CONNX
INTEGRATION WITH MICROSOFT® SQL SERVER®

Larry McGhaw, VP, R&D CONNX, Software AG

Techie Deep Dive

SQL Server linked servers

SQL Server's linked server feature enables fast and easy integration of SQL Server data and non-SQL Server data, directly in the SQL Server engine itself. It has many technical features. Each is individually described in various locations in SQL Server documentation or knowledge base articles. This guide is an attempt to provide context to these items. Also, we describe the reasons why certain options exist and their ramifications.

Building a linked server connection

Building a linked server connection requires a couple of ingredients:

1. SQL Server
2. A third-party driver, OLE DB or ODBC for the database to be integrated with SQL Server, such as CONNX

Using CONNX, SQL Server easily integrates with all of the data sources supported, including Adabas. Linked servers can be created in two ways: either via the SQL Server Management Studio; or via a stored procedure called sp_addlinkedserver. If there is a need to dynamically create linked servers via a script, then the stored procedure route is better. Please refer to the existing detailed SQL Server documentation on this stored procedure if desired. Using the SQL Server Management Studio, linked servers can be found under "Security Objects." Add a new linked server by right-clicking on "Linked Servers." And then select "New Linked Server…"

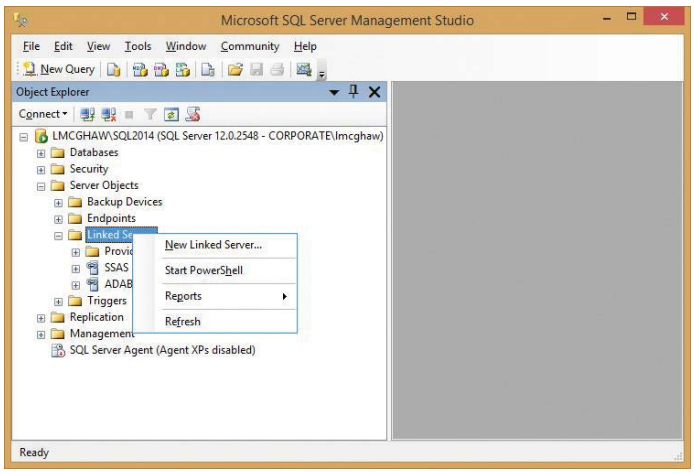

Figure 1: Creating a new linked server from the Management Studio

After selecting the name of your linked server connection, you must decide what type of driver to use to access the non-SQL Server database.

Microsoft is moving away from OLE DB and standardizing on ODBC again for access to external data sources. Select the "Microsoft OLE DB Provider for ODBC Driver."

Once the driver has been selected, fill in the remaining fields. The "Product Name" field is not used for any technical purpose. But it is required to complete the linked server connection. Please refer to the documentation of the third-party driver for details on what to supply for the remaining fields. In the provider string field, specify the CONNX driver, along with the name of the CDD, and any desired application filters.

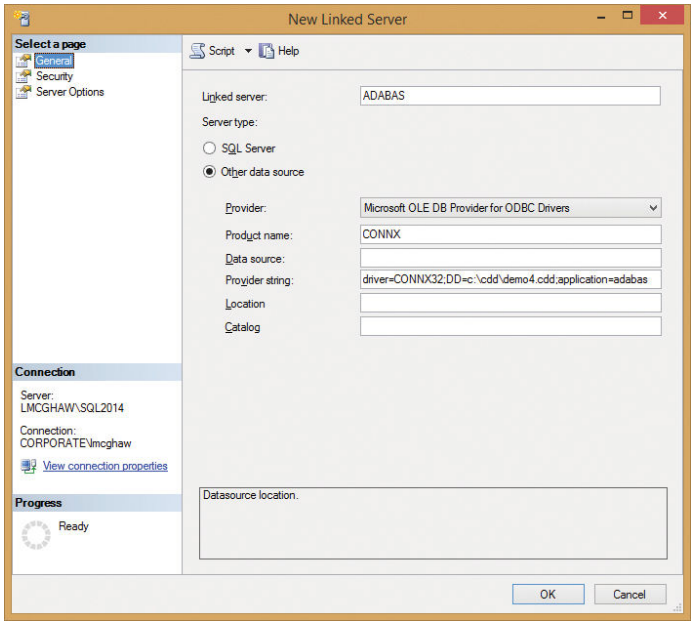

Figure 2: Specifying the linked server driver connection properties

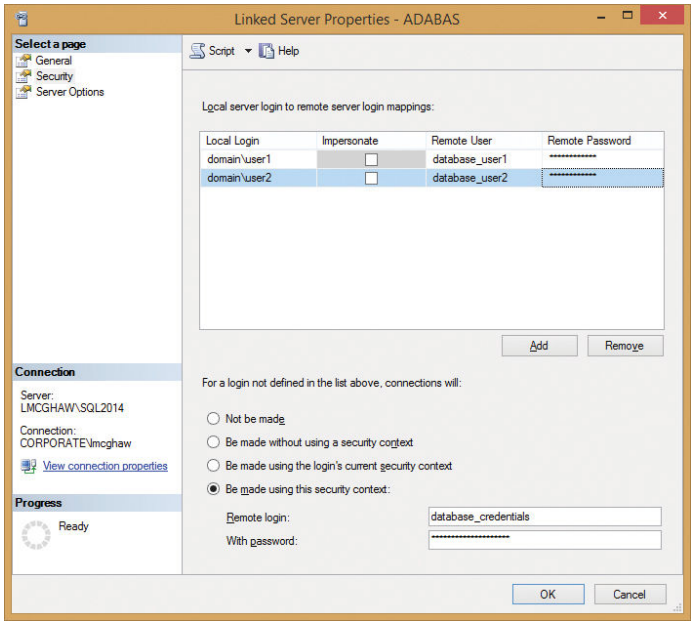

Figure 3: Specifying the authentication method for the linked server connection

Don't press the "OK" button yet. Important authentication and security options are located on the "Security" page. This can be selected by clicking Security on the upper left hand side of the dialog. Some databases do not require a username and password to connect. In these cases, select the option "Be made without using a security context." However, most databases require a username and password. This makes the process of connecting to them through SQL server a bit challenging. A case in point: a SQL Server is configured to use Microsoft® Active Directory® logons. But the other database does not support this type of logon. If the desired data source does not support active directory authentication, the mismatch in authentication credentials would make it impossible to connect. The solution to the problem is provided on the following dialog. Here, a static set of credentials for the other database will be used for all linked server connections. Select "Be made using this security context." Then provide the appropriate username and password for the other database.

If at all possible, do not use a single set of authentication credentials for all linked server connections. Instead, a separate mapping is established for each SQL Server logon by pressing the "Add" button on the Security page. The "impersonate" option is just another word for using Active Directory authentication for the other database – do not select this checkbox. Additional important options can be found by clicking "Server Options" in the upper left area of the "New Linked Server" dialog.

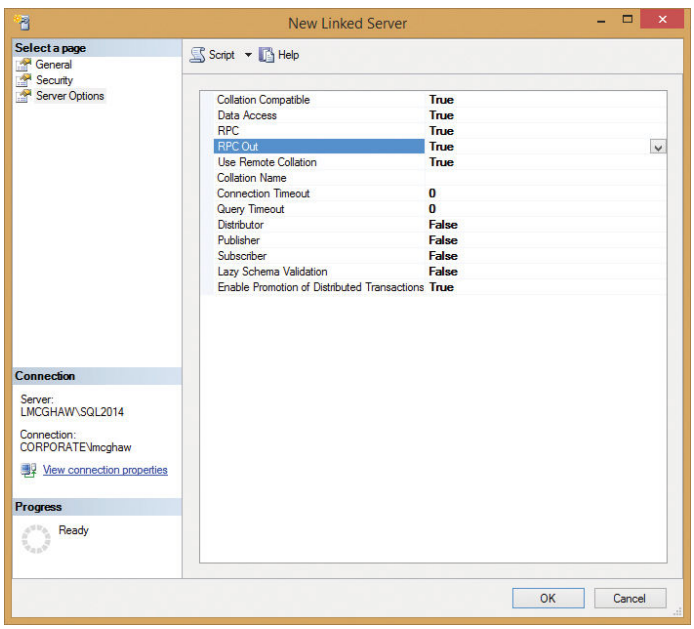

Figure 4: Specifying the linked server "server options"

The following table lists each of the options and its function:

Collation Compatible	If the codepage and sort order of the other database is the same a SQL Server, enable this option for better performance. By default this option is not enabled. This means that SQL Server will re-sort all data returned from the third-party driver. This option should be turned on.
Data Access	Enables or disables the use of queries with the third-party driver. Obviously, this needs to be turned on.
RPC & RPC OUT	Enables the use of stored procedures stored within the other database. This option should be turned on.
Use Remote Collation	If the collation of a column in the other database is different than the default collation on the SQL Server, SQL Server will honor the column collation as specified by the linked server instead of re-sorting the data using the default SQL Server collation if this option is enabled.
Collation Name	The collation of the other database can be specified here if it differs from SQL Server.
Connection Timeout & Query Timeout	Timeout options
Distributor, Publisher, Subscriber	These options are typically used in SQL server to SQL Server or Oracle® to SQL Server replication.
Lazy Schema Validation	If set to true: schema validation occurs at execution time verses prepare time, when it is set to false. For most third-party drivers, this has no performance impact.
Enable promotion of distributed transactions for RPC	When enabled, SQL Server will automatically start a distributed transaction when a stored procedure is executed via the third-party driver.

The important options here are "Collation Compatible" and "Data Access." "Collation Compatible" improves performance. It should be enabled. "Data Access" must be enabled for the linked server to work with queries.

Once all options for the new linked server have been set, press OK to create the linked server. To validate connectivity to the linked server, double click on the linked server name in SQL Server Management Studio. Then, expand the catalog until you can see your tables.

Accessing your non-SQL server data

There are four ways to access data with CONNX with SQL Server: two ways use linked servers, and two ways do not.

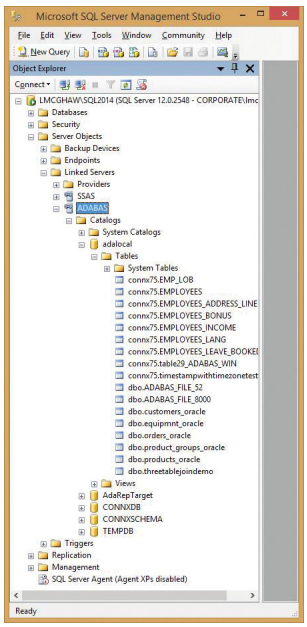

Figure 5: Tree view of tables available via a linked server connection

The entire process of creating a linked server can be skipped by using OPENDATASOURCE and OPENROWSET. Both of these options enable direct access to the CONNX driver via SQL without any prior linked server creation. When using these two methods, there are many driver configuration options that are not available for configuration such as collation compatibly. Additionally, both of these ad-hoc query methods are disabled by default in SQL Server. To enable these ad-hoc methods execute the following commands:

```
sp_configure 'show advanced options', 1;
RECONFIGURE;
Go

sp_configure 'Ad Hoc Distributed Queries', 1;
RECONFIGURE;
Go
```

The potential danger in enabling this option is that now any user can access data from any OLE DB Provider or ODBC Driver on the SQL Server … not just the ones that have been specifically defined as linked servers. However the option is available and it is up to each admin to determine the risk of these features based on the project and security needs.

Here is an example of the use of OPENDATASOURCE:

```
SELECT *
FROM OPENDATASOURCE('MSDASQL',
 'driver=connx32;DD=c:\cdd\demo4.cdd;application
=adabas;UID=xxxxx;PWD=xxxxx')
 .adalocal.connx75.employees
```

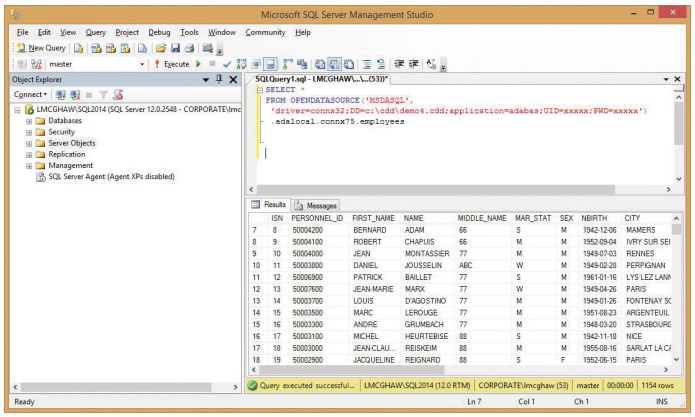

Figure 6: Example of the OpenDatasource statement

And a corresponding example of the use of OPENROWSET:

```
SELECT a.*
FROM OPENROWSET('MSDASQL',
'driver=connx32;DD=c:\cdd\demo4.cdd;APPLICATION=
ADABAS;UID=XXX;PWD=XXX;',
    'SELECT e.first_name, e.name, en.LANG
    from adalocal..employees e inner join
    adalocal..employees_lang en
      on (e.isn = en.isn)
      where e.city = ''PARIS''
    ORDER BY e.name DESC') AS a;
```

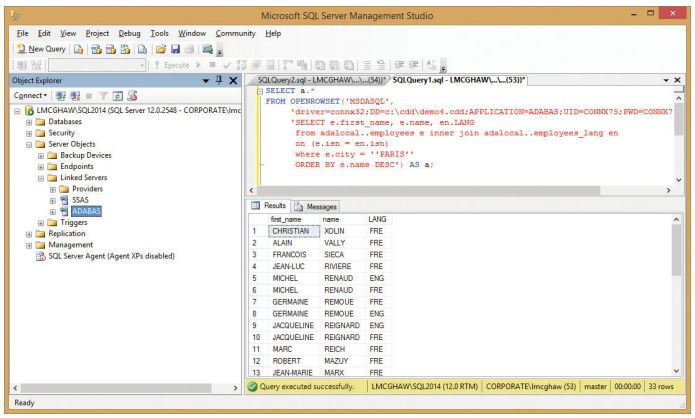

Figure 7: Example of the OpenRowset statement

Accessing data with linked server objects

The remaining two ways to query data via a third-party driver involve using linked server objects. The first method involves using a four-part name with the following syntax:
<Linked Server Name>.< Object Catalog>.< Object Owner>.<Object Name>

If your third-party driver does not support catalog or owner names, leave those portions empty. For example if a provider does not support catalog names, but does support owner names, the syntax would look like this:
Select * from MyLinkedServer..richard.products

For a driver that does not support either catalog names, or owner names, it would look like this:
Select * from MyLinkedServer…products

And for a fully compliant driver that supports both catalog and owner names, like CONNX, it would look like this:
Select * from MyLinkedServer.oregon.richard.products

The advantage of the four-part name syntax is that it is the easiest and most compact to write. Users can leverage their existing knowledge of SQL Server syntax, and mix SQL Server and non-SQL Server tables in a single SQL statement with ease. The huge disadvantage of using the four-part name syntax is performance. When performing any type of complex SQL operation on multiple tables in the linked server connection, SQL Server will typically download a snapshot each table into SQL server. And, then it performs the filtering and joins operations on those snapshots. For large tables this is a performance bottleneck. The way to avoid this is to use the fourth and final option of data access – the OPENQUERY syntax.

Using the OPENQUERY syntax, a complex sub-component of a larger SQL statement can be sent directly to the third-party driver. This increases performance. An example of the syntax is shown here:

```
SELECT a.*
FROM OPENQUERY(ADABAS,
    'SELECT e.first_name, e.name, en.LANG
    from adalocal..employees e inner join
    adalocal..employees_lang en
      on (e.isn = en.isn)
      where e.city = ''PARIS''
    ORDER BY e.name DESC ') AS a;
```

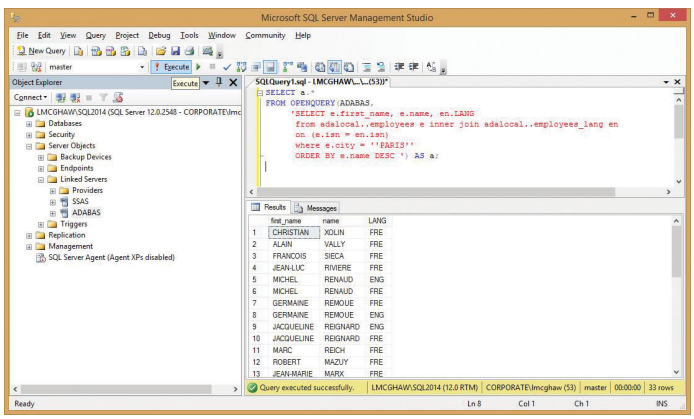

Figure 8: Example of the OpenQuery statement

Alternatively, the exec syntax can be used, providing the ability to specify parameters:

```
exec ('SELECT e.first_name, e.name, en.LANG
from adalocal..employees e inner join
adalocal..employees_lang en
on (e.isn = en.isn)
where e.city = ?
ORDER BY e.name DESC ', 'PARIS') at ADABAS;
```

Global OLE DB provider options

Each OLE DB provider has eight global options that can be configured for all linked servers for that provider.

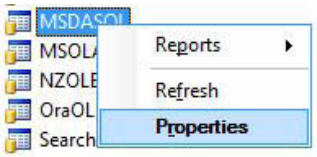

Figure 9: Accessing global properties for an OLE DB Provider

Select MSDASQL, as this is the short name for the OLEDB provider for ODBC drivers.

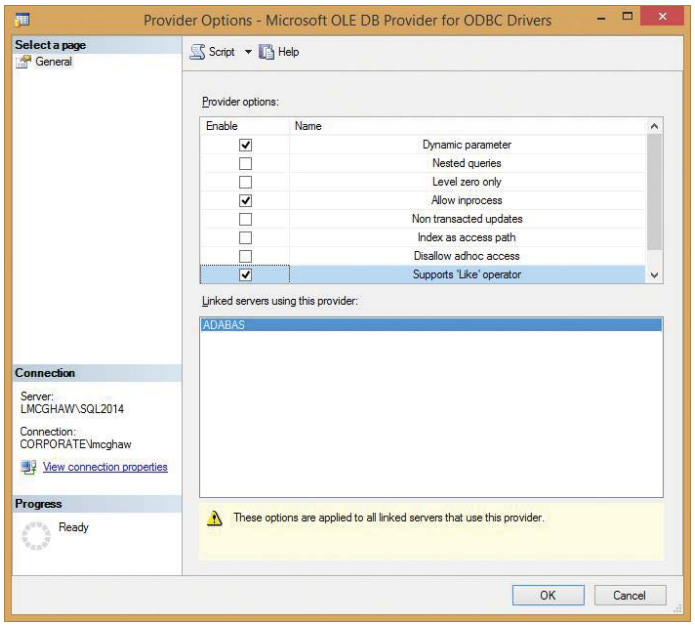

Figure 10: Global properties for an OLE DB provider

Dynamic Parameter	Enables SQL server to send parameterized queries to the OLEDB provider. It can result in greater performance. This option should be selected.
Nested Queries	Enables SQL Server to send "nested" queries. The table name is actually a sub-select, offloading the query processing to the third-party provider. Many third-party providers do not support nested queries. So, typically, this option is turned off.
Level Zero Only	Zero level refers to "basic" OLE DB query interfaces. Most OLE DB providers only implement these basic query interfaces. This option should not be selected.
Allow In Process	This option affects both memory usage and performance to a large extent. "In process" means the OLE DB provider will be loaded in the same process as the SQL Server engine. The advantage of In process is performance. The disadvantage is that SQL Server and the OLE DB provider are fighting for the same process virtual memory. SQL Server typically reserves all but a small amount of memory for itself. If your OLE DB provider is running out of memory, you may need to add or adjust the "-g" SQL Server command line option. The "-g" memory_to_reserve option indicates, in megabytes, how much memory SQL Server will leave alone for third-party drivers and components to use. The default amount is 256M. Depending on how heavily linked servers are used, this may need to be increased. If the "Allow In Process" is not enabled, then SQL Server will create a surrogate process and load the third-party driver in that process instead. This gives the third-party driver its own separate memory space. But, it also results in slower performance due to the time it takes to transfer data from the surrogate process to the main SQL Server process and also the load time of the driver in the surrogate process. The surrogate process provides complete isolation of the third-party driver and the SQL Server database engine. So any instability in the driver will not affect SQL Server itself.

Allow In Process	Additionally, from a Windows® security perspective, drivers that are loaded under the surrogate process have the impersonated security context of the SQL server user. The driver may not have access to the physical resources on the SQL Server computer required of the driver. Drivers loaded in process have a Windows security context of the SQL Server engine itself. This is usually sufficient to access any resources required by the driver.
Non Transacted Updates	By default, SQL Server uses Microsoft Distributed Transaction Coordinator (MSTDS) to ensure that any updates made to date in the third party driver are properly synced to SQL Server transactions. This is typically called a two-phase commit. If your third-party does not support distributed transactions (this option is only available with OLE DB providers, not ODBC driver), and you still want to allow updates and take the risk of SQL Server commits not being in sync with commits to you third-party driver, enable this option.
Index as Access Path	This option is typically for very basic OLE DB providers that don't support SQL. But they do support accessing data via indexes. This option is normally not enabled.
Disallow Adhoc Access	Enable this option if you want to prevent access to the linked server.
Supports 'Like' Operator	Enable this option if your third party driver supports the like operator. Most providers for modern databases support this. This option should be selected.

In summary, SQL Server provides many ways of integrating CONNX data sources (such as Adabas) and SQL Server data together. Each has its own set of advantages and disadvantages. The security and business needs of each project should be evaluated against the available options to determine what is best for the specified use case.

To learn more, visit www.connx.com.

ABOUT SOFTWARE AG

Software AG (Frankfurt TecDAX: SOW) helps companies with their digital transformation. With Software AG's Digital Business Platform, companies can better interact with their customers and bring them on new 'digital' journeys, promote unique value propositions, and create new business opportunities. In the Internet of Things (IoT) market, Software AG enables enterprises to integrate, connect and manage IoT components as well as analyze data and predict future events based on Artificial Intelligence (AI). The Digital Business Platform is built on decades of uncompromising software development, IT experience and technological leadership. Software AG has more than 4,500 employees, is active in 70 countries and had revenues of €879 million in 2017. To learn more, visit www.softwareag.com.

SAG_CONNX_Integration_With_SQL_TECHniques_Apr18

software AG