj-NO
Norway	NO	NOR	Sami, Southern	sma	sma	sma-NO
Oman	OM	OMN	Arabic	ar	ara	ar-OM
Panama	PA	PAN	Spanish	es	spa	es-PA
Paraguay	PY	PRY	Spanish	es	spa	es-PY
Peru	PE	PER	Quechua	quz	qup	quz-PE
Peru	PE	PER	Spanish	es	spa	es-PE
Philippines	PH	PHL	Filipino	fil	fil	fil-PH
Poland	PL	POL	Polish	pl	pol	pl-PL
Portugal	PT	PRT	Portuguese	pt	por	pt-PT
PRC	CN	CHN	Tibetan	bo	bod	bo-CN
PRC	CN	CHN	Yi	ii	iii	ii-CN
PRC	CN	CHN	Uyghur	ug	uig	ug-CN
Puerto Rico	PR	PRI	Spanish	es	spa	es-PR
Qatar	QA	QAT	Arabic	ar	ara	ar-QA
Republic of the Philippines	PH	PHL	English	en	eng	en-PH
Romania	RO	ROU	Romanian	ro	ron	ro-RO
Russia	RU	RUS	Russian	ru	rus	ru-RU

Russia	RU	RUS	Tatar	tt	tat	tt-RU
Russia	RU	RUS	Bashkir	ba	bak	ba-RU
Russia	RU	RUS	Yakut	sah	sah	sah-RU
Rwanda	RW	RWA	Kinyarwanda	rw	kin	rw-RW
Saudi Arabia	SA	SAU	Arabic	ar	ara	ar-SA
Senegal	SN	SEN	Wolof	wo	wol	wo-SN
Simplified, PRC	CN	CHN	Chinese	zh	zho	zh-CN
Simplified, Singapore	SG	SGP	Chinese	zh	zho	zh-SG
Singapore	SG	SGP	English	en	eng	en-SG
Slovakia	SK	SVK	Slovak	sk	slk	sk-SK
Slovenia	SI	SVN	Slovenian	sl	slv	sl-SI
South Africa	ZA	ZAF	Setswana	tn	tsn	tn-ZA
South Africa	ZA	ZAF	isiXhosa	xh	xho	xh-ZA
South Africa	ZA	ZAF	isiZulu	zu	zul	zu-ZA
South Africa	ZA	ZAF	Afrikaans	af	afr	af-ZA
South Africa	ZA	ZAF	Sesotho sa Leboanso		nso	nso-ZA
South Africa	ZA	ZAF	English	en	eng	en-ZA
Spain, International Sort	ES	ESP	Spanish	es	spa	es-ES
Sri Lanka	LK	LKA	Sinhala	si	sin	si-LK
Sweden	SE	SWE	Swedish	sv	swe	sv-SE
Sweden	SE	SWE	Sami, Northern	se	smf	se-SE
Sweden	SE	SWE	Sami, Lule	smj	smk	smj-SE
Sweden	SE	SWE	Sami, Southern	sma	smb	sma-SE
Switzerland	CH	CHE	Romansh	rm	roh	rm-CH
Switzerland	CH	CHE	German	de	deu	de-CH
Switzerland	CH	CHE	Italian	it	ita	it-CH
Switzerland	CH	CHE	French	fr	fra	fr-CH
Syllabics, Canada	CA	CAN	Inuktitut	iu	iku	iu-Cans-CA
Syria	SY	SYR	Syriac	syr	syr	syr-SY
Syria	SY	SYR	Arabic	ar	ara	ar-SY

Thailand	TH	THA	Thai	th	tha	th-TH
Traditional Mongolian, PRC	CN	CHN	Mongolian	mn	mon	mn-Mong-CN
Traditional, Hong Kong S.A.R.	HK	HKG	Chinese	zh	zho	zh-HK
Traditional, Macao S.A.R.	MO	MAC	Chinese	zh	zho	zh-MO
Traditional, Taiwan	TW	TWN	Chinese	zh	zho	zh-TW
Trinidad and Tobago	TT	TTO	English	en	eng	en-TT
Tunisia	TN	TUN	Arabic	ar	ara	ar-TN
Turkey	TR	TUR	Turkish	tr	tur	tr-TR
Turkmenistan	TM	TKM	Turkmen	tk	tuk	tk-TM
U.A.E.	AE	ARE	Arabic	ar	ara	ar-AE
Ukraine	UA	UKR	Ukrainian	uk	ukr	uk-UA
United Kingdom	GB	GBR	Welsh	cy	cym	cy-GB
United Kingdom	GB	GBR	Scottish Gaelic	gd	gla	gd-GB
United Kingdom	GB	GBR	English	en	eng	en-GB
United States	US	USA	English	en	eng	en-US
United States	US	USA	Spanish	es	spa	es-US
Uruguay	UY	URY	Spanish	es	spa	es-UY
Vietnam	VN	VNM	Vietnamese	vi	vie	vi-VN
Yemen	YE	YEM	Arabic	ar	ara	ar-YE
Zimbabwe	ZW	ZWE	English	en	eng	en-ZW

SLURP

All tables from an underlying datasource can be imported all at once using the SLURP Command. SLURP does not import views. SLURP functions against TARGIT InMemory Databases, Access & Microsoft SQL Server.

Note: Other source DBs are currently not supported by the slurp function.

Syntax:

SLURP <source_name>

Example:

```
SLURP A1
/*other sources can also be added*/
IMPORT decimal_test= source2.decimal_test
```

TRY/CATCH

If using code that potentially can fail from time to time it is possible to use a Try Catch-statement and programmatically handle the error as opposed to having the application error out.

Syntax for Try/Catch statement:

```
Try
    // Commands to try to execute
End Try
Catch
    // Commands to execute when an error occurs
End Catch
```

Example:

```
TRY     // Try the commands between this line and End Try
   IMPORT [budget]=excelfile.{select * from [c:\budget\custombudgetfile.xlsx]}   // Import budget from file
that may be edited by people in the organization
END TRY
CATCH  // If the try fails the commands in this catch block will be executed
   PRINT 'An error occurred loading the custom budget file - will default to loading the audited budget file'
   IMPORT [budget]=excelfile.{select * from [c:\budget\auditedbudgetfile.xlsx]}  // Load the budget file that
we know
END CATCH
```

VAREXISTS

VarExists can be used to check whether a variable has been defined in the script during the previous execution steps or not.

Syntax:

VarExist(**@variable_name**)

Example:

```
/*
  Check whether the varilable exists
  In this example the variable @notfound is not defined, therefore it will show the
  message 'Variable not defined' during execution
*/

IF VarExists(@notfound)=FALSE THEN
   PRINT 'Variable not defined!'
ELSE
   PRINT 'Variable @notfound is defined'
END IF
```

WHILE Loop

The WHILE loop iterates while the expression evaluates to TRUE.
With the WHILE loop you can either increment a variable or make use of a table cursor.

Syntax with table cursor:

WHILE *table.EOF=false*
PRINT 'We just read the column and got the value '+*table.Column_name_in_table*
MOVENEXT *table*
LOOP

Syntax with variable:

WHILE *@variable<max_value* PRINT 'Variable is now '+*@variable* SET *@variable=@variable+increment_value*
LOOP

Iterating a value until condition is reached.

```
DECLARE @Company varchar

/* Create the internal table SalesCompanies in the InMemory data store */
IMPORT SalesCompanies=me.{SELECT 'Denmark' AS TablePrefix UNION ALL SELECT 'United States'}

/* While Loop for each company in the SalesCompanies (Denmark/United States)  (see above) */
WHILE SalesCompanies.EOF=false

/* Set up the variable company */
SET @Company=SalesCompanies.TablePrefix

/* Read the data from the SQL server connection from the table substituting part of the tablename - SQL
becomes e.g. SELECT  from Denmark_Sales */
IMPORT factSalesData=sqldb.{SELECT * FROM @@COMPANY_Sales}
                MOVENEXT SalesCompanies  / Move cursor to the next Company
LOOP    /* Iterate the loop */
DECLARE @i as int


SET @i=1

WHILE @i<=10
                PRINT @i
                SET @i=@i+1
LOOP
```

Eexport to SQL SERVER

Data can be exported from an InMemory Database format to SQL Server using the EXPORT command.

Syntax:

EXPORT {destination data source}.[Destination Table Name] from {source data source}.[Source Table Name]

Example:

EXPORT a1.orders FROM me.orders

Adding WITH TRUNCATE forces truncation of the existing table before export.

Example:

EXPORT with TRUNCATE a1.orders FROM me.orders

To help prevent accidental deletion of data, it is necessary to run the command ALLOW TRUNCATE on the target data source before using the command.

allow truncate on a1

Adding WITH DROP forces dropping of the existing table before export.

Example:

EXPORT WITH DROP a1.orders FROM me.orders

To help prevent accidental deletion of data, it is necessary to run the command WITH DROP, run the command ALLOW DROP on the target data source before using the command.

Example:

ALLOW DROP ON a1

Complete Example:

DATASOURCE db1=local targitdb 'c:\databases\data\northwind.targitdb'

DATASOURCE a1=SQLSERVER 'Data Source=localhost;Initial Catalog=ir_export_test;User Id=ir;Password=mypassword;'

slurp db1

allow drop on [a1]

export with drop a1.orders from me.orders

export with drop a1.[order details] from me.[order details]

export with drop a1.[customer] from me.[customers]

export with drop a1.[employees] from me.[employees]

export with drop a1.[Products] from me.[products]

export with drop a1.[shippers] from me.[shippers]

export with drop a1.[suppliers] from me.[suppliers]

Server and Import optimizations

LAZY LOAD

By default, the entire database file is loaded into memory. With the lazy loading feature, the database will load table columns on demand when they are needed. If a column is never used in a query, then less memory is used since that column never needs to be loaded.

The potential small trade off is in performance as a column will need to be loaded before it can be used in a query. Also, the lazy load feature will use additional disk space in the InMemory data directory as temporary copies of the targitdb data files need to be created. The temporary copy is needed to avoid the original file being locked and can therefore still be updated. Lazy load is supported from TARGIT 2021 version of TARGIT InMemory and requires that the database is created using tiImport from that version or later.

To use lazy loading in the tiServer process, **LazyLoad=true** needs to be added to the targitdb.ini file and the tiServer restarted. In the default mode when using lazy loading, columns will stay loaded after first use. However, columns can be configured to unload after a period of not being used. This period is specified in the **lazyload_unload_column_timer** option in targitdb.ini. The unload time is specified in minutes, e.g. **lazyload_unload_column_timer=15**.

TARGIT Query Tool can also open a database in lazy load mode.

To use the lazy loading feature in ETL scripts, add the lazyload keyword when declaring a datasource. e.g. DATASOURCE A1 = LAZYLOAD local inmemorydb ‘c:\targit datadir\mydatabase.targitdb’

USEDISK

By default, tiImport.exe takes advantage of RAM that is available to it. However, you may not want to use so much memory, in which case it is possible to *buffer data to disk* while a data extraction is being run by including the keyword USEDISK.

Example: IMPORT table=**USEDISK** a1.table1

This loads data in chunks of 1,000,000 rows, saving to a temp file. Data is then Columnized to a different file containing an InMemoryTable. The table is loaded in its entirety into memory only when finished.

USEDISK FLUSH

USEDISK FLUSH performs the same operation in the same way as USEDISK, except the end-result table is saved to a single (flushed) table that is then consolidated when SAVE is called at the end of the Import Script.

Example: IMPORT table=USEDISK FLUSH a1.table1

Note: Tables are not available for Querying, Updates or Deletes with the FLUSH variant.

By using USEDISK FLUSH, it is possible to generate larger extract files using a lot less memory as it only needs to be able to contain one column of information at a time. Extraction times will take about 5-10% longer. Do not use USEDISK FLUSH if this table needs to be referenced in a later part of the import script.

Sufficient storage is required during the import phase for an extra copy of this table *(on the disk in which the targitdb file is being saved)*.

DO PARALLEL …. END PARALLEL

If you want to import a series of tables in parallel, to speed up your import process, the PARALLEL statement can be used. This is best used with multiple import statements. While other statements can be put in the parallel block , ideally each statement should be self-contained, and not depend on the sequence of execution and any other statement in the parallel block.

Syntax:

```
DO PARALLEL
        Statement 1
        ……
        Statement N
END PARALLEL
```

Example:

Imports 3 tables in Parallel

```
DO PARALLEL

  IMPORT TABLE1 = A1.TABLE1

  IMPORT TABLE2 = A1.TABLE2

  IMPORT TABLE3 = A1.TABLE3

END PARALLEL
```

Refresh TARGIT InMemory Database

After building the IMP file the targitDB file is generated by the tiImport command line tool.

If you're taking advantage of the TARGIT InMemory Scheduler component the entire process is automated and you only need to deploy your project to the TARGIT InMemory Scheduler Service.

Should you want to execute an import script externally you can take advantage initiating the refresh of either calling the tiImport application and reloading the data by using e.g. a batch file or if integrating TARGIT InMemory Database into a Microsoft SQL Server solution a dedicated custom task is available as a custom task called TARGIT SSIS InMemory Import SSIS Compoent.

On a typical on-premise solution for a medium sized company (~120GB of data) the import process will typically run for 2 to 15 minutes, depending on the amount of data and performance of hardware. In our experience this is a lot less than what you would expect when refreshing e.g. an analysis services cube.

When running the tiImport command it is important to notice that the output targitdb file will be created with the same same name as the imp file.

After generation the targitdb file built must be moved to the TARGIT InMemory Database Data-folder and loaded into memory with the tiLoad command..

A batch file automating the full process would typically be structure like this:

Note: *the password that is being used is the one entered during the installation of the TARGIT InMemory Database*

REM Create targitDB
"C:\Program Files\TARGIT\TARGIT InMemoryDB\tiImport.exe" c:*path_to_impfile*\my_import.imp

REM Move targitDB to the TARGIT Folder updated database for the server.
copy c:*path_to_impfile*\my_import.targitdb "c:\programdata\targit\targit InMemoryDB\data\"

REM Reload the newly
"C:\Program Files\TARGIT\TARGIT InMemoryDB\tiLoad.exe" database=my_import;pwd=*password* /reload

TARGIT SSIS InMemory Custom Import Task

As an alternative to building and loading the targitdb into memory using the command promt the TARGIT SSIS InMemory Import Custom Task can be used and is described below:

Point to the script (.imp) file and input your InMemory Database password and the task will take care of the rest. The task will build the targitdb file based on the script file and copy it to the TARGIT InMemoryDB folder (default: C:\ProgramData\TARGIT\TARGIT InMemoryDB\Data).

Keep in mind that while data is being prepared for the targitdb file its being loaded into memory by the tiImport program. If you are not unloading the existing version of the InMemory database you need to have enough memory told hold 2 copies of the datasets in Memory. If not you will need to run the **tiload /unload** script to remove the current version from memory before loading a new version of the InMemory database.

To ensure data is unloaded from memory before creating a new targitdb file when using the SSIS component just check the Unload database before run box from the SSIS task.

/*Upload into Memory*/
"C:\Program Files\TARGIT\TARGIT InMemoryDB\tiLoad.exe" database=inmemoryMAC;pwd=password /unload

/* Create targitDB */
"C:\Program Files\TARGIT\TARGIT InMemoryDB\tiImport.exe" c:*path_to_impfile*\my_import.imp

/* Move targitDB to the TARGIT Folder */
copy c:*path_to_impfile*\my_import.targitdb "c:\programdata\targit\targit InMemoryDB\data\"
"C:\Program Files\TARGIT\TARGIT InMemoryDB\tiLoad.exe" database=my_import;pwd=*password* /reload

Note: *the password that is being used is the one entered during the installation.*
Running the update from an SSIS Package via (SQL Server 2012, SQL Server 2014)

tiLoad

The tiLoad command send commands to the tiServer database engine related to load, refreshing or unloading a database from memory.

The command must have database name and password specified as described in the syntax

host= Name of server with the TARGIT InMemory Database Engine running (optional, default is localhost)

database= Name of the database (required)

pwd= Password for the TARGIT InMemory instance (required)

/unload Unloads the database specified from memory

/reload Loads a new instance of the database from the InMemory data store and hot swaps to the new version

Syntax:

tiLoad host=*servername*;database=*databasename*;pwd=*database_engine_password* [/reload|/unload]

Example:

tiLoad database=mydatabase;pwd=dbpass123 /reload

tiLoad database=mydatabase;pwd=dbpass123 /unload

Picking up new TARGIT InMemory data

If a TARGIT InMemory Database (.targitdb) is repopulated with an import of the queries from the existing database it will not automatically reflect the newly-imported data-set (the data has been imported to the .targitdb file, but not loaded into memory).

To refresh this data there are two possible methods: requesting the database to reload the data utilizing the tiLoad command or simply restarting the tiServer Database Engine service. Below are examples of both methods:

1. Reload using tiLoad

tiLoad database=mydatabase;pwd=dbpass123 /reload

2. The second method to restart the server will impact the availability of the server requiring all TARGIT users to log out and in again, secondly all databases will again have to be loaded from the disk causing heavier disk IO on the server, therefore the tiLoad command is the suggest method to perform this operation.

The server can however be restart using these commands:
net stop “tiServer”
net start “tiServer”

InMemory Data Drivers

Accompanying the TARGIT InMemory Database is a range of specialized drivers for data sources that we see a need to have a dedicated connection to. This list is dynamic and drivers will be added or retired as time passes. Here, please consider the documented drivers as examples of use. The complete list of current drivers is available for the latest version of TARGIT ETL Studio as a separate installation package also included in the TARGIT installation package.

CSV

The CSV driver is capable of loading CSV files into the InMemory Database.

Connection String Parameters:

Keyword	*Value*	*Description*
HasHeaders	true/false	Flag for whether the file has a header as the first row or not.
loadasstring	true/false	Flag for the driver to perceive all rows as being strings and thereby not interpreting data types
type	local/url	Use 'local' if the file is placed on the local file system. The driver also supports retrieving data from a regular URL without password protections
delimiter	;	Sets the delimiter character. The string quotes are automatically detected
encoding	Windows-1252	Determines the character set. The encoding specified must be in the list of encodings available in the .NET API

Query parameters:

Retrieving the data from the file is done by performing a SELECT command, e.g.

SELECT * FROM [sample_csv_file.csv]

The driver does not support retrieving individual columns. If this is required the solution is to load the data into another table after the import finishes.

Example:

LOAD ASSEMBLY 'TARGIT.Csv.dll'
DATASOURCE [CSV] = DOTNET CONNECTION 'TARGIT.CSV.CsvConnection'
'HasHeaders=true;loadasstring=true;type=local;delimiter=;;encoding=Windows-1252;'

IMPORT [BudgetData] = [CSV].{**SELECT** * **FROM** [C:\DEMODATA\BUDGETDATA.CSV]}

Excel

The Excel driver supports loading data from either XLS or XLSX files.

Connection String Parameters:

Keyword	*Value*	*Description*
ignoreemptyvalues	true/false	Sets the flag whether the driver should ignore rows where all numeric values containt 0 and all string values are empty
datasource	C:\myfile.xlsx	Filename including path
detectionrowscount	100	Number of rows from file to sample when detecting data type

Query Parameters:

Retrieving the data from the file is done by performing a SELECT command, e.g:

SELECT * FROM [Case_Sensitive_Name_Of_Sheet]

Please note that the name of the worksheet is CASE SENSITIVE, also the driver does not support retrieving individual columns. If this is required the solution is to load the data into another table after the import finishes. In case it's necessary to load all sheets in a file the list of sheets can be accessed by the sys.sheets function e.g.

SELECT * FROM [sys.sheets]

Example:

LOAD ASSEMBLY 'TARGIT.Excel.dll'

DATASOURCE [ExcelFile] = DOTNET CONNECTION 'TARGIT.Excel.ExcelConnection'
'ignoreemptyvalues=false;datasource=C:\DemoData\Budget.xlsx;detectionrowscount=100;'

IMPORT [ListOfSheetsInBudget] = [ExcelFile].{**select** * **from** [sys.sheets]}

IMPORT [Budget] = [ExcelFile].{**select** * **from** [My Budget Sheet]}

FileList

The FileList driver is designed to retrieve a listing of files on a specified path in the local filesystem.

Connection String Parameters:

There are no parameters to set in the connection string.

Query Parameters:

The filelist can be retrieved by a SELECT command in the following form:

SELECT * FROM [*path_to_folder*] WHERE filename='*filemask*' AND directory='*current|full*'

Current List all files in the specified folder

Full List all files in the specified folder and recurse subdirectories

Example:

LOAD ASSEMBLY 'TARGIT.FileList.dll'

DATASOURCE [FileList] = DOTNET CONNECTION 'TARGIT.FileList.FileListConnection' ''

IMPORT [csvFileList] = [FileList].{**SELECT** * **FROM** [D:\DemoData] **WHERE** filename='*.csv' **AND** directory='full'}

Fixed Width

The Fixed Width driver is used to load data from a Fixed Width text file. The driver does not handle data types but loads the data into string that then can be parsed using the InMemory functions.

Connection String Parameters:

Keyword	Value	Description
encoding	Windows-1252	Determines the character set. The encoding specified must be in the list of encodings available in the .NET API

Query Parameters:

The query parameters for the Fixed Width driver requires specification of the size of the individual columns in the format

[column_width_as_integer] columnname

for each column, which in full syntax would be:

SELECT [column_width_as_integer] columnname FROM [path_to_fixedwidth_file]

This is best described with the following data example:

Entry	Per.	Post Date	GL Account	Description	Srce.	Cflow	Ref.	Post	Debit	Credit	Alloc.
16524	01	10/17/2012	3930621977	TXNPUES	S1	Yes	RHMXWPCP	Yes		5,007.10	No
191675	01	01/14/2013	2368183100	OUNHQEX XUFQONY	S1	No		Yes		43,537.00	Yes
191667	01	01/14/2013	3714468136	GHAKASC QHJXDFM	S1	Yes		Yes	3,172.53		Yes
191673	01	01/14/2013	2632703881	PAHFSAP LUVIKXZ	S1	No		Yes	983.21		No
80495	01	11/21/2012	2766389794	XDZANTV	S1	Yes	TGZGMOXG	Yes		903.78	Yes
80507	01	11/21/2012	4609266335	BWWYEZL	S1	Yes	USUKVMZO	Yes		670.31	No
80509	01	11/21/2012	1092717420	QJYPKVO	S1	No	DNUNTASS	Yes		848.50	Yes
80497	01	11/21/2012	3386366766	SOQLCMU	S1	Yes	BRHUMGJR	Yes		7.31	Yes
191669	01	01/14/2013	5905893739	FYIWNKA QUAFDKD	S1	Yes		Yes	9,167.93		Yes
191671	01	01/14/2013	2749355876	CBMJTLP NGFSEIS	S1	Yes		Yes	746.70		Yes
191674	01	01/14/2013	4530359106	OTAVZGH ZUQFISZ	S1	Yes		No	7,035.74		Yes
244819	01	02/04/2013	4679391677	EGHLQTI ABE	S1	Yes		No		89,947.13	No
96062	01	11/30/2012	5996493062	KTSVTADFF EHEHFMX	S1	Yes	UBNQLRCC	Yes	7.10		Yes
16527	01	10/17/2012	5595769375	ILCVJYC	S1	Yes	HCVZOUMY	Yes		321.19	Yes
191670	01	01/14/2013	1948028853	RPPDCWC UWODNIO	S1	Yes		No	9,293.80		No
191672	01	01/14/2013	4938823703	CTMDXXP HXOXVFF	S1	Yes		No	175.00		Yes
191668	01	01/14/2013	4207018603	DBZZULF QGDZQMD	S1	Yes		Yes	206.26		Yes

Being transformed with the following command:

SELECT [8] Entry, [4] Period, [12] PostDate, [13] GLAccount,[26] Description, [6] Srce, [4] CFlow, [10] Ref, [4] Post, [18] Debit, [20] Credit,[6] Alloc FROM[d:\demodata\fixedwidthexample.txt]

Will be transformed into:

Entry	Period	PostDate	GLAccount	Description	Srce	CFlow	Ref	Post	Debit	Credit	Alloc
80509	01	11/21/2012	1092717420	QJYPKVO	S1	No	DNUNTASS	Yes		848.50	Yes
80497	01	11/21/2012	3386366766	SOQLCMU	S1	Yes	BRHUMGJR	Yes		7.31	Yes
191669	01	01/14/2013	5905893739	FYIWNKA QUAF...	S1	Yes		Yes	9,167.93		Yes
191671	01	01/14/2013	2749355876	CBMJTLP NGFS...	S1	Yes		Yes	746.70		Yes
191674	01	01/14/2013	4530359106	OTAVZGH ZUQ...	S1	Yes		No	7,035.74		Yes
244819	01	02/04/2013	4679391677	EGHLQTI ABE	S1	Yes		No		89,947.13	No
96062	01	11/30/2012	5996493062	KTSVTADFF EH...	S1	Yes	UBNQLRCC	Yes	7.10		Yes
16527	01	10/17/2012	5595769375	ILCVJYC	S1	Yes	HCVZOUMY	Yes		321.19	Yes
191670	01	01/14/2013	1948028853	RPPDCWC UW...	S1	Yes		No	9,293.80		No
191672	01	01/14/2013	4938823703	CTMDXXP HXO...	S1	Yes		No	175.00		Yes
191668	01	01/14/2013	4207018603	DBZZULF QGDZ...	S1	Yes		Yes	206.26		Yes

Example:

LOAD ASSEMBLY 'TARGIT.FixedWidth.dll'

DATASOURCE [Fixed Width] = DOTNET CONNECTION 'FixedWidth.FixedWidthConnection' 'encoding=Windows-1252;'

IMPORT [FixedWidthExample] = [Fixed Width].{**SELECT** [8] Entry, [4] Period, [12] PostDate, [13] GLAccount,[26] Description, [6] Srce, [4] CFlow, [10] Ref, [4] Post, [18] Debit, [20] Credit,[6] Alloc **FROM** [d:\demodata\fixedwidthexample.txt]}

Google Big Query

To query Google BigQuery a file with authentication information (secrets file) must be obtained from the Google BigQuery console. To utilize large result sets it is required that the user authenticating must have permissions to create tables when executing the query job.

Connection String Parameters:

Key	*Value*	*Description*
ProjectId	myerpdata-1234	The project id
Serviceaccountsecretspath	myerpdata-1234-123123.json	Path the the with authentication

Query parameters:

The query executed must be a standard Google BigQuery SQL syntax, additional it is possible to specify additional options in this form

SELECT column_name FROM table_name [OPTIONS:optionlist]

Everything specified in the options section will be parsed and supplied to the driver as parameters during runtime and will not be executed as a part of the query.

An example of an options list could be:
[OPTIONS:allowLargeResults=true;destinationDatasetID=MyTempTables;destinationTableID=tmpSalesTrans20160401;writeDisposition=WRITE_TRUNCATE]

For explanation of the options please refer to the Google BigQuery documentation.

The query also takes a special command in the form of (CASE SENSITIVE!):

SELECT * FROM INFORMATION_SCHEMA.tables

This will return a complete list of tables available to the user with additional meta data information as medication date etc. This can be extended with this: (OPTIONS:infodatasetid=dims) restricting the returned information to the dataset dims:

SELECT * FROM INFORMATION_SCHEMA.tables (OPTIONS:infodatasetid=facts)

The meta data information returned is:

TableProject string
TableDataSet string
TableName string
SizeKb double
RowCount long
CreationDate datetime
ModifiedDate datetime
DataLocation string

Example:

LOAD ASSEMBLY ‘TARGIT.GoogleBigQuery.dll'

DATASOURCE bigquery = DOTNET CONNECTION 'TARGIT.GoogleBigQuery.GoogleBigQueryConnection' 'projectid=project_id;serviceaccountsecretspath=d:\demo\CasualClothing-2070791221.json;'

IMPORT tablelist=bigquery.{**SELECT** * **FROM** INFORMATION_SCHEMA.tables}

IMPORT SalesData = bigquery.{**SELECT** * **from** facts.SalesInformation}

MongoDB

The MongoDB driver is a specialized driver that can turn the documents with all specific arrays into columnar format making it possible to use with TARGIT InMemory Database and TARGIT Decision Suite.

It is optimized for speed and has been designed so developers without knowledge of MongoDB can write SQL a simple SQL select statement to extract data from MongoDB.

Connection String Parameters:

The connection string parameter must be given as a standard MongoDB connection string, in this form.

mongodb://username:password@servername/databasename'

In case you're connecting to a MongoDB version 2.x the authentication method must be set to legacy MongoDB, this is done by changing the parameter to this form:

mongodb://username:password@servername/?authSource=nameofauthdb&authMechanism=MONGODB-CR

Query Parameters:

The MongoDB driver supports queries in the following formats:

SELECT *<field1>,...,<fieldN>* FROM *<collection>.<array>* WHERE *<field1><operator><value>* (*AND*|*OR*) *<fieldN><operation><value>*

1. To select the document identifier of current row the special keyword: root._id must be specified
Example: SELECT *root._id* FROM *values*

This query selects identifiers for collection values.

2. To limit the amount of rows returned from a query the LIMIT clause can be utilized: LIMIT <value>
Example: SELECT *name* FROM *values LIMIT 10*

This query returns the first 10 rows from collection values.

3. If you select data direct from collection use such syntax: FROM <collection>
Example: SELECT name FROM values

This query selects name field from values collection.

4. If you want to select from array in collection use: FROM <collection>.<table>.<innerTable1>...<innerTableN>
Example: SELECT array.name FROM values.array

This query selects names field in array object "array".

5. It is possible to select index for selected array using syntax <array_name>.INDEX() (only for MongoDB version 3.2 and higher).
Example: SELECT array.name, array.INDEX() FROM values.array

6. You can select fields from object : <object_field>.<inner_field>
Example: SELECT object.id, object.name FROM values

7. If you want to select all fields from array or object , then select data by fields of object or array.
Example: SELECT *my_object* FROM *values*
This query returns all fields of object *my_object*.

8. To filter the data the WHERE clause can be used.
Example: SELECT *name* FROM *values* WHERE *count* > 10 AND *order* < 5 AND *name*="SEO"
Possible where operations: >, <, >=, <=, =, !=, IN, LIKE

9. Queries for dates must be made in the format of yyyy-MM-dd hh:mm:ss.
Example: SELECT *StartDate* FROM *date* WHERE *startDate* > "2016-12-23 12:00:00"

Example:

LOAD ASSEMBLY TARGIT.MongoDB.dll'
DATASOURCE mongodb=DOTNET CONNECTION 'MongoDB.MongoDBConnection'
'mongodb://localhost/crmdata'
IMPORT Contacts=mongodb.{**SELECT**
Personal.FirstName,Personal.Surname,Personal.Title,Personal.JobTitle,Category,PersonID
FROM Contacts}

Starting tiQuery

TARGIT InMemory Query Tool (tiQuery) can be used to execute SQL syntax against an TARGIT InMemory Database and see the results in a grid. It has a tree structure that allows you to view databases, tables and field names.

To run, execute tiQuery.exe in the Program Files\TARGIT InMemoryDB folder (double-click, or via command prompt). tiQuery can connect to a local or remote TARGITDB datasource.

To view a database in tiQuery, there are two methods:

- "Open" allows you to choose a particular .TARGITDB file, thus opening that database only.
- "Open (Lazy Load)" to enable on demand loading of tables. Table contents are brought into memory only as needed/ selected. This may prove a quicker/ less consuming way to open a large database on a system with limited resources.
- "Connect" allows you to view any database available to a particular TARGIT InMemory Database Server.

By selecting Connect, you can enter the servername (including path/ ip address if necessary), username and password.

After "Connecting", you can select which database you are currently querying via the Database dropdown at the top of the screen.

Note: there is no need to select a Database if you have selected "Open", since only one database will be available to tiQuery.

tiQuery - General Shortcuts/Tips

In tiQuery, double-clicking on a field name will add that fieldname to the text editor window. You can run a query by clicking the Execute button or by pressing F5. We strongly suggest you make use of the Limit clause to limit the number of rows returned as TARGIT InMemory Database will otherwise attempt to display all valid rows in your SQL statement.

Example:

```
SELECT *
FROM ORDERS
LIMIT 1000
```

TARGIT InMemory ETL Studio

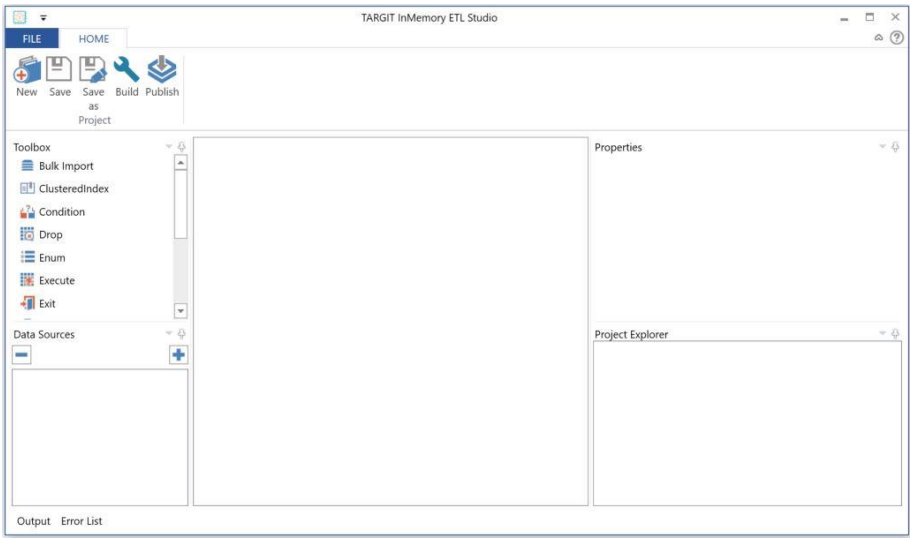
The TARGIT InMemory ETL Studio is designed to allow the user a means to create customized InMemory Databases to be consumed through TARGIT Decision Suite. This site will explain the interface and all its components, but will not go into details on developing an InMemory Database.

After having installed the application, it is simple to get started. Depending on your Operating System, simply locate the TARGIT InMemory ETL Studio icon and double click.
You may see the following dialog:

Simply click the “Log In” button to begin your journey.

Troubleshooting Tip: When the user first launches TARGIT InMemory ETL Studio, if a database is not able to be found, during the adding of a Data Source, ensure the permissions in SQL Server are setup, or when running local ensure to run TARGIT InMemory ETL Studio as **Administrator**. This will ensure that all rights and abilities will be enabled during your configuration of the TARGIT InMemory Database.

You will now see the TARGIT InMemory ETL Studio application, which might look something like this:

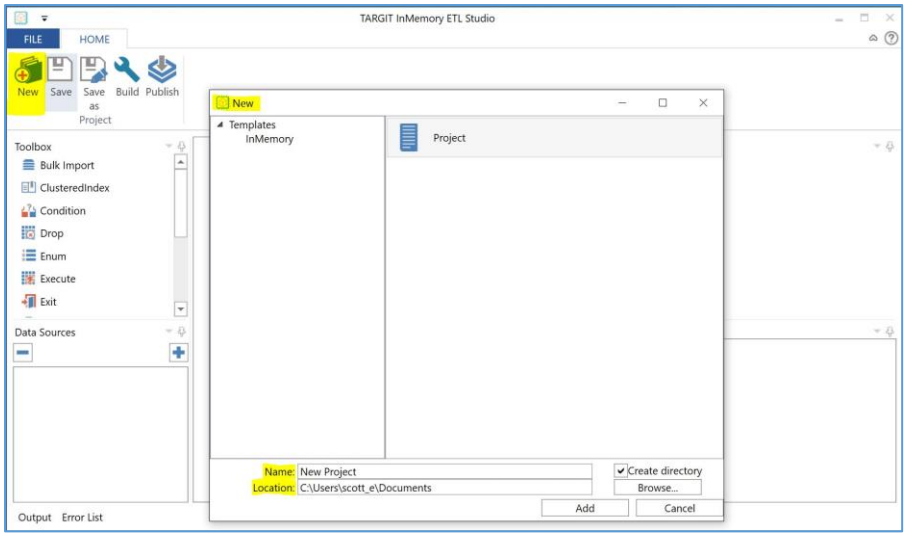


To get a detailed description about the TARGIT InMemory ETL Studio, click the links to the left.

Creating a New Project

Like all projects, they must have a beginning and an end. Using the TARGIT InMemory ETL Studio is no different. To create a New TARGIT InMemory Database, first start by Clicking "**New**" project, the button with the Red Cross on the Ribbon Bar.

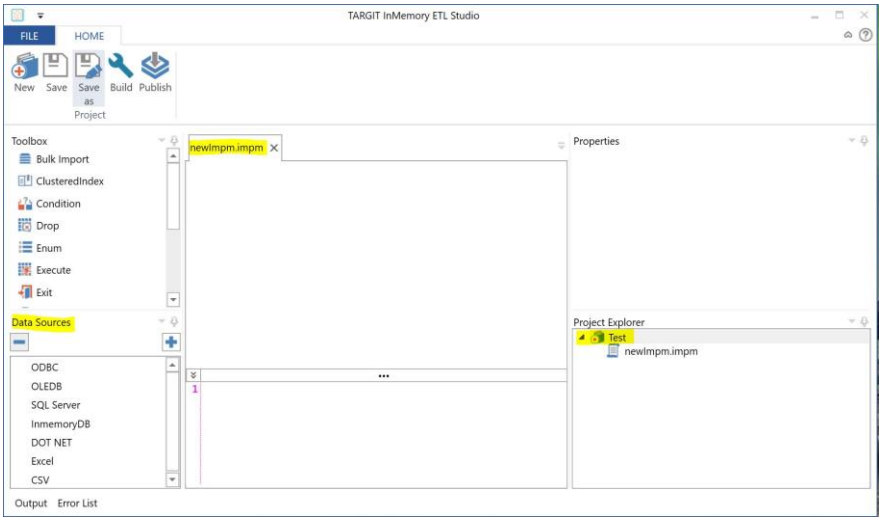
The New Project dialog appears, as shown below:



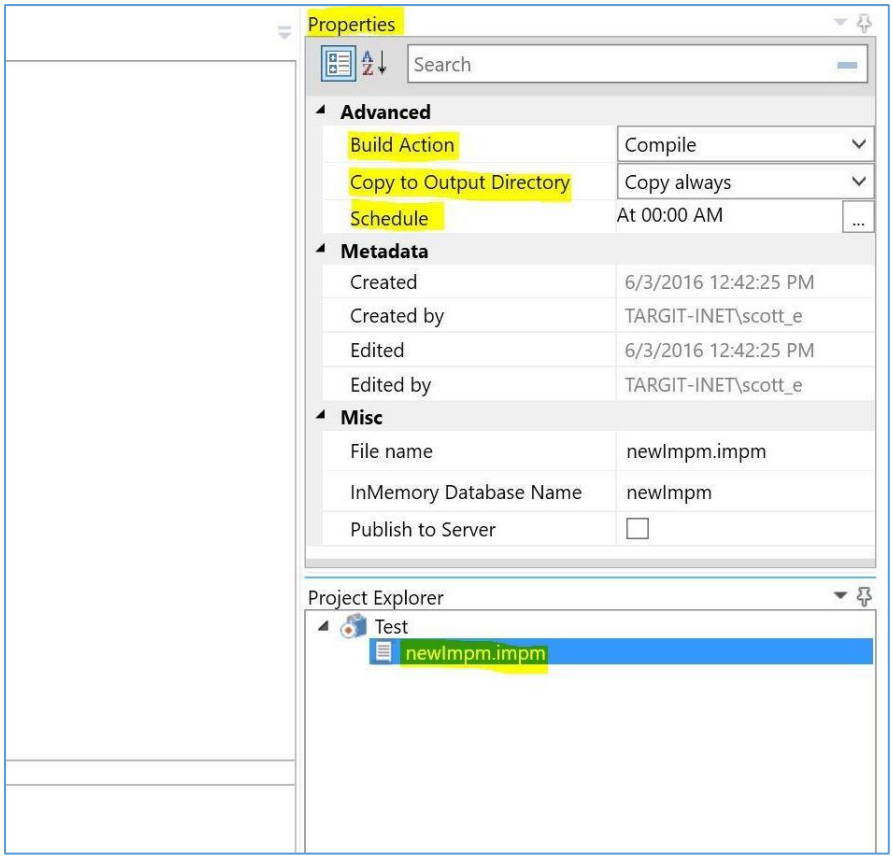
You will need to **NAME** your project, ensure the correct **LOCATION** is chosen, and select the Add button.

Your project is added, and you see you have a selection of **Data Sources** now, a **Project Workspace** and **Project Explorer** shows the new Project and Name, as well as the Import File you are about to define.

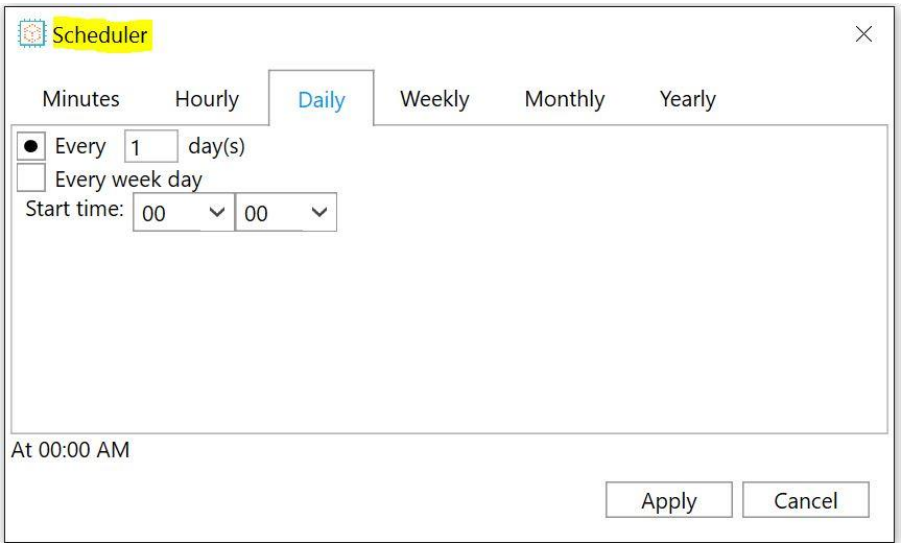
Note: If you need to rename your project, it is as simple as right clicking on the "**Project Name**" and chose "**Rename**".



When you click the **Import File (.impm)**, then the **Properties** will be shown. The user can setup the **Build Action**, **Copy to Output Directory**, and **Schedule** for the newly created script.
The user can also define the **File name**, **InMemory Database Name**, and **Publish to Server**.



- **Build Action** allows you two options Compile or None.
- **Copy to Output Directory** allows the user to, while troubleshooting, disable copying the file out to the Output Directory.
- **Schedule** allows the user to define the schedule for importing the data. Click the "**…**" icon to define the scheduling, the user will see the following dialog, which is standard scheduling details.

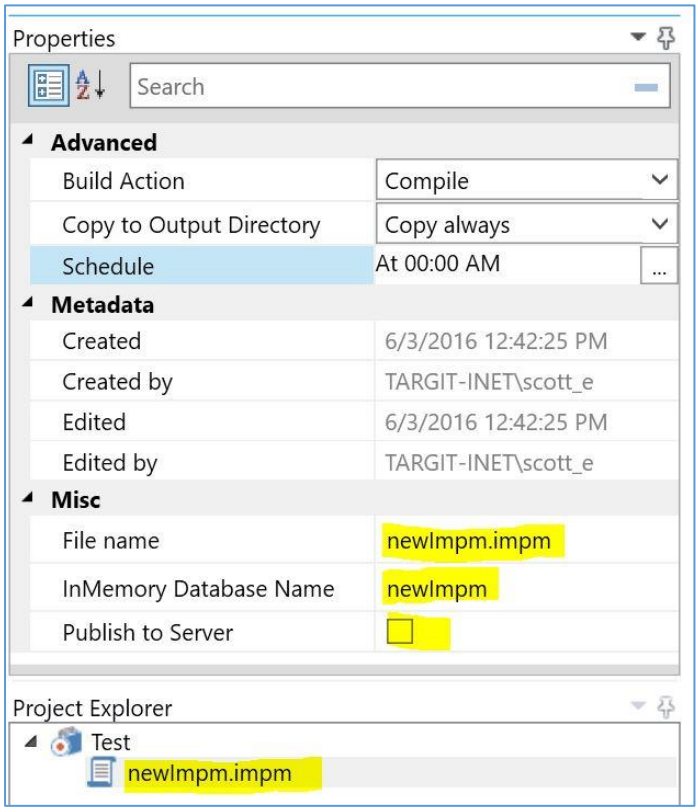


The user is able to change:

File Name of the Import file

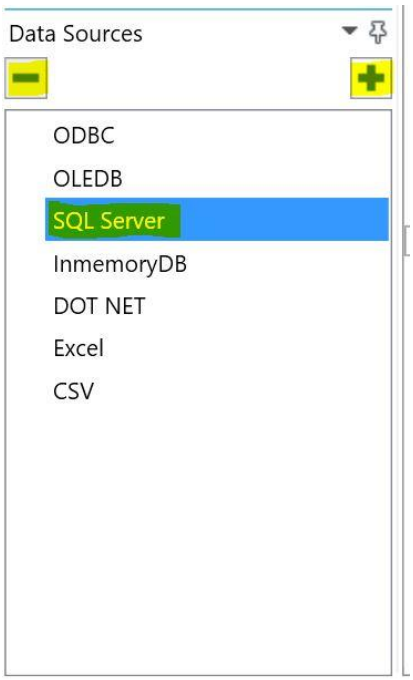
- **InMemory Database Name**, which is what is referred to in TARGIT Management portion of the TARGIT Decision Suite, to make it distinct or according to specific naming conventions.

- **Publish to the Server** or not by selecting the check box. In the case that you might have one import that needs to be run daily and another that is historical and only runs monthly. This will allow you to determine, based on your Schedule, to allow this publish to the server or not as necessary.



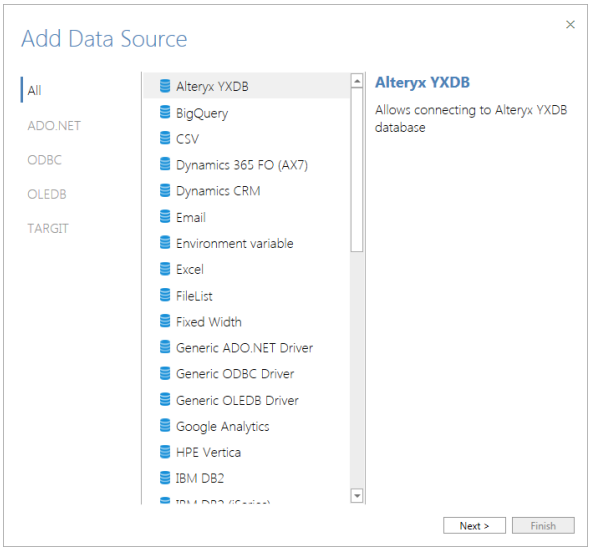
Select Data Source

Adding a Data Source has never been easier. The user can either **Right Click** the "Data Source" or select the big **Blue Plus (+)** button in the **Data Sources** section, shown below. The **Blue Minus (-)** will remove a data source.

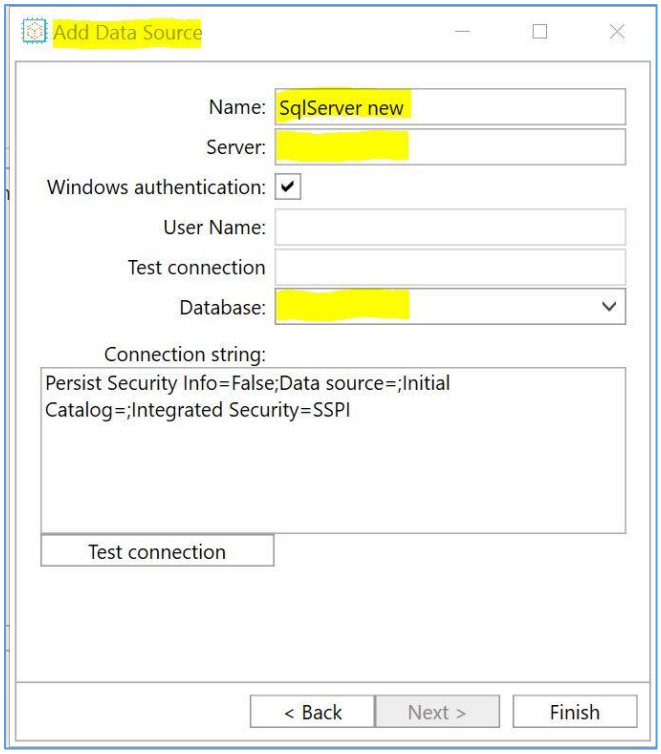


Using the Blue Plus (+) Button

Selecting the **Blue Plus (+)** button will present the following dialog, where the user can select the data Source.

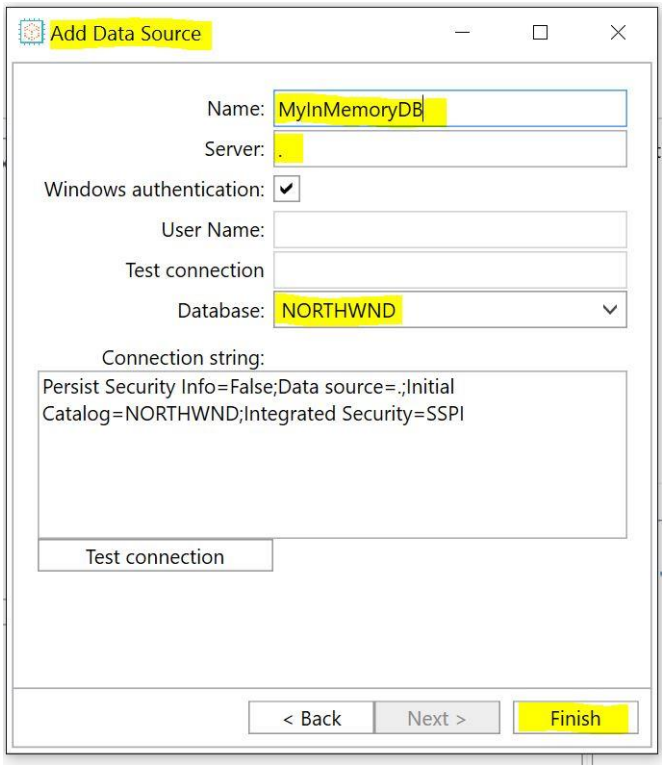


The user will see the following Add Data Source dialog, after using the **Blue Plus (+)** button or by **Right Clicking** a specific Data Source from the **Data Sources** list:



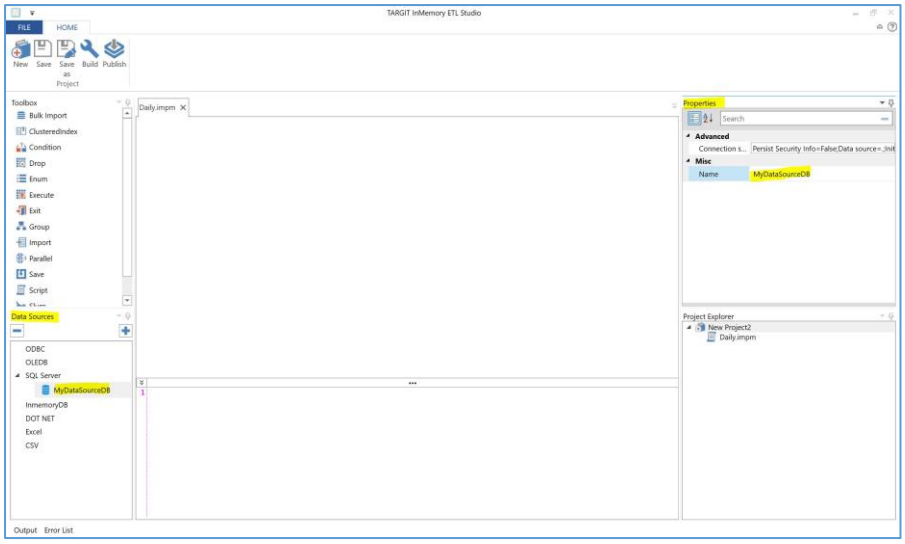
When at the Add Data Source dialog the user can, **Name** your Data Source, Identify the **Server**, and identify your existing **Database**.

Note: TARGIT InMemory does support Windows Authentication, as well Username and Connection String. The dialog might look something like the following image, just click **Finish** to complete adding a data source.



After Adding a Data Source

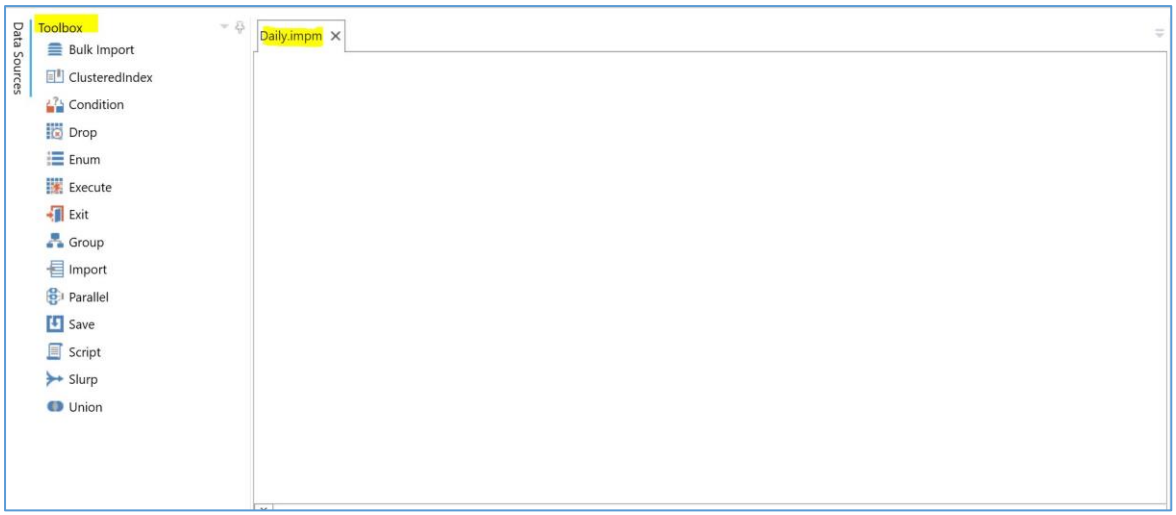
The user will see that the TARGIT InMemory ETL Studio looks something like the following. The Data Source is identified, the user can adjust the properties of the new data source as needed.



Creating an InMemory Database

A project has been created, the data source has been identified, now the user can add from your Toolbox to the Work Space and begin defining the import for the New TARGIT InMemory Database.

The user can select from any of the following icons in the Toolbox to begin the process:



Toolbox item description

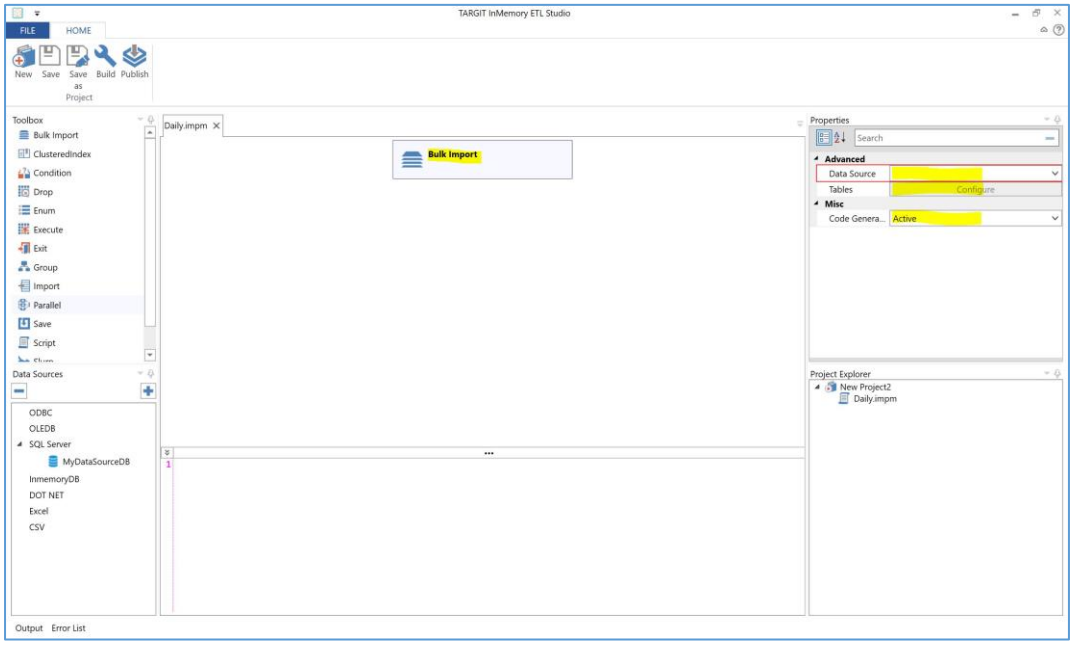
Here is a rundown of the Toolbox:

- **Bulk Import** – Similar to Import, the user is provided means to specify all tables, as well as columns to be imported from the user defined Data Source. This allows tremendous flexibility of the import, whereas Import will import only the table defined to import. The imports run serialized, so if you have a very large database, then you will probably want this to run in Parallel. See: **BULK IMPORT**

- **ClusteredIndex** – Allows the user, when the data becomes extremely big, then you can generate a clustered index. See: **CLUSTEREDINDEX**
- **Condition** – Allows the user to define a condition, like an IF Statement, if condition one is true, then execute what is in the Condition block, else execute condition two. See: **CONDITION**
- **Drop** – Allows a user to drop temporary tables that may have been created via a Script. See: **DROP**
- **Enum** – Allows the user to create an enumeration, which is a new table, used for translating one value to another as needed. See: **ENUM**
- **Execute** – Allows the user to execute a custom script. See: **EXECUTE**
- **Exit** – Allows the user to exit and update the views and exits out of a condition with an exit code, defined by the user or by default. See: **EXIT**
- **Group** – Allows the user to group toolbox items which can simplify/clean their workspace. See: **GROUP**
- **Import** – The most basic import tool, which the user will have to define the Data Source, Query, Query Type, Code Generation and Name. See: **IMPORT**
- **Parallel** – When the user is attempting to Bulk Import a large database, this will allow a user to import the Bulk Import all at once, rather than serialized. Simply place your Parallel Tool item on the workspace, then drag the Bulk Import into the Parallel box. See: **PARALLEL**
- **Save** – Allows the user to complete the import and save to local storage, in the publish deployment folder the InMemory Database Store File. See: **SAVE**
- **Script** – Allows the user to define a specific script to perform queries that suits their needs. See: **SCRIPT**
- **Slurp** – Allows the user to define a data source to consume the entire contents of the database, which is unlike Bulk Import and Import, this may be useful, if new tables are added to the database. See: **SLURP**
- **Union** – Allows the user to concatenate two tables if they are identical. See: **UNION**

Bulk Import

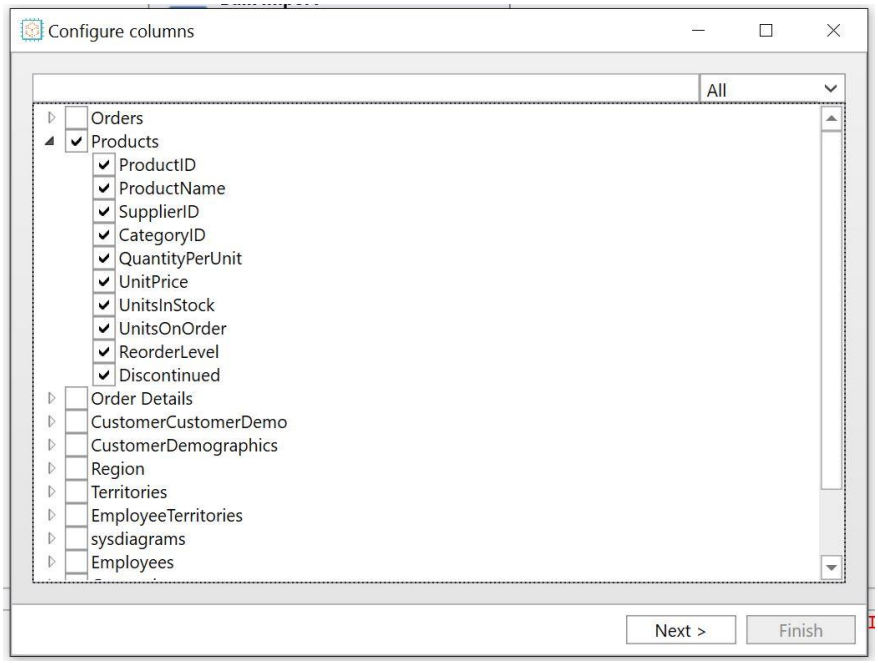
The TARGIT InMemory ETL workspace might look something like this:



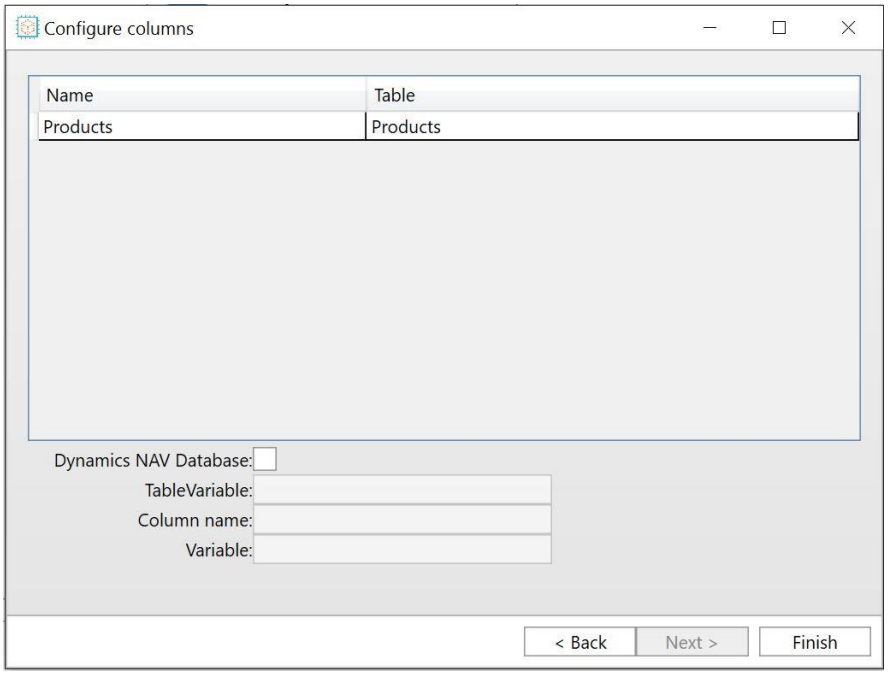
Data Source – the user defined data source from Data Sources
Tables – are the tables the user has selected from the Data Source, see below for more details.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**)

Configure Columns

Configure Columns – allows the user to pick specific columns and specific tables to be imported into the TARGIT InMemory Database.

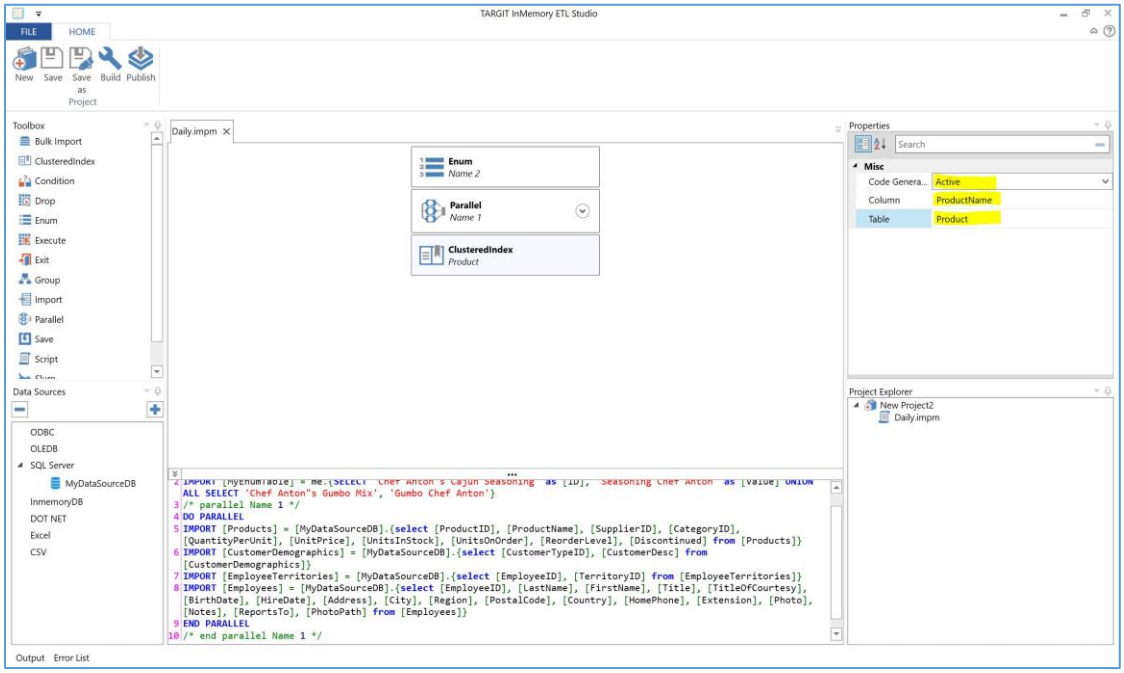


Clicking the Next button allows the user to provide any additional details to the Tables, including renaming the name of the table. Clicking Finish will complete all the configuration of the Tables.



Clustered Index

The TARGIT InMemory ETL workspace might look something like this:



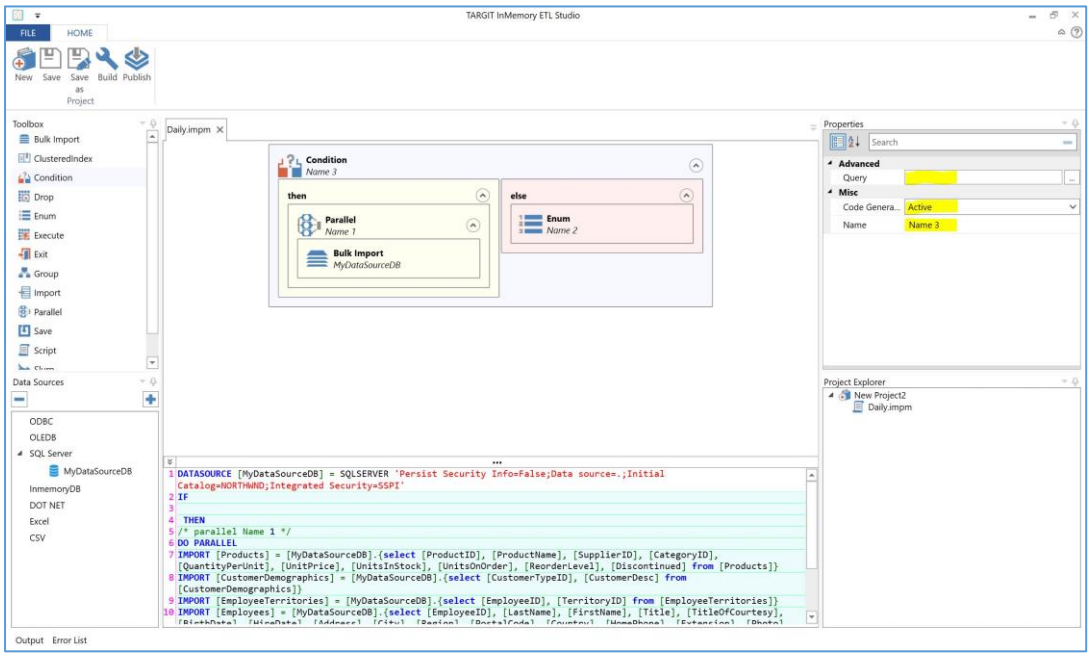
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**)

Column – user defined column

Table – user defined table

Condition

The TARGIT InMemory ETL workspace might look something like this:



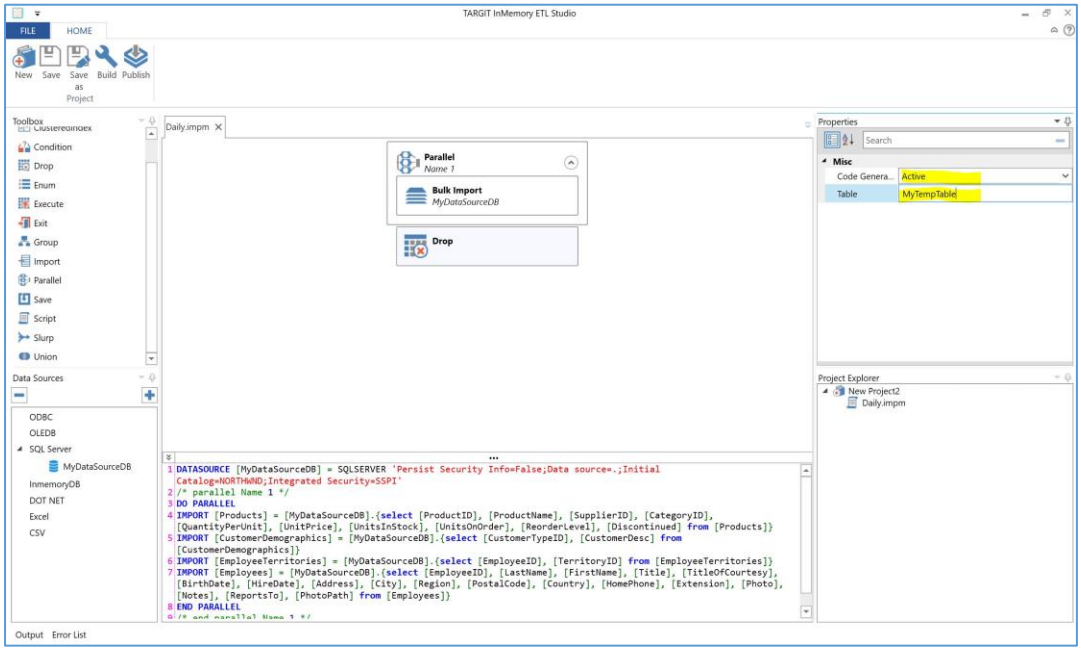
Query – user defined condition to test.

Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code**

Generation)
Name – allows the user to define a name for the Condition block

Drop

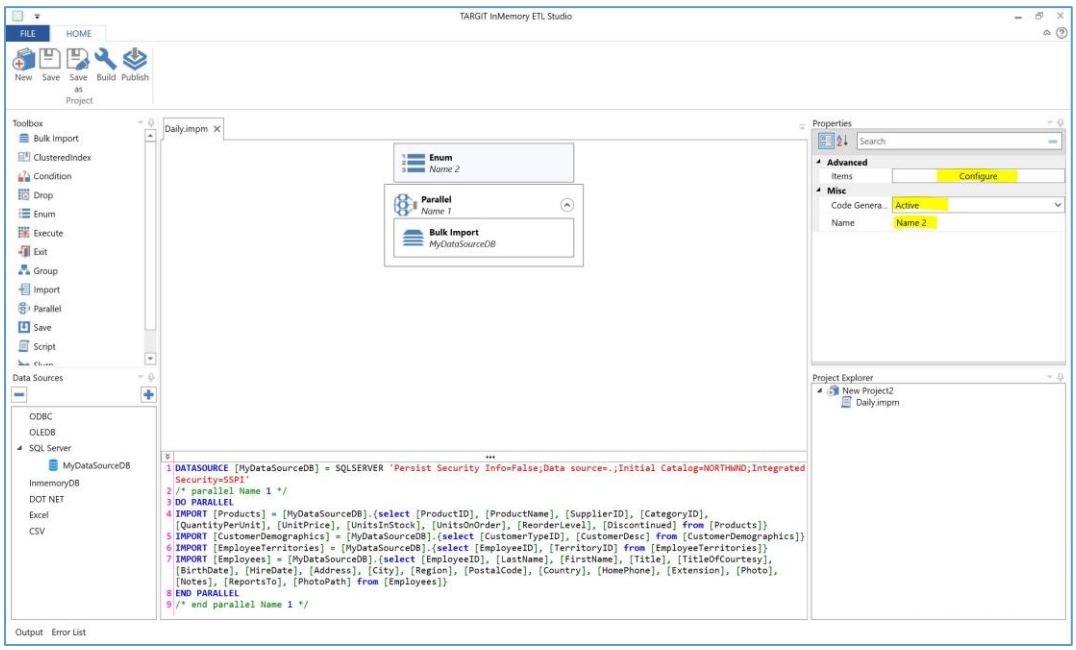
The TARGIT InMemory ETL workspace might look something like this:



Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**)
Table – allows the user to identify the Table that should be dropped.

Enum

The TARGIT InMemory ETL workspace might look something like this:



Items – allows the user to predefine the enumerations by clicking the Configuration button, see below.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).
Name – allows the user to name the Enum Block.

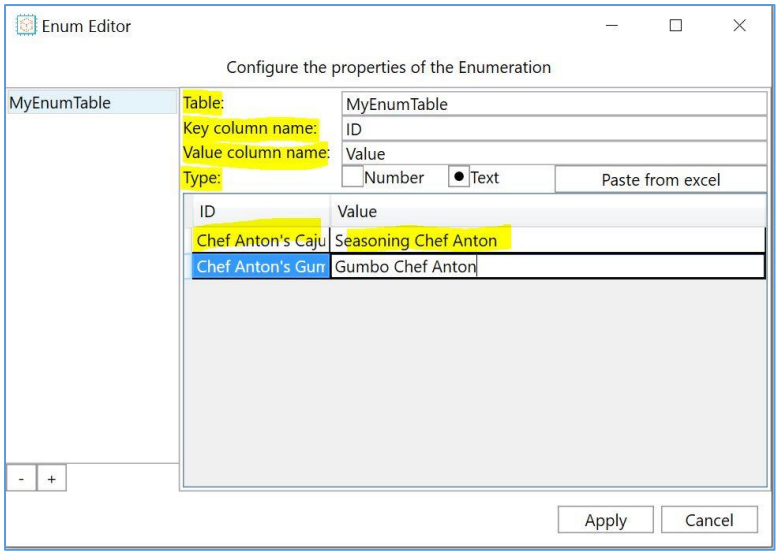
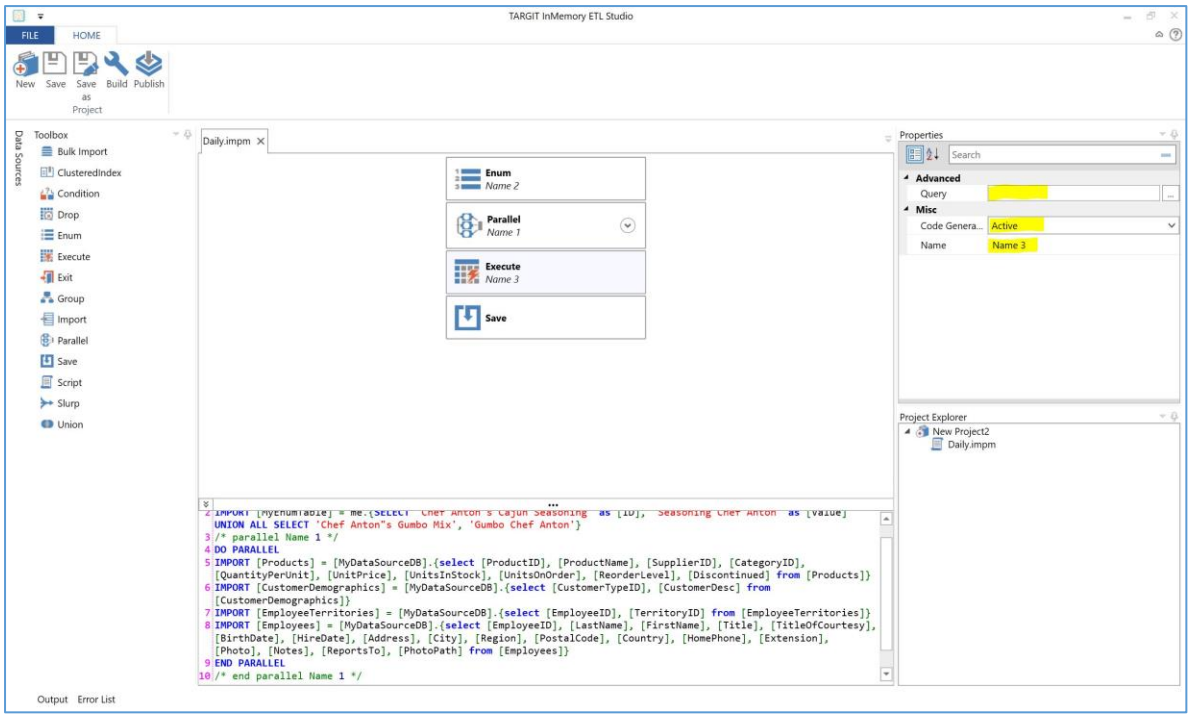


Table – allows the user to name the new Table
Key column name – allows the user to define the column name (Generally default is acceptable)
Value column name – allows the user to define the value name (Generally default is acceptable)
Type – allows the user to define what type of value is being defined as well as pasting values from excel

Execute

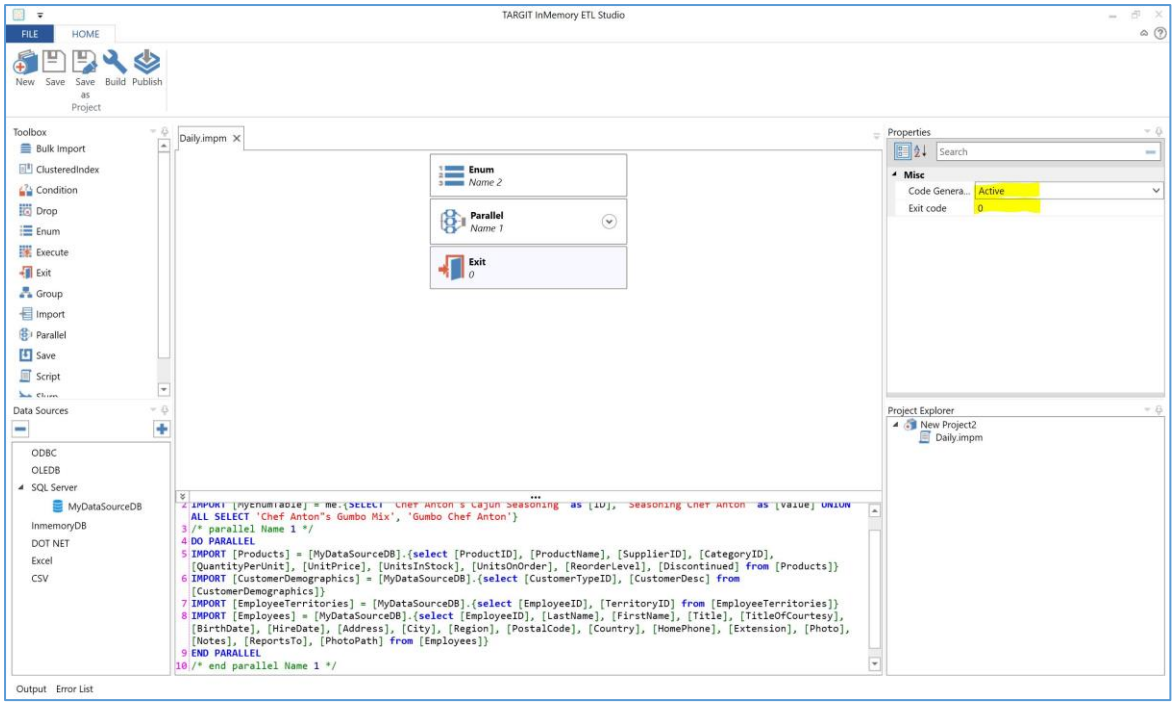
The TARGIT InMemory ETL workspace might look something like this:



Query – is the user defined script that will be executed.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).
Name – allows the user to define a name for the Execute block.

Exit

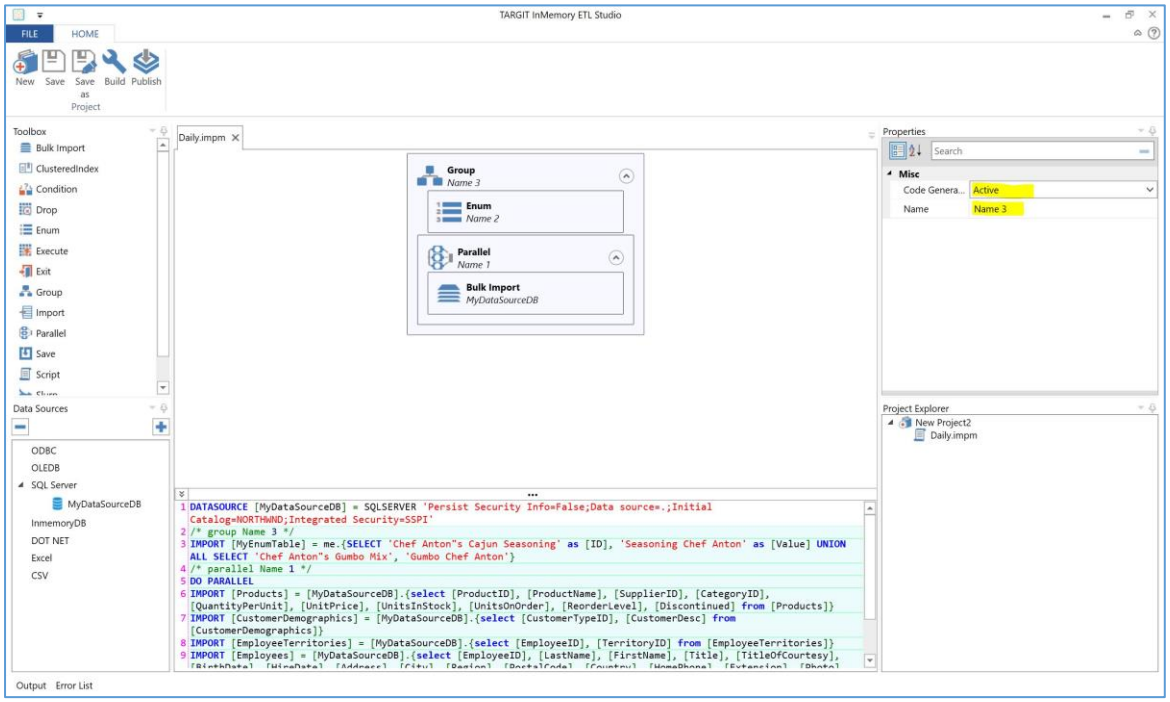
The TARGIT InMemory ETL workspace might look something like this:



Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).
Exit Code – allows the user to define a specific exit code or utilize the default.

Group

The TARGIT InMemory ETL workspace might look something like this:

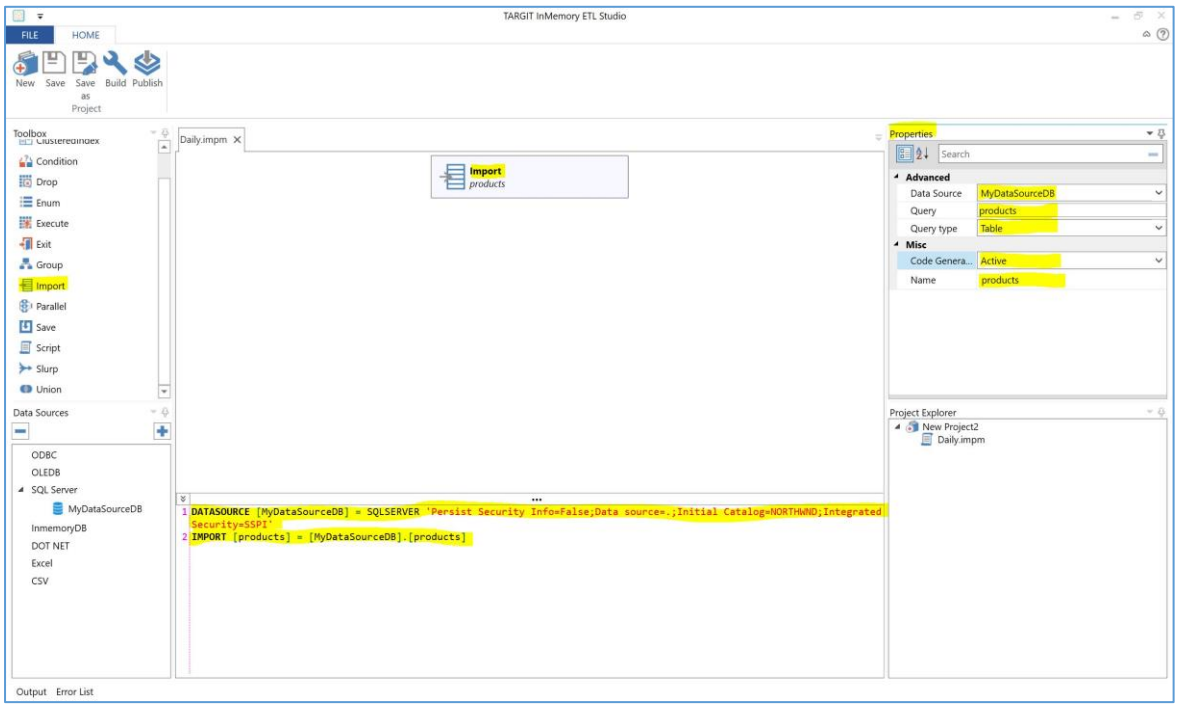


Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).

Name – allows the user to define a name for the Group block.

Import

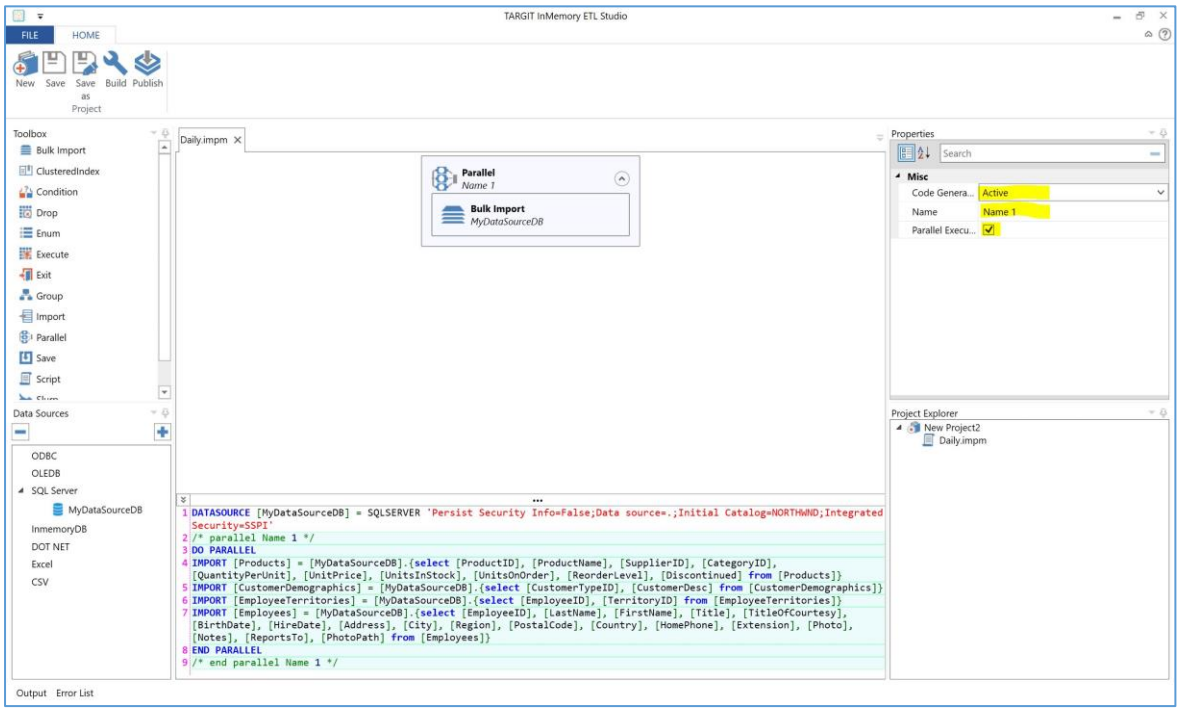
The TARGIT InMemory ETL workspace might look something like this:



Data Source – the user defined data source from Data Sources.
Query – is the table the user has selected from the Data Source.
Query Type – allows the user to define a query, which will exist in the Query box above.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).
Name – allows the user to define a name for the Import block.

Parallel

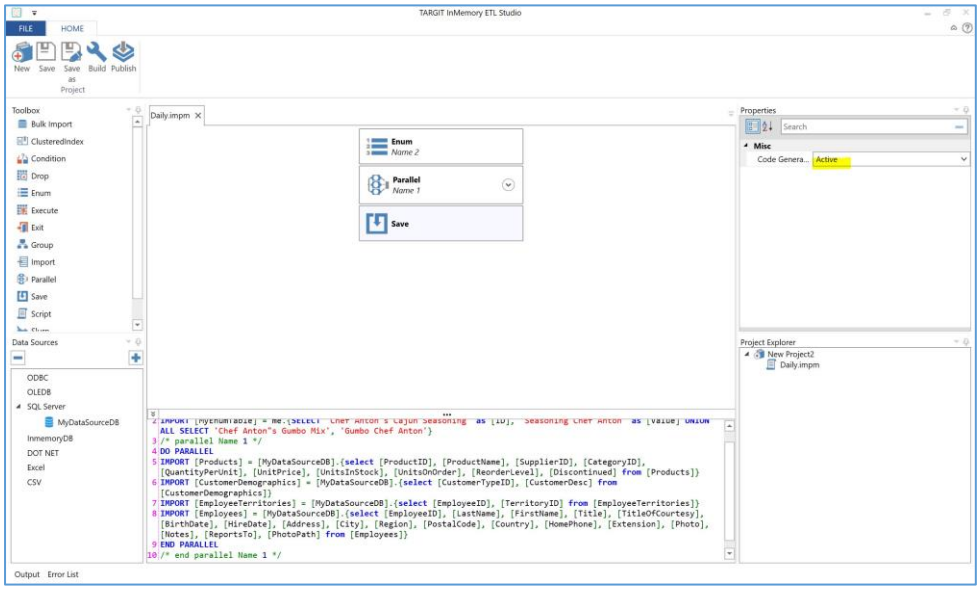
The TARGIT InMemory ETL workspace might look something like this:



Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).
Name – allows the user to define a name for the Parallel block.
Parallel Execution – allows the user to enable or disable parallel processing.

Save

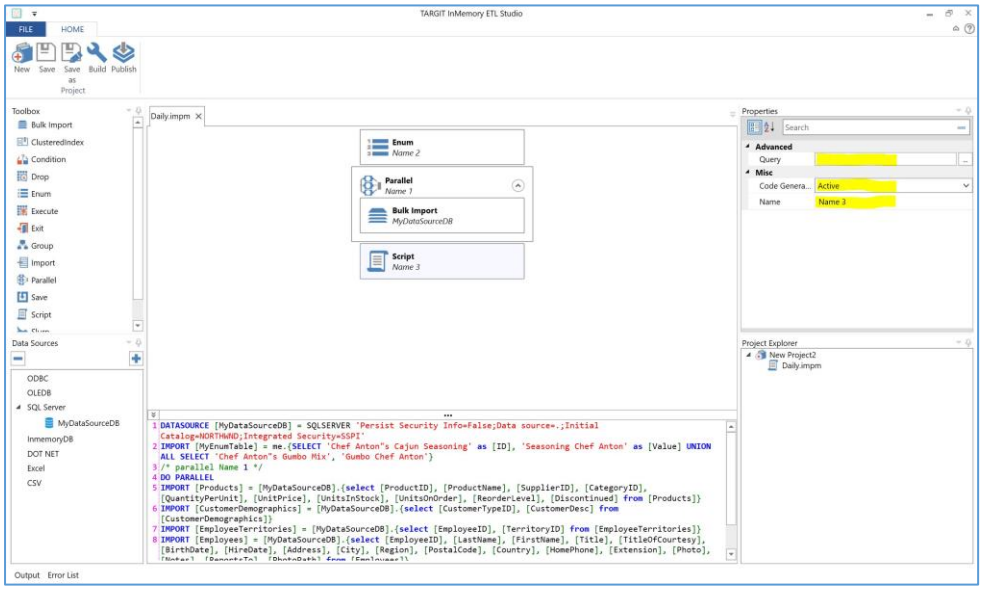
The TARGIT InMemory ETL workspace might look something like this:



Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).

Script

The TARGIT InMemory ETL workspace might look something like this:



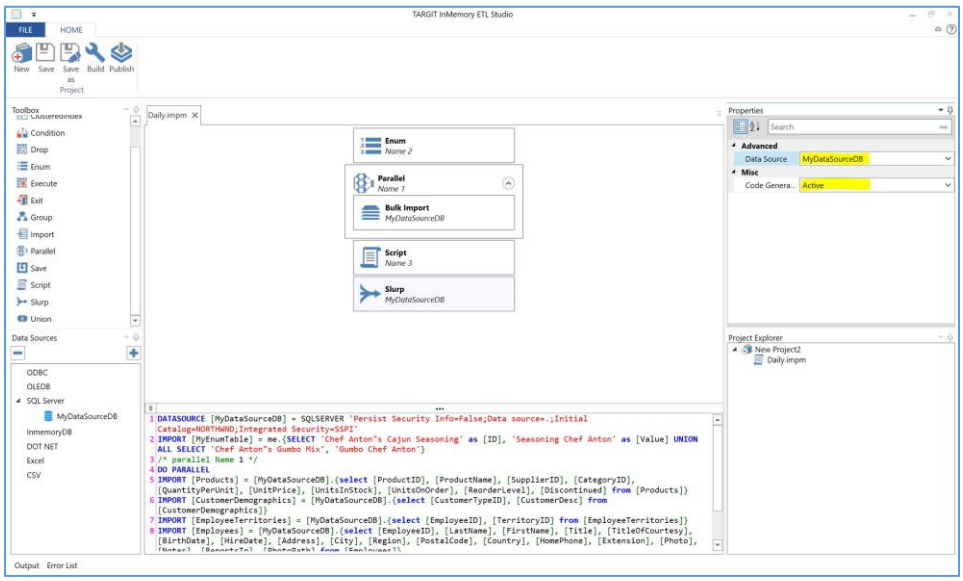
Query – the user defined script.

Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).

Name – allows the user to define a name for the Script block.

Slurp

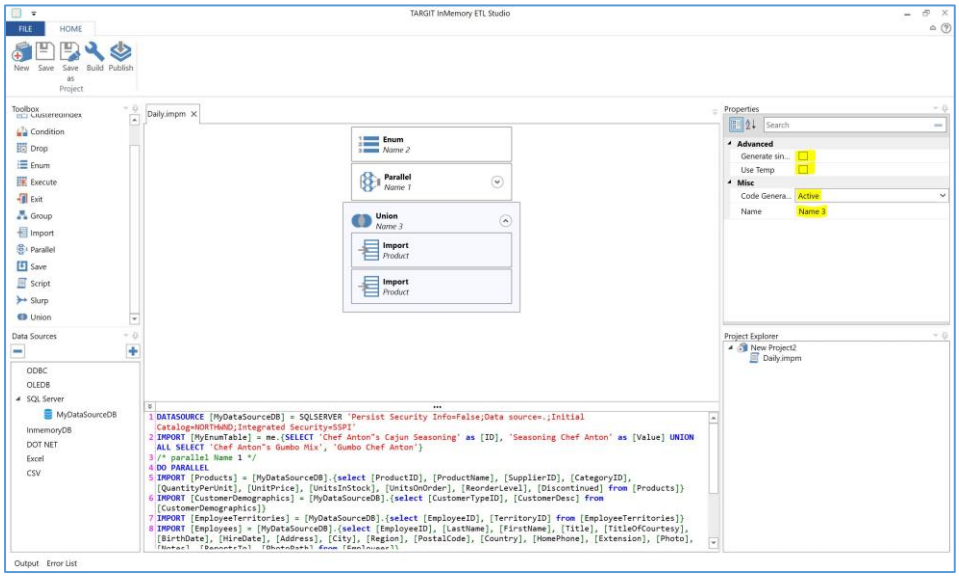
The TARGIT InMemory ETL workspace might look something like this:



Data Source – the user defined data source from Data Sources.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code Generation**).

Union

The TARGIT InMemory ETL workspace might look something like this:



Generate Single Command – allows the user to generate a single command line, and concatenate all at once. When unchecked the import is done for each table separately, then the concatenation happens.
Use Temp – allows the user to use a Temp table, which will utilize more TARGIT InMemory Database memory.
Code Generation – allows the user to make the code Active, Disable, or Disable and Hide: (See: **Code**

Generation).
Name – allows the user to define a name for the Union block.

Code Generation

Active.

If the user selects **Active** then the code will be allowed to Run, shown in the Code Generation Box on the Work Space as illustrated below:

```
1 DATASOURCE [MyDataSourceDB] = SQLSERVER 'Persist Security Info=False;Data source=.;Initial Catalog=NORTHWND;Integrated
  Security=SSPI'
2 IMPORT [products] = [MyDataSourceDB].[products]
```

Disable.

If the user selects **Disable** then the code will be Commented Out and will not Run, shown in the Code Generation Box on the Work Space (used for troubleshooting import issues) as illustrated below:

```
1 //IMPORT [products] = [MyDataSourceDB].[products]
```

Disable and Hide.

If the user selects **Disable and Hide** then the code will be Commented Out and will not Run, and not shown in the Code Generation Box on the Work Space (used for troubleshooting import issues) as illustrated below:

```
1
```

TARGIT InMemory Scheduler Management

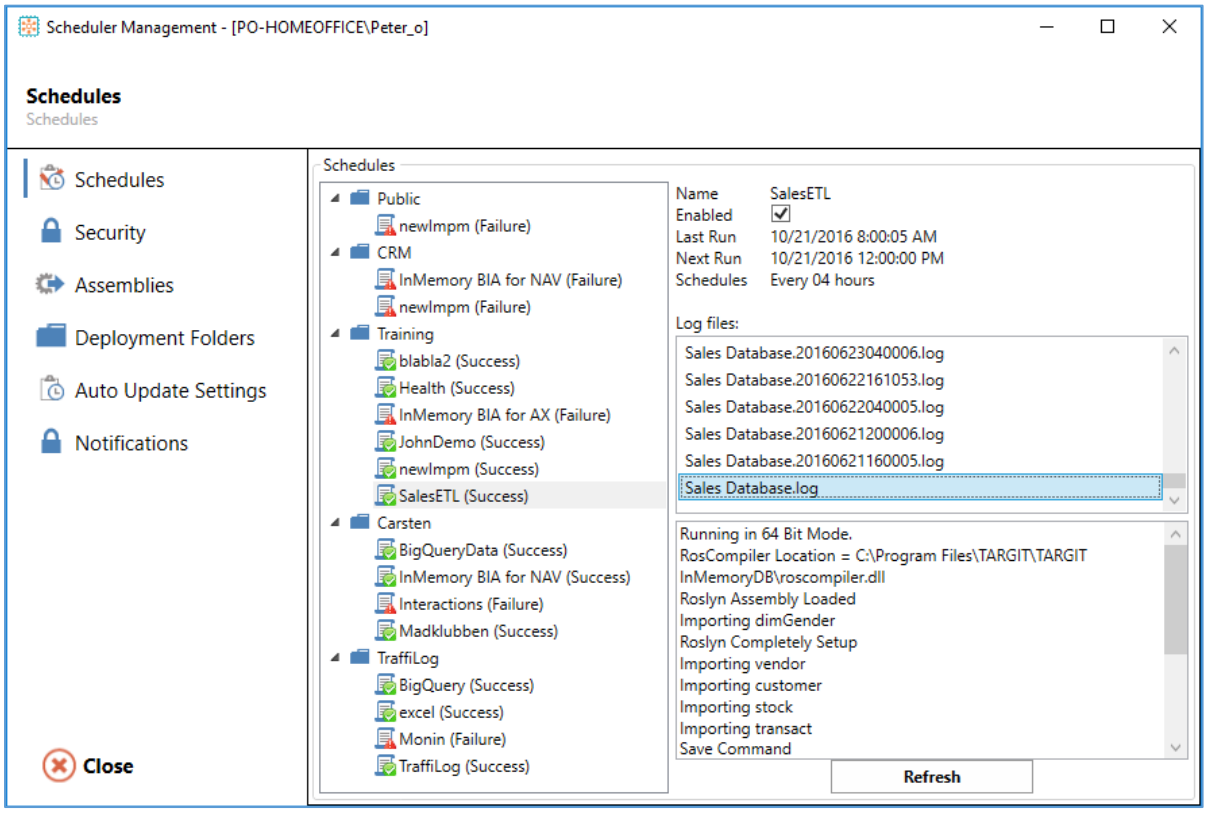
The TARGIT InMemory Scheduler is managed by this application. It enables you to:

- Control which projects are loaded into the scheduler

- View and delete logs
- Execute jobs manually
- Manage assemblies (e.g. additional data sources)
- Manage deployment folders
- Manage auto update settings
- Configure notifications to be sent out based on how ETL jobs complete.

Note: ALL security for the scheduler is configured separately from TARGIT Decision Suite and supports Windows Authentication exclusively.

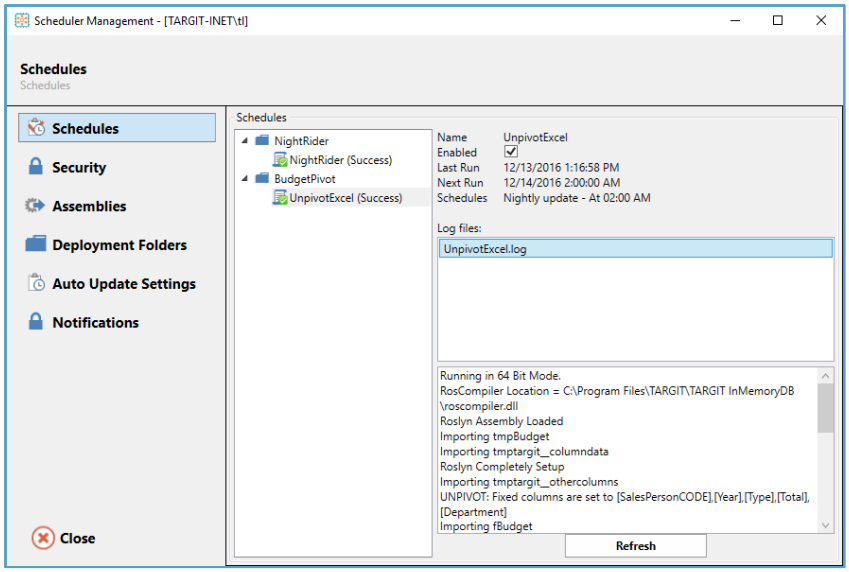
For more details about the TARGIT InMemory Scheduler Management dialog, please click the links to the left.



Schedule Administration

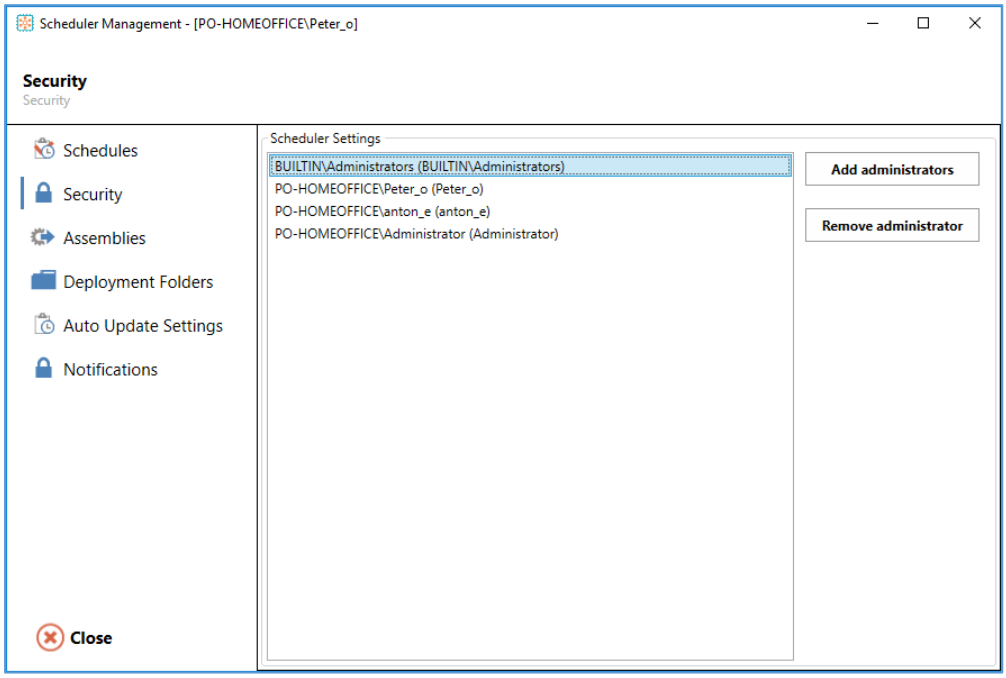
The scheduler windows left pane indicates what projects that are available to the scheduler and their execution status. You can enable and disable jobs by clicking a project and toggling the enabled setting in the upper left corner. Jobs can also be deleted by right-clicking the job.

On the right side is a list of log files. All failed executions are stored as their separate timestamped logfile and the current log will be available without a timestamp. By clicking the individual logfiles the content will be shown below. Each log can also be opened in the default text editor by clicking the file and right-clicking and choosing open.



Security

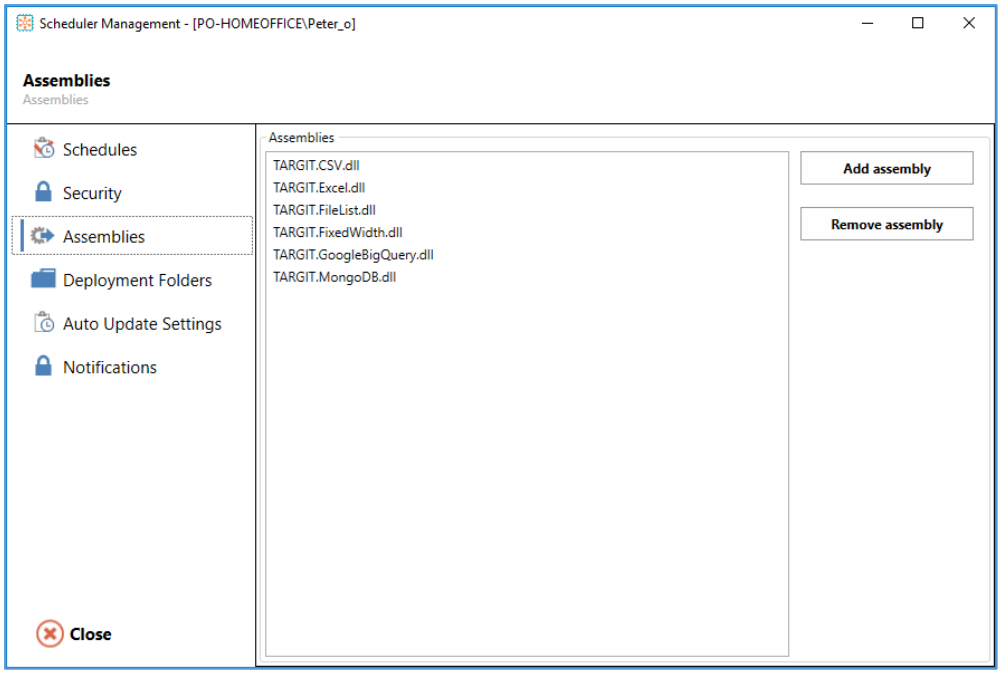
In the security section users who should have access to this application can be configured. Simply add/remove the users from the list.



Assemblies

The assemblies folder contains a list of the assemblies necessary to connect to other data sources. TARGIT supplies a range of Data Drivers (installed separately) for use with TARGIT InMemory Database.

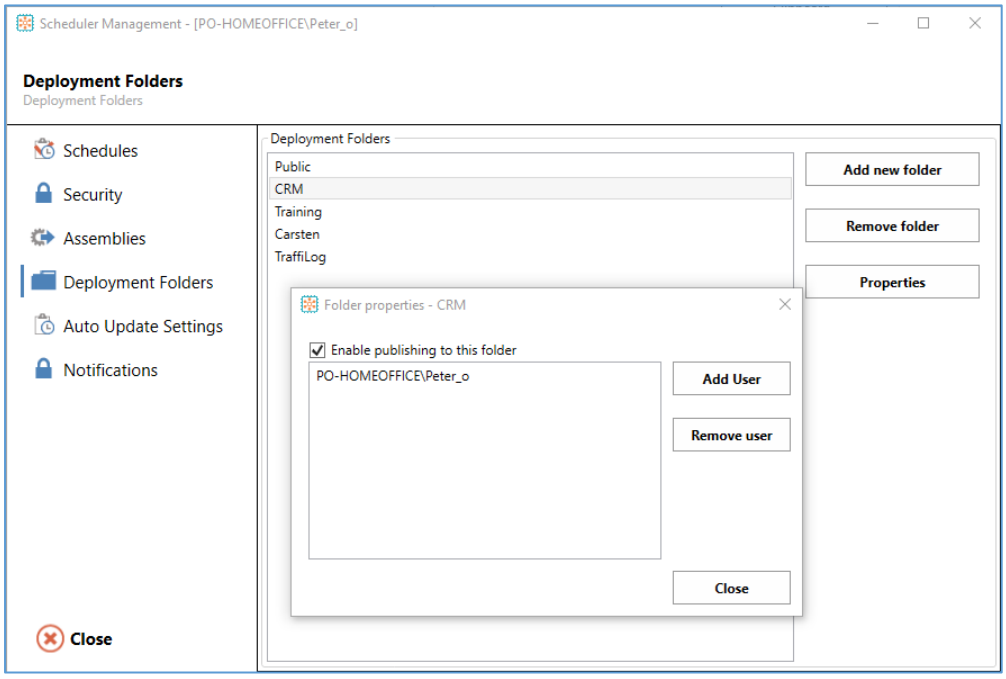
Examples of other 3rd party drivers would be the Oracle ADO.NET driver or other similar ADO.NET drivers.



Deployment Folder Administration

After having created an ETL project using TARGIT InMemory ETL Studio the project can be deployed to the Scheduler component. By default a folder called public is created, this folder will by default only be accessible for the user installing the TARGIT InMemory Scheduler, but can of course be managed with this application. Each folder will reside on the local filesystem as a subfolder to where the Scheduler service is installed. These folders can get permissions applied that will allow multiple developers to publish new versions of ETL project to the scheduler.

Note: Only individual users are allowed to be entered into the Properties Dialog, Windows User Groups are NOT supported.



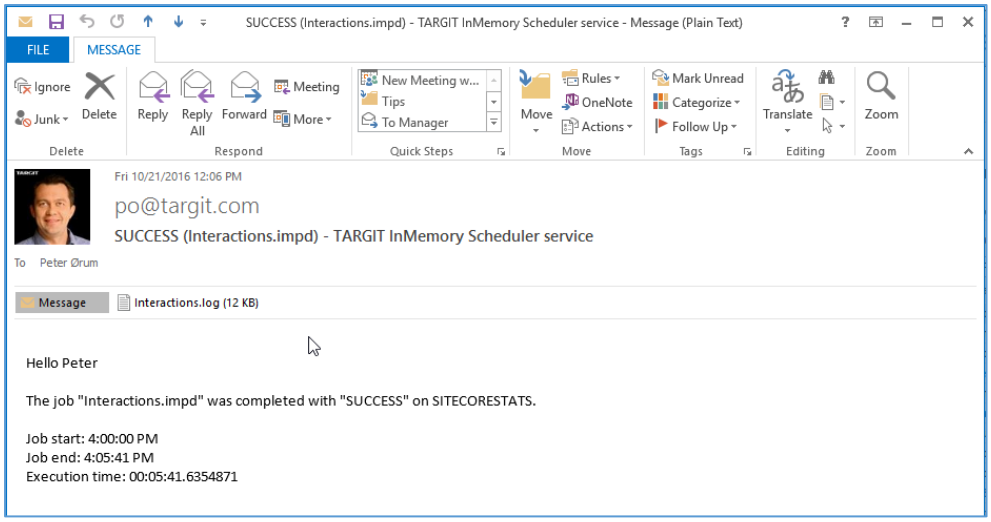
Auto Update Settings

In here the automatic update of the TARGIT InMemory Scheduler service can be configured. Automatic updates will not interrupt scheduled jobs but will be applied when the scheduler is idle.

Notifications

Under notifications messages regarding job status can be configured. The notifications can be configured per job and can be sent with or without the log attached to the specified email address.

The result of a job status will be like this:



SQL Language Reference

Whether running queries in tiQuery, in an tiImport script (against an TARGITDB source) or anywhere else, TARGIT InMemory Database supports the following SQL commands.

It is not the intent of this document to go into too much detail on the following, since TARGIT InMemory Database commands (where available) adopt common SQL syntax. Here we list the commonly-used SQL commands and conventions that are available in TARGIT InMemory Database and describe its syntax in a general sense –or in more detail if different from the norm.

RENAME

Renames a table.

Syntax:
RENAME from_table_name to_table_name

Example:

RENAME dCustomer Old_Customer

REORDER

The REORDER command allows you physically reorder a table by a different sort order. This can be useful it you want to go through a table record by record.

Syntax:

REORDER TABLE [TABLE NAME] BY column1 (optional desc),…, columnN (option desc)

Example:

REORDER TABLE orders by ORDERID Desc

SELECT

SELECT is the cornerstone of any SQL query. TARGIT InMemory Database supports the regular SQL select syntax.

Example:

```
SELECT [NOCACHE]
select_list
[ INTO new_table ]
[ FROM table_source  [AS alias1] ]
{[ INNER|LEFT JOIN table2 (as a2) on alias1.ColX = a2.ColY ]
} [ ,...n ]
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY {order_expression} [ ,...n ] ]
[ LIMIT n ]
```

The UNION ALL operator can be used between queries to combine or compare results into one result set.

The SELECT Clause supports selecting from fields in the From Clause, doing expression using + / - * and ( ) and other functions. It also supports SUM, MAX, MIN , AVG, COUNT,COUNT (DISTINCT some-field) aggregate functions. Aggregate functions can also be combined.

CASE Statements with multiple WHEN Clause are also supported.

Note1: If you do use an Alias for a field or table, the keyword “AS” is required.

Note2: SELECT INTO works in tiQuery, creating the named table in the InMemory database. However, since there is no way to SAVE from within tiQuery, once you close the tiQuery window the created table will be lost.

CACHE

By default, TARGIT InMemory Database Server creates temp tables when it feels sub-query-returns might be reused (e.g. for nested JOINs or where dynamic dates are used).

The creation of these temp-files can be switched off by the inclusion of nocache=true in the targitdb.ini file. Including the CACHE keyword in a select statement, overrides this statement by using cache. See also: Nocache.

CASE

Evaluates a list of conditions and returns one of multiple possible result expressions.

Example:

```
CASE WHEN logic_expression THEN SomeExpression
WHEN logic_expression2 THEN SomeExpression2
ELSE SomeOtherExpression END
```

COUNT

Count(field_name | * | 1) counts the number of non null data rows in the selection.

Example:

```
SELECT COUNT ( fieldname) FROM table_a
[WHERE field2 = 'test']
```

WITH

TARGIT InMemory supports the standard WITH statement. WITH Statements allow you to break more complicated SQL logic into more easily handled chunks. They are consumed by the next SELECT statement. They act like temporarily views that are used by the next SQL Select statement.

Example:

```
with MyCustomTable  as ( select * from orders )
with MyCustomTable2 as ( select * from customers )
select sum(1) from MyCustomTable
inner join MyCustomTable2
on MyCustomTable.customerid = MyCustomTable2.customerid
```

Recursive WITH statements are supported using the RECURSIVE keyword before the recursive SELECT.

Example:

```
WITH RecursiveExample as (select 5 as level
union all
recursive select level-1 as level
from RecursiveExample
where level > 0 )
select * from RecursiveExample
```

DELETE

Data rows can be deleted from a table using Delete placed inside embedded SQL brackets.

Example:

```
IMPORT orders = a1.orders
{DELETE FROM orders WHERE employeeid = 2}
```

Note: The Delete command does not work with data that has been created using the CREATE TABLE command if that table has not yet been Columnized.

OVER() / OVER (PARTITION BY)

OVER (PARTITION BY …. ORDER BY….) allows values to be partitioned based on the list of expressions following PARTITION BY and ordered by a list of expressions in the ORDER BY Clause.

Can be used with the RANK() and DENSE_RANK(),SUM,MIN,MAX,ROW_NUMBER() functions. ROWS UNBOUNDED PRECEDING at the end allows for CUMULATIVE Calculations.

Example:

```
SELECT EMPLOYEEID, CUSTOMERID,
   /* number of customers per employee */     SUM(1) AS NoOfCustsPerEmployee,
   /* Total number of Custs per employee  -note, same total in each row within Customerid */
    SUM ( SUM(1) )
                    OVER ( PARTITION BY EMPLOYEEID ORDER  BY CUSTOMERID )   AS NoOfAllCustsPerEmployee,
   /* Cumulative sum of Customerid within Employeeid */
    SUM ( SUM(1) )
                    OVER ( PARTITION BY employeeid ORDER  BY customerid ROWS UNBOUNDED PRECEDING )
    AS CumulativeSumOfCustsPerEmp
FROM ORDERS
GROUP BY EMPLOYEEID,CUSTOMERID
```

GROUP BY

GROUP BY supports grouping by multiple elements. Alias, Field Style or with Expressions that only use one dimension, e.g. DAY (Alias.SomeDateField).

Group by will run faster if the cardinality of the group dimensions multiplied together are less than 500,000. This will use a Parallelizable algorithm using arrays.

Cardinality greater than this will currently use a different parallel algorithm using a dictionary, which are slower. Cardinality of (9^10^18) ^ 16 is supported. This should be good for at least 32 dimensions, and many more in practice.

HAVING

HAVING is supported by TARGIT InMemory Database Server. The ALIAS from the select part of the query is used along with the keyword Having - or else an aggregate expression.

Example:

```
SELECT item, sum(qty) AS quantity_sum FROM sometable HAVING quantity_sum>100
OR
SELECT item, sum(qty) AS quantity_sum FROM sometable HAVING sum(qty)>100
```

WITH ROLLUP & GROUPING

WITH ROLLUP can be used after the GROUP BY clause in an aggregate query. It will take the GROUP BY elements one by one from the right and remove them and run the query, with that group by clause (nulling out the unused GROUP by columns), then append those results to the Result Set.

The GROUPING Function is also supported in the SELECT statement. It takes a column as a parameter. It returns 0 if the column is in the GROUP BY, and 1 if it is not.

Example:

Returns the Number Of Orders by Customer, Employee, then the number of orders by Customer, and then at the bottom, just the total number of orders. The last two columns will display 0,0 for the first part, 0,1 for the customer subtotals and 1,1 for the last row. This clause is useful to add higher level subtotals to an aggregate query.

```
select customerid, employeeid, sum(1) as NoOfOrders,

grouping(customerid),

grouping(employeeid)

from orders

group by customerId,Employeeid

with rollup
```

JOIN

1-1 or 1-Many inner/left joins can be supported. Your query will execute best if you have the table with the highest cardinality first, joining against the tables with decreasing granularity. You can join as many levels deep from the first table that is driving it.

TARGIT InMemory Database supports LEFT, INNER and CROSS JOIN ONLY.

LEFT JOIN

All joins are driven by the first table which is assumed to be the main fact table. It supports joining on up to eight dimensions. Queries should be designed without join loops and joined in such a way that the joins all flow from the first table or higher level tables.

Example:

```
SELECT *
FROM b
LEFT JOIN a ON b.id = a.id
```

CROSS JOIN

Cross joins can be created by having tables and joins separated by a comma.

```
SELECT *
FROM a,b,c
```

You may be able to speed up cross join queries by moving related parts of the where clause into B,C as sub-queries in this case.

Example

```
SELECT *
FROM a,b, (SELECT value1, value2 FROM c) AS cx
WHERE a.id = b.id AND b.value = cx.value1
```

INNER JOIN

An inner join is a join in which the values in the columns being joined are compared using a comparison operator.

```
SELECT *
FROM HumanResources.Employee AS e
        INNER JOIN Person.Person AS p
        ON e.BusinessEntityID = p.BusinessEntityID
ORDER BY p.LastName
```

RIGHT JOIN

All joins are driven by the first table which is assumed to be the main fact table. It supports joining on up to eight dimensions. Queries should be designed without join loops and joined in such a way that the joins all flow from the first table or higher level tables.

Example:

```
SELECT *
FROM a
RIGHT JOIN b ON b.id = a.id
```

PIVOT

Use the pivot operator as part of a join. The pivot operator allows you generate columns dynamically based on the list in the FOR subclause.

Example:
Sums freight by customer & year with a column for each employeeid in 1..9

```
select pivotTable.* from
( select customerid ,year(orderdate) as orderYear,freight,employeeid from orders ) as sourceTable
pivot ( sum(  freight) for employeeid in ([1],[2],[3],[4],[5],[6],[7],[8] ,[9])) as pivotTable
```

LIMIT

You can limit the number of results returned by the TARGIT InMemory Database Server by adding a limit clause to the end of a query. There are two means to achieve the same results.

Example:

SELECT * FROM sometable LIMIT 10 returns the first 10 records
SELECT * FROM sometable LIMIT 10,20 -ignores the first 10 records, returning the next 20.

NOCACHE

As a performance-booster the TARGIT InMemory Database Server creates temp tables when it feels sub-query-returns might be reused (e.g. for nested JOINs or where dynamic dates are used).

The creation of these temp-files can be switched off by the inclusion of the NOCACHE keyword. See also: CACHE.

ORDER BY

ORDER BY can use any amount of fields/ columns. Columns should be contained as expressions within the SELECT clause.

Table names, Column Names and Aliases are case insensitive and may contain an alphabetic character followed by alphanumeric -unless you put square brackets around the token, and then any character can be used. Names and aliases with a space also need to contained within square brackets.

Example:

```
SELECT OrderID, productid AS [1pID], suppID +' '+empID AS SuppEmp
FROM [ORDER DETAILS] AS details
ORDER BY productid, suppID +' '+empID
```

SELECT DISTINCT

SELECT DISTINCT is supported, but it is limited internally by the group by cardinality limits. To make a distinct selection from a query that uses GROUP BY, you will need to add a new outer select.

Example:

```
SELECT DISTINCT fieldX FROM ( SELECT fielda, fieldb, fieldX FROM table1 GROUP BY fieldb ) AS inner_query
```

SELECT*

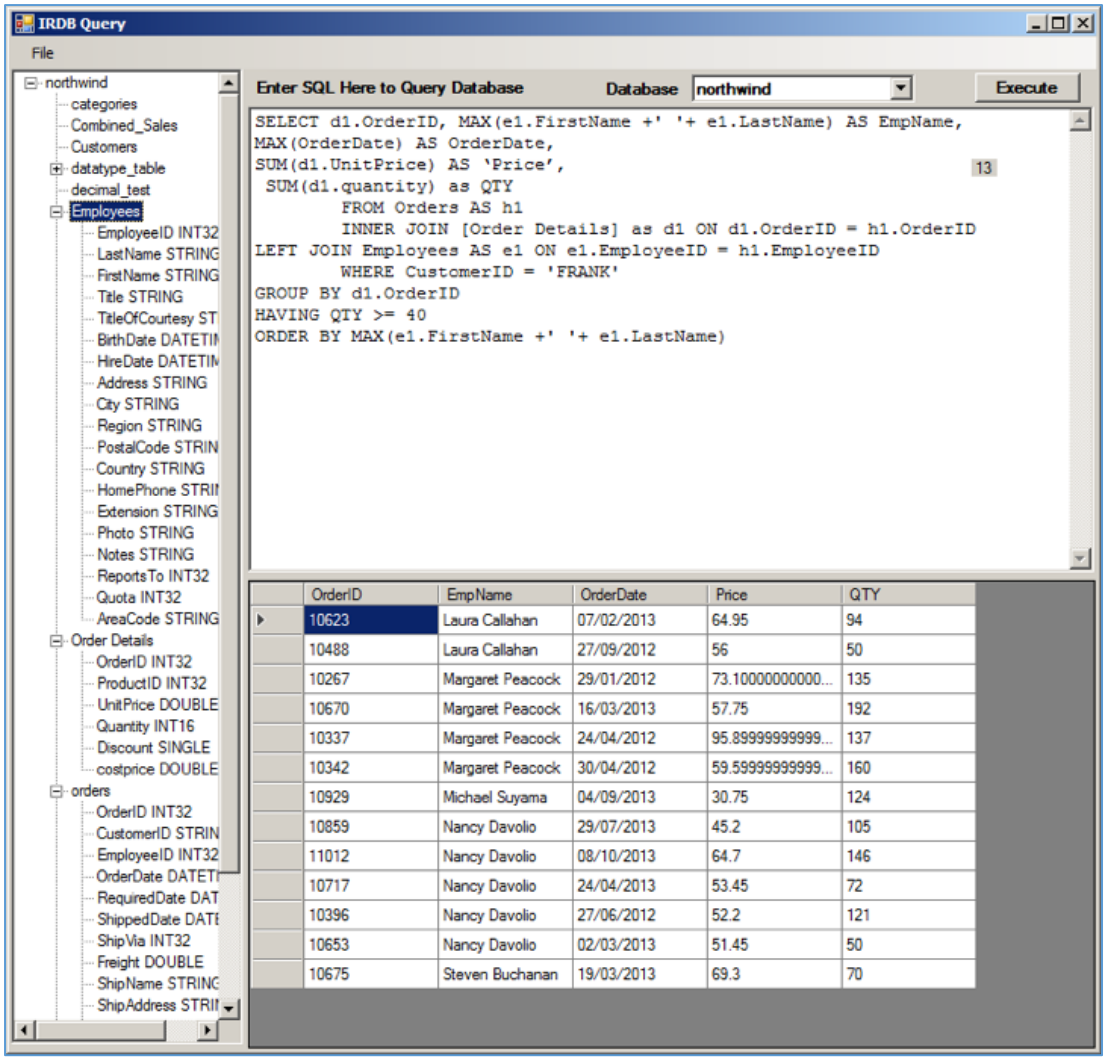
Select * or SELECT alias.* are also supported, but if you have columns with non supported characters, they will not be validated, and will cause the query to misbehave.

TARGIT InMemory Database is designed for doing aggregative queries quickly. Non Aggregative queries will run and will also continue to execute in full if you select a large number of records from them.

In our testing, tables with millions of records is not a problem, but if you pass a query that performs a SELECT * from a table with hundreds of millions records, internally it goes through the process of creating the table from scratch and columnizing it, thereby possibly running out of memory.

Note: Order By requires the relevant field/ expression as an argument rather than a column number.

```
SELECT *
FROM Orders
WHERE FREIGHT > 50SELECT d1.OrderID,
MAX(e1.FirstName +' '+ e1.LastName) AS EmpName,
MAX(OrderDate) AS OrderDate,
SUM(d1.UnitPrice) AS ‘Price’,
SUM(d1.quantity) as QTY
FROM Orders AS h1
INNER JOIN [Order Details] as d1 ON d1.OrderID = h1.OrderID
LEFT JOIN Employees AS e1 ON e1.EmployeeID = h1.EmployeeID WHERE CustomerID = 'FRANK'
GROUP BY d1.OrderID
HAVING QTY >= 40
ORDER BY MAX(e1.FirstName +' '+ e1.LastName)
LIMIT 100
```



Note: Order By requires the relevant field/ expression as an argument rather than a column number.

UNION ALL

You can combine multiple tables and queries together by using UNION ALL.

Note: UNION will parse, but it will be treated as UNION ALL –ie. duplicate records kept in the returned data set.

Example:

SELECT * FROM table1 UNION ALL SELECT * FROM table2.

INTERSECT

Returns the common rows between two SELECT statements.

Example:

TABLE1 has one column with the following data: a,b,c,d,e,f,g

TABLE2 contains: c,d,f,x,z,

SELECT * FROM TABLE1 INTERSECT SELECT * FROM TABLE2

Results in the following: c,d,f

EXCEPT

Returns rows that are in an initial SELECT statement, but not in a secondary query.

Example:

TABLE1 has one field with the following data: a,b,c,d,e,f,g

TABLE2 contains: c,d,f,x,z,

SELECT * FROM TABLE1 EXCEPT SELECT * FROM TABLE2

Results in the following: a,b,e,g

WHERE

WHERE clause supports = , > , < , >=, <= predicates and IN (value1,value2…,valueN) & LIKE. it also supports AND, OR and NOT, using Parentheses. Expressions can be used and also the DAY, Month and Year functions.

Strict String equality executes and returns results quickly if = or IN is used, but will run less quickly if other comparison predicates are used. String comparisons are not case sensitive. You can also use a subquery with one parameter with IN.

Null values may be checked in the following manner: expression IS NULL or expression = NULL. You can also use a NULL where a string would be expected in order to specify a null parameter.

You cannot use NULL with numeric or date expressions.

UPDATE

Updates columns in a columnized table.

Note: In import scripts the update command must be enclosed in brackets {}

Syntax:

```
UPDATE table_to_update from_alias [LEFT JOIN|INNER JOIN table_to_update_with join_alias]
   set amount=amount*100,
  somecolumn=join_alias.somecolumn_in_alias
WHERE from_alias.somecolumn is null
{UPDATE Customers SET CustName='Schmitt', City='Hjorring' WHERE CustNumber=2134}
{
UPDATE fInvoiceLines a left join dInvoiceHeaders b on a.Company=b.Company and a.[DocNo]=b.[DocNo]
 set wPricesIncludingVAT = b.[Prices Including VAT]
 where b.[Prices Including VAT]=1}
}
```

Function Overview

Below is a breakdown of the functions available in TARGIT InMemory Database SQL dialect.

String Functions

String Functions - CAST | CAST_STR_AS_INT | CAST_STR_AS_DECIMAL | CHAR | CHARINDEX | CHARINDEXSTD | COALESCE | CONCAT | CSTR | ENDSWITH | INSERT | ISNULL | ISNULLOREMPTY | LEFT | LEN | LCASE | LTRIM | REMOVE | REPLACE | REVERSE | RIGHT | RTRIM | SUBSTRING | STARTSWITH | TRIM | UCASE

Date Functions - CDATE | DATE | GETDATE | DATEADD | DATEDIFF | DATEDIFFMILLISECOND | DATEPART | DATESERIAL | DAY | DAYOFWEEK | MONTH | TRUNC | YEAR

Math - ABS | CAST_NUM_AS_BYTE | CAST_NUM_AS_DECIMAL | CAST_NUM_AS_DOUBLE | CAST_NUM_AS_INT | CAST_NUM_AS_LONG | CAST_NUM_AS_SHORT | CAST_NUM_AS_SINGLE | FLOOR | LOG | MAX | MAXLIST | MIN | MINLIST | POWER | RAND | ROUND | SIGN | SQRT

Trigonometric - ASIN | ACOS | ATAN | ATAN2 | COS | COSH | SIN | SINH | TAN | TANH

Aggregate Functions - MIN | MAX | COUNT | AVG | SUM | COUNT (DISTINCT()) | MINLIST | MAXLIST

Statistical Functions - Statistical Functions

STDEV| STDEVP | VAR | VARP

Functions that are available in tiImport Expressions

These functions are available in tiImport expressions only. They are a subset of the TARGIT InMemory Database SQL functions.

Functions with 1 parameters - DAY | DAYOFWEEK | MONTH | YEAR | ABS | FLOOR | INT | LEN | UCASE | LCASE | TRIM | LTRIM | RTRIM | SIGN | WEEKDAY

Functions with 2 parameters - LEFT | RIGHT | CONCAT | ENDSWITH | FORMAT

Functions with 3 parameters - DATESERIAL| DATEADD| DATEDIFF | IIF | REPLACE

ABS (number)

returns *number* if positive -or the positive part of *number* if negative

ACOS (number)

returns the ACOS of *number*

ASCII (char)

returns the ASCII code for a given *character* – or the first character in case of a *character* array (*string*)

ASIN (number)

returns the principal value of the arc sine of *number*

ATAN (number)

returns the ATAN of *number*

ATAN2 (number1, number2)

returns the ATAN2 of *number1*, *number2*

AVG (expression)

Returns the average of the values in a group. Null values are ignored.

Syntax:

AVG (*expression*)

Example:

```
SELECT AVG(VacationTime)AS 'Average vacation time in hours',
                SUM(SickLeave) AS 'Total sick leave time in hours'
FROM HR.Employee
WHERE JobTitle LIKE 'Vice Admiral%'
```

CAST (expression)

Converts *expression* from its original format into the chosen format.

Syntax:

CAST(*expression* as DataType)

Data Types:

- BIT
- CHAR
- VARCHAR
- NVARCHAR
- NCHAR
- TEXT
- NTEXT
- DATE
- DATETIME
- DECIMAL
- DOUBLE
- FLOAT
- INTEGER
- VARCHAR
- REAL
- SMALLINT
- TINYINT

See also: SAFECAST.

Example:

CAST(*Expression* AS TEXT)
CAST(*Expression* AS DATETIME)
CAST(*Expression* AS BIT)
CAST(*Expression* AS TINYINT)
CAST(*Expression* AS SMALLINT)
CAST(*Expression* AS INTEGER)
CAST(*Expression* AS BIGINT)
CAST(*Expression* AS REAL)
CAST(*Expression* AS DOUBLE)
CAST(*Expression* AS DECIMAL).

CAST_NUM_AS_BYTE (number)

Returns a byte corresponding to *number.* See also: CAST, SAFECAST.

CAST_NUM_AS_DECIMAL (number)

Returns a decimal corresponding to *number*. See also: CAST, SAFECAST.

CAST_NUM_AS_DOUBLE (number)

Returns a double corresponding to *number*. See also: CAST, SAFECAST.

CAST_NUM_AS_INT (number)

Returns an int corresponding to *number.* See also: CAST, SAFECAST.

CAST_STR_AS_INT (string)

Returns an int corresponding to *string.* See also: CAST, SAFECAST.

CAST_NUM_AS_LONG (number)

Returns a long corresponding to *number*. See also: CAST, SAFECAST.

CAST_NUM_AS_SHORT (number)

Returns a short corresponding to *number*. See also: CAST, SAFECAST.

CAST_NUM_AS_SINGLE (number)

Returns a single corresponding to *number*. See also: CAST, SAFECAST.

CAST_STR_AS_DECIMAL (string)

Returns a decimal corresponding to *string*. See also: CAST, SAFECAST.

CAST_STR_AS_DOUBLE (string)

Returns a double corresponding to *string*. See also: CAST, SAFECAST.

CAST_STR_AS_LONG (string)

Returns a long corresponding to *string*. See also: CAST, SAFECAST.

CDATE (string)

Converts *string* as a date.

CHAR (number)

Returns a char from an ASCII code *number.*

CHARINDEX

Returns the position of the specified character or sequence. Zero (0) is returned if the string given as search parameter does not exist. The optional *start_position* parameter specifies the position in the string to start searching.

Syntax:

CHARINDEX (string_to_search, search_for [,start_position])

Example:

```
/* returns 3 as it is the first occurrence of the character 'c' in the string */
SELECT CHARINDEX ('abcda1234', 'c')

/* returns 5 as the first occurrence of 'a' on or after position 2 is on position 5 in the string */
SELECT CHARINDEX ('abcda1234','a',2)

/* returns 0 as the character does exist in the string */
SELECT CHARINDEX ('abcda1234','x')
```

CHARINDEXSTD

CHARINDEXSTD has a different parameter order than CHARINDEX to align to other SQL syntaxes.

Returns the position of the specified character or sequence. Zero (0) is returned if the string given as search parameter does not exist. The optional *start_position* parameter specifies the position in the string to start searching.

Syntax:

CHARINDEXSTD (search_for, string_to_search [,start_position])

Example:

```
/* returns 3 as it is the first occurrence of the character 'c' in the string */
SELECT CHARINDEXSTD ('c','abcda1234')

/* returns 5 as the first occurrence of 'a' on or after position 2 is on position 5 in the string */
SELECT CHARINDEXSTD ('a','abcda1234',2)

/* returns 0 as the character does exist in the string */
SELECT CHARINDEXSTD ('x','abcda1234')
```

CLOSEBAL (measure, [DateColumnName])

This functions provides native support for doing balances, e.g. for onhand inventory or general ledger balances. CloseBal returns the closing balance – which is the sum of the measure since beginning of time including “todays” transactions.

Syntax:

SUM(CLOSEBAL(*measure*,*[DateColumnName]))*

Note: In case there are no transactions for a specific date the date will be excluded from the output. See also: OPENBAL.

Example:

SUM(CLOSEBAL(*measure*,*[DateColumnName]))*

CLOSEBALWITHZEROES (measure, [DateColumnName])

This functions provides native support for doing balances, e.g. for onhand inventory or general ledger balances. CloseBal returns the closing balance – which is the sum of the measure since beginning of time including “todays” transactions. Compared to CLOSEBAL, CLOSEBALWITHZEROES will add additional rows to the rowset if there are missing entries derived from the date being used.

Syntax:

SUM(CLOSEBALWITHZEROES(*measure*,*[DateColumnName]))*

TIMESHIFT

TIMESHIFT operates within a SUM expression to make a SUM where the date dimension specified by columnToTransform is transformed to transformationToMake and the resulting SUM made. Useful for comparative results, e.g. TY vs LY.

Syntax:

SUM (TIMESHIFT (aggExp , columnToTransform, transformationToMake))

Example:

Produces a sum of This Year vs Last Year for 2015 on Orders

```
select  year( orderdate),
sum(1) as total_TY ,
sum( timeshift ( 1, orderdate, dateadd ( y , 1 , orderdate) )  )  as Total_LY
from orders
where orderdate>='2015-01-01'  and orderdate<='2015-12-31'
group by year(orderdate)
```

LASTCHILD

Takes the parent SQL and adds the childDim to the grouping Clause, then returns the last entry ordered by childDim as an aggregation on aggExp. The resultset will not be expanded by the grouping level. It will only show the last entry from this additional query.

Ideal for working with snapshot data, e.g. inventory.

Syntax:
SUM (LastChild (aggExp, childDIm))

Example:

```
select customerid,                year(orderdate) as theyear,
month(orderdate) as theMonth,
    sum(lastchild(freight, orderdate ))  as [lastchild]
from orders
group by customerid,year(orderdate),month(orderdate)
order by customerid, year(orderdate),month(orderdate)
```

LASTPRICE

Takes the parent SQL and adds the childDim to the grouping clause, then returns the last entry ordered by childDim and does an aggregation on aggExp. Can handle cases where the grouping clause involves expressions of ChildDim or child tables expression derived from it. ChildDim needs to be a database field. It will populate an entry in the resultset once it comes across an entry in the child data, and will add subsequent rows derived from ChildDim, propagating the LastPrice entry, even if there is no underlying data within the group.

Only values of ChildDim that satisfy the where clause are considered. It uses the distinct values of childDim, that satisfy the where clause, for the pivot basis. If an entry of childDim doesn't appear at least once in the resultset, it will not add additional entries for that value.

Ideal for working with snapshot data, e.g. inventory.

Syntax:

SUM (LastPrice (aggExp, childDIm))

Example:

```
select customerid, year(orderdate) as theyear,

month(orderdate) as theMonth,

sum(lastprice(freight, orderdate ))  as [LastPrice]

from orders

group by customerid,year(orderdate),month(orderdate)

order by customerid, year(orderdate),month(orderdate)
```

LASTPRICEWITHZEROES

This is similar to the above functions, but it will add additional entries to the output, so that all grouping rows, will be pivoted against all values of childDim, and its joined dimension table, that a where clause with the childDim part only. It will also examine data, from before the minimal value of childDim, that satisfies the where clause. In the example below it would also add entries for months 1 to 11 to the output, because there is prior data in 2014, and there are orders in 2015 for months 1 to 11 for other customers. This adds entry for childDim for values that are in the underlying data, but don't satisfy the where clause.

Ideal for working with snapshot data, e.g. inventory.

Syntax:

SUM (LastPriceWithZeroes (aggExp, childDIm))

Example:

```
select customerid, year(orderdate) as theyear,

month(orderdate) as theMonth,

sum(lastpricewithzeroes(freight, orderdate ))  as [LastPriceWithZeroes]

from orders

where customerid='DRACD' and year(orderdate) = 2015

group by customerid,year(orderdate),month(orderdate)

order by customerid, year(orderdate),month(orderdate)
```

COALESCE (VAL1, VAL2… VALN)

Returns the first non null *value* in the list otherwise *valn*.

COLUMNEXISTS

Returns VALUE_IF_EXISTS, if COLUMN_NAME is present in the FROM Clause of the current query, otherwise return CALUE_IF_NO_EXISTS.

Example:

COLUMNEXISTS (COLUMN_NAME,VALUE_IF_EXISTS,VALUE_IF_NO_EXISTS)

CONCAT (exp1, exp2… expN)

Returns a string containing the string result of concatenating *exp1* through *expN*. Parameters must be expressions or fields representing a string or number. A NULL value will be treated as an empty string. You can also do string concatenation with +.

COS (number)

Returns the cosine of *number.*

COSH (number)

Returns the hyperbolic Cos of *number.*

COUNT

Returns the count of all values respecting the parameters given in the expression.

Syntax:

SELECT COUNT([DISTINCT] column [WHERE filtercolumn=value_to_test OR filtercolumn=another_value_to_test]) FROM table

Examples:

/* returns the number of distinct id's from the table transactions */
SELECT COUNT(DISTINCT id) FROM transactions

/* returns the number of distinct ids in column one and the number of distinct ids where the column chain=2 please note that this expression can be used in the datamodeler by simply typing the WHERE clause after the columnname. */

SELECT COUNT(DISTINCT id) AS distinctids, COUNT(DISTINCT id WHERE chain=2) AS distinctidchain2 FROM transactions.

CSTR (expression)

Converts *expression* to a string.

DATE ( )

Returns the current date without the time part based on the current date of the TARGIT InMemory Database Server. See also: GETDATE, TRUNC, FORMAT, MONTH, DAY, YEAR .

Example:

SELECT DATE()

DATEADD (intervalstring, integer, date)

This adds *integer* to *date* corresponding to the interval *intervalString*.

Valid intervals are d,dd,w,ww,m,mm,q,y,yy,yyyy,h,hh,n,s
Example: dateadd ('m',1,'2010-01-01')

See also: DATEDIFF, DATEFIFFMILLISECOND, DATEPART, DATESERIAL.

DATEDIFF (intervalstring, date1, date2)

Returns an integer corresponding to the difference according to *intervalString* between *date1* and *date2*.

Valid intervals are d,dd,w,ww,m,mm,q,y,yy,yyyy,h,hh,n,s

See also: DATEADD, DATEFIFFMILLISECOND, DATEPART, DATESERIAL.

DATEDIFFMILLISECONDS (end_date, start_date)

DATEDIFF returning elapsed milliseconds.

Syntax:

DATEDIFFMILLISECONDS (*end_date*, *start_date*)

See also: DATEADD, DATEDIFF, DATEPART, DATESERIAL.

DATEDIFFTICK (end_date, start_date)

DATEDIFF returning elapsed ticks.

Syntax:

DATEDIFFTICK (*end_date*, *start_date*)

See also: DATEADD, DATEDIFF, DATEPART, DATESERIAL.

DATEPART (datepart, date)

Returns a value corresponding to the *datepart* requested from the given *date*.

Example: DATEPART (h, getdate()) returns the current hour of the current *datetime*.

Valid *dateparts* are d,dd,m,mm,q,y,yy,yyyy,h,hh,n,s

See also: DATEADD, DATEDIFF, DATEFIFFMILLISECOND, DATESERIAL.

DATEPARSE

Returns a date corresponding to STRING1 parsed using the format specified in FORMAT_INTEGER. Currently, a value of 101 is supported for FORMAT_INTEGER for MM/DD/YYYY formats. This is useful if you want to parse US Style dates anywhere in the world.

Syntax:

DATEPARSE (STRING1, FORMAT_INTEGER)

DATESERIAL (yearint, monthint, dayint)

Returns a date corresponding to the 3 *year*, *month* and *day* parameters.

See also: DATEADD, DATEDIFF, DATEFIFFMILLISECOND, DATEPART.

DAY (date)

Returns the Day in the month of the *date* parameter. See also: DAYOFWEEK, FORMAT.

DAYOFWEEK (date)

Returns an integer of the day of week. 1..7. See also: DAY, FORMAT.

Example:

```
/* Retrieve list of weekdays with weekday name. */
/* Be aware that the day name may be differ based on culture of the machine you're running this script on.
*/
/* E.g. Sunday may be day #1 */
SELECT DISTINCT thedate,DAYOFWEEK(thedate) WeekDayNo,
CASE DAYOFWEEK(thedate)
WHEN 1 THEN 'Monday'
WHEN 2 THEN 'Tuesday'
WHEN 3 THEN 'Wednesday'
WHEN 4 THEN 'Thursday'
WHEN 5 THEN 'Friday'
WHEN 6 THEN 'Saturday'
WHEN 7 THEN 'Sunday'
ELSE
'none'
END AS WeekDayName
FROM transact
```

ENDSWITH (string, substring)

Returns True if a *string* ends with *substring*.

FLOOR (number)

Returns the largest integer less than or equal to the specified numeric expression.

CEILING (number)

Returns the smallest integer greater than or equal to the specified numeric expression.

FORMAT (date|number, format_string)

This returns a *string* from a *date* or *number* using the *format_string* syntax. In the case of a date, the *format_string* should be structured to indicate the year, month, day, etc. with their initial.

Working with time:

- To remove the time from a date time use the TRUNC function.
- To retrieve the time from a date use format(GETDATE(),'HH:mn:ss')

See also: MONTH, DAYOFWEEK.

Example:

```
/* might return something like 29-2015-12 */
SELECT FORMAT (GETDATE(), ‘dd-yyyy-mm’)
/* returns 5,459,40 */
SELECT FORMAT (5459.4, '##,##0.00')

/* Returns name of month in full e.g. 'October' */
SELECT DISTINCT thedate,FORMAT(thedate,'MMMMM') AS MonthName FROM transact
TheDate      Month
10/1/2000    October
10/2/2000    October
10/3/2000    October
10/4/2000    October

/* Returns name of month in abbreviated e.g. Oct */
SELECT DISTINCT thedate,FORMAT(thedate,'MMM') AS MonthName FROM transact
TheDate      Month
10/1/2000    Oct
10/2/2000    Oct
10/3/2000    Oct
10/4/2000    Oct
```

GETDATE ( )

Retrieves the current date and time on the TARGIT InMemory Database Server. See also: TRUNC, DATE, FORMAT, MONTH, DAY, YEAR.

GETUTCDATE ( )

Retrieves the current UTC (Coordinated Universal Time) date and time.

IIF (condition, valueifTrue, valueifFsalse)

Returns *valueIfTrue* if *condition* is TRUE otherwise valueIfFalse.

INSERT (base_string, startPos, strtoInsert)

Insert *strtoInsert* into *base_string* at *startPosition.*

ISNULL (value, alternateValue)

Returns *alternateValue* if *value* is null. Otherwise returns *value.* See also: ISNULLOREMPTY_AFIELD.

ISNULLOREMPTY(column)

Checks whether the contents of the specified column is NULL or that it is empty. The function returns TRUE (1) if a *column* is null or if its (*string*) length is zero. Otherwise returns FALSE (0). Currently only works with Strings.

Syntax:

SELECT * WHERE ISNULLOREMPTY(*columnname*)=FALSE|TRUE

See also: ISNULL.

Example:

/* Get all customers from the customer table where country is Denmark and where the customer name is not NULL or empty */ SELECT * FROM Customer WHERE Country='Denmark' and ISNULLOREMPTY(CustomerName)=FALSE

LEFT (string, number)

Returns the left most *number* chars of *string.*

LEN (string)

Returns the length of *string* as an integer.

LCASE (string)

Returns string in lower case.

LOG (number)

Returns the natural logarithm of a *number.* See also: LOG_NUMBER_BASE.

LOG (number, base)

Returns the logarithm of a *number* for the given *base*. See also: LOG_NUMBER.

LTRIM (string)

Removes leading whitespace from *string.*

MAX (number1, number2)

Returns the greater of *number1*, *number2*. See also: MINLIST, MIN, MAXLIST.

MAXLIST (val1, val2… valn)

Returns the max value of *val1*,..*valn*. It supports an arbitrary number of parameters. See also: MINLIST, MIN, MAX.

MIN (number1, number2)

Returns the lower of *number1*, *number2*. See also: MINLIST, MAX, MAXLIST.

MINLIST (param1, param2, param3 ,.. paramn)

Returns the minimum value from list of parameters *param1* through *paramn*. See also: MIN, MAX, MAXLIST.

MONTH (date)

Returns the month no of the *date* parameter. See also: FORMAT for a textual representation of the month name, day names etc.

MONTHNAME (date)

Returns the full month name of the *date* parameter. See also: FORMAT for a textual representation of the month name, day names etc.

OPENBAL (measure, [DateColumnName])

This functions provides native support for doing balances, e.g. for onhand inventory or general ledger balances. OPENBAL returns the opening balance – which is the sum of the measure since beginning of time until "yesterday".

Syntax:

SUM(OPENBAL(*measure*,*[DateColumnName]))*

Note: In case there are no transactions for a specific date the date will be excluded from the output. See also: CLOSEBAL

Example:

SUM(OPENBAL(*measure*,*[DateColumnName]))*

OPENBALWITHZEROES (measure, [DateColumnName])

This functions provides native support for doing balances, e.g. for onhand inventory or general ledger balances. OPENBAL returns the opening balance – which is the sum of the measure since beginning of time until "yesterday". Compared to OPENBAL, OPENBALWITHZEROES will add additional rows to the rowset if there are missing entries derived from the date being used.

Syntax:

SUM(OPENBALWITHZEROES(*measure*,*[DateColumnName]))*

POW (number, power)

Returns *number* raised to *power*.

RAND()

Returns a random double that will vary from 0 to 1.

REMOVE (string, start_pos)

Removes all characters in *string*, beginning with position *start_pos*.

REPLACE (string1, string2, string3)

Returns a string which replaces all occurrences of *string2* in *string1* with *string3.*

REVERSE (string)

Reverse character positions of the given *string* –eg. 'abcd' becomes 'dcba'.

RIGHT (string, number)

Returns the rightmost *number* chars of *string.*

ROUND (number, decimal_places)

Rounds a given *number* to the *decimal_places* limit.

RTRIM (string)

Removes trailing whitespace from *string.*

SAFECAST (expression)

The SAFECAST function casts the input data type to the explicit value without causing exceptions. SAFECAST is when used with tiImport code additionally affected by the SET CULTURE command.

Syntax:

SAFECAST(*Expression* as *DataType*) .See also: CAST

Example:

SAFECAST(*Expression* as TEXT)
SAFECAST(*Expression* as DATETIME)
SAFECAST(*Expression* as BIT)
SAFECAST(*Expression* as TINYINT)
SAFECAST(*Expression* as SMALLINT)
SAFECAST(*Expression* as INTEGER)
SAFECAST(*Expression* as BIGINT)
SAFECAST(*Expression* as REAL)
SAFECAST(*Expression* as DOUBLE)
SAFECAST(*Expression* as DECIMAL)

SIGN (number)

Returns the sign of the *number*. 1 if > 0, 0 if = 0 , -1 if < 0.

SIN (number)
Returns the Sine of *number.*

SINH (number)
Returns the Hyperbolic Sine of *number.*

SRQT (number)
Returns the square root for the given *number*.

STARTSWITH (string, substring)
Returns True if a *string* starts with substring.

STDEV | STDDEV (numeric fieldname)
Returns the standard deviation for all values in the selected *field*. See also: STDEV_P.

STDEV_P | STDDEV_P (numeric fieldname)
Returns the population standard deviation for all values in the selected *field*. See also: STDEV.

SUBSTRING (string, start_pos)
Returns the string value of *string*, beginning in position *start_pos.*

SUM (expression)
Returns the sum of the column specified in the expression. SUM can be used with numeric columns only.

Syntax:

SUM (expression)

Example:

```
SELECT Size, SUM(Price), SUM(Cost)
FROM Merchandise.Product
WHERE Size IS NOT NULL
AND Price != 0.00
AND Name LIKE 'Tshirt%'
GROUP BY Size
ORDER BY Size
```

TAN (number)
Returns the TAN of *number.*

TANH (number)
Returns the hyperbolic TAN of *number.*

TRIM (string)
Removes leading and trailing whitespace from *string.*

TRUNC (date)
Removes the time part from a *date.* The example below shows a date column generated from the current server date and time (GETDATE) and the TRUNC'd VALUE.

Example:

```
SELECT GETDATE() as CurrentDateWithTimeStamp, TRUNC(GETDATE()) AS TruncDateOnly
/* Returns the table
CurrentDateWithTimeStamp          TruncDateOnly
6/3/2016 9:42 AM                  6/3/2016
*/
```

UCASE (string)
Returns *string* in upper_case.

VAR | VARIANCE (numeric fieldname)
Returns the variance for all values in the selected *field*. See also: VARP.

VARP | VARIANCE_P (numeric fieldname)
Returns the population variance for all values in the selected *field*. See also: VAR.

YEAR (date)
Returns the year of the *date* parameter.

Functions and Stored Procedures

STORED PROCEDURES

TARGIT InMemory supports stored procedures. Stored procedures are created with the **CREATE PROCEDURE** Command and dropped with **DROP PROCEDURE**. They are executed using the **EXEC** or **EXECUTE** commands. **RETURN** is used to return from a Stored Procedure.

Stored Procedures accept parameters.

Example 1 - Declaring a SP with 1 parameter and using EXEC:

```
CREATE PROCEDURE myProc @param1 integer as
    BEGIN
    SELECT * FROM ORDERS
      where orderid in (@param1 )
    END
```

```
EXEC myProc 10248
```

Or

```
EXEC myProc @param1=10248
```

Example 2 - Setting a default value to a parameter:

```
CREATE PROCEDURE myProc3 @param1 integer = 10250 as
    BEGIN
    SELECT * FROM ORDERS
      where orderid in (@param1 )
    END
```

```
EXEC myProc3
```

DYNAMIC SQL – EXEC / EXECUTE

Dynamic SQL is executable with the EXEC / EXECUTE statement. It takes one string parameter.

Example:

```
declare @tableName as varchar

set @tableName = 'Orders'

execute ('select * from [' + @tableName +']')
```

You can also execute dynamic SQL with the built in **sp_executesql** store procedure. This can take multiple parameters. The first parameter is the dynamic SQL statement. The next N are variable definitions for the dynamic SQL, and the next N are the values for variables.

Example:

```
exec sp_executesql 'select * from orders where orderid=@orderid' , '@orderid int' , 10248
```

SCALAR USER DEFINED FUNCTIONS

CREATE FUNCTION defines a scalar user defined function. Functions are dropped with DROP FUNCTION. These functions can be used in SET and DECLARE expressions and can take constants and variables as parameters within SELECT or other SQL statements. When operating within other SQL statements, they cannot accept column values as input. They are evaluated at the start of the SQL Statement and have a constant value for the rest of the life time of the SQL statement.

Example - Squares the parameter passed as input:

```
CREATE FUNCTION udf_squared (@nr int)
   RETURNS long
   as begin
     declare @result as long
    set @result = @nr*@nr

     return @result
   end

select udf_squared(1000)
```

TABLE VALUE FUNCTIONS

Call parameterized functions within the FROM clause of a SELECT statement.

The function returns a table in a specified format and the results work like a table with the specified alias.

Table value functions can take 0-n nparameters.

Example - Shows how to declare a table value function that takes one parameter and outputs a table with two columns. @outputTable works as virtual table where the output goes.

```
create function tvf_example ( @param1 integer )

returns @outputTable table ( col1 int ,col2 int )

as BEGIN

  insert into [@outputTable] select @param1,2

end

select * from tvf_example(10248) as xxx
```

INLINE TABLE VALUE FUNCTIONS

A simplified version of a table value function with one SQL statement. It is not necessary to declare the output table format. The following example shows how to declare an inline table value function with one parameter, and how to call it from a SELECT statement.

Example:

```
CREATE FUNCTION getOrder ( @orderid int  )
RETURNS TABLE
as RETURN (    select * from orders where orderid in  (@orderid  ) )
select * from getOrder(10248) as OrderInfo
```

SPLIT_TABLE TABLE VALUE FUNCTION

SPLIT_TABLE is a special built in table value function. It takes 3 parameters. The first parameter is a SELECT statement. The second parameter is a column in the SELECT statement and the third parameter, is a character to parse for. The function is designed to take the column specified by the second parameter and generate a new table splitting the data one per row by the third parameter. This function is designed to help parse information within a column into separate rows.

Example - Produces a result set with 3 rows. A in row 1, B in Row 2, C in row 3:

```
select * from split_table ( (select 'a,b,c'  as col1    )   , col1 , ',' ) as xyz
```

System Specific Tables and Objects

Within the TARGIT InMemory Database there are some system specific tables and objects that can be accessed from both tiImport and tiQuery.

INFORMATION_SCHEMA.COLUMNS

Returns a list of all tables and columns in the current InMemory database.

Table_Catalog	Name of the tables catalog (targitdb - static)
Table_Schema	Name of the tables schema (dbo - static)
Table_Name	Name of the table
Column_Name	Name of the column
Ordinal_Position	Position of column within table
Data_Type	Type of data

Example:

SELECT * FROM INFORMATION_SCHEMA.COLUMNS.

INFORMATION_SCHEMA.TABLES

Returns a list of table information for the current InMemory database. The following information will be returned:

Table_Catalog	Name of the tables catalog (targitdb - static)
Table_Schema	Name of the tables schema (dbo - static)
Table_Name	Name of the table
Table_Type	Type of the table (BASE TABLE - static)

Example:

SELECT * FROM INFORMATION_SCHEMA.TABLES.

INFORMATION_SCHEMA.UPDATED

Returns a list with information about the information for the current database.

Last_Loaded	Date and Time information for the last time the file was loaded from storage
Loaded_File_Created	Date and Time information for the last time the file was created
Loaded_File_Size	Size of the loaded file
Estimated_Memory_Usage	Estimates how much memory the file will consume when unpacked and loaded into memory
Current_File_Size	Size of the current file
Current_File_Created	Date and Time information for when the file was created

Example:

SELECT * FROM INFORMATION_SCHEMA.UPDATED.

INFORMATION_SCHEMA.TABLE_USAGE

Shows memory usage for each table in the database.

Example:

SELECT * FROM INFORMATION_SCHEMA.TABLE_USAGE.

INFORMATION_SCHEMA.COLUMN_USAGE

Shows memory usage for each column in the database.

Example:

SELECT * FROM INFORMATION_SCHEMA.COLUMN_USAGE.

INFORMATION_SCHEMA.QUERY_LOG

Shows a list and stats on recently run queries. By default, the query_log is turned off. You can enable the log by running the sql statement ***SET querylog = 1000***, and the most recent 1000 queries will be logged. You can disable it by running ***SET querylog = 0***.

Example:

SELECT * FROM INFORMATION_SCHEMA.QUERY_LOG.

INFORMATION_SCHEMA.QUERY_LOG_DELTA

Shows queries since the last time query_log or querylog_delta was queried.

DBSTATS

Use DBSTATS to get statistical information on the databases on the server and can be run from both import scripts and Query Tool. It will show

Database Name, Status (NOT_LOADING / LOADING / LOADED), Estimated Memory Usage, Time Loaded, Time Created, Disk Size when loaded, Latest File Created and Latest File Size

This is useful to manage memory usage and the load status of databases to reduce memory consumption on the server.