# ECM Distributed Repository Architecture

In dealing with unstructured data, the potential volume of data and size of databases can rapidly become overwhelming. Also when one considers that a 10GB size database provides optimum response time, the question arises, “How can we keep our data safe, secure, contained, and our searches fast… no matter the size of the database?” This is a tricky question to say the least. We do not look at the current performance of ECM as a problem, but how to maintain that blazing speed when databases can potentially grow into the hundreds of Terra bytes is our focus. Our development team is developing a new approach to parallel/distributed databases. And, the distribution will be done automatically by the ECM Parallel Active Databases management software. The ECM PAD is an integrated monitoring service that will monitor all registered ECM Repositories and when any Repository reaches threshold (a predetermined size), will create and add to the cluster another content repository and immediately start directing archives and searches across the full cluster (ring) of repositories. We are not just suggesting “adding another database to the existing SQL Server instance, but far above and beyond that. When performance gains are needed on very large databases, and the stored procedures are already optimized, and the application is already n-tiered out, and hardware is already upgraded, it is time for distributing the database work over ***several servers***. For ECM Repositories, this is done by horizontally partitioning large tables over multiple servers. If we think of splitting one table with many columns, into several tables of only a few columns, as ***Vertical Partitioning of the Data***, then Horizontal Partitioning is the act of taking one table with many rows, and splitting it into many tables with only a few rows. If these new smaller tables are placed on different servers, it is called a Federated Database configuration. The word Federated is used because all the servers involved may cooperate to balance the processing load. They will act as one federation. Once your data is split among several servers, a new type of statement is needed for retrieving records. These new statements are called Distributed Partitioned Views. They use standard SQL statements, along with the key word UNION, to pull data from all the distributed servers. DML statements (INSERT, UPDATE, and DELETE) can also be used when just a few special rules are observed on the underlying tables. While performance gains are going to vary from application to application, gains of 20% to 30% seem common.

Three setup tasks need to be executed. Start by connecting all severs to each other via Linked Servers, followed by creating matching tables on each server, and finally writing the new views.

In this example, I will distribute the SQL EcmRepository "Authors" table across two different servers. NOTE: *The rules and procedures are identical regardless of the number of servers involved.*

**Linked Servers**

*I found a good detailed explanation of linked servers [Linked Servers PART1](http://databasejournal.com/article.php/3085211).*

The first step in creating this “***federation”*** is to link the involved servers together. Start Query Analyzer as "sa," run from the first server. This code will link the second server to the first with an alias of "server2."

USE master

GO

EXEC sp\_addlinkedserver

@server = 'server2',

@srvproduct = 'SQLServer OLEDB Provider',

@provider = 'SQLOLEDB',

@datasrc = 'InfoNet'

The @server variable is our alias. The @datasrc is the actual Sql server name. If there was more than one instance on the server, ServerName\InstanceName would have been used.

If the first server will be running all the views, and your Windows login has the appropriate rights to both machines, then special logins are not needed. However, if a different client will be executing the views, then login mappings may be needed. Of course, I have another like, please see [*Linked Servers PART3*](http://databasejournal.com/article.php/3116011)and <http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/articles/q238/4/77.asp&NoWebContent=1> for detailed explanations of linked server login options and client problems. We will assume remote clients may be wanted in the future, so a linked server login, mapping the local "sa" account to remote "sa" account, is being used as an example. For production, create a new login rather than expose "sa."

EXEC sp\_addlinkedsrvlogin 'server2', 'false', 'sa', 'sa', 'secret'

To test the link, execute:

SELECT \*

FROM server2.EcmRepository.dbo.authors

All of the rows from Authors should be returned. We now need to repeat the same from server 2. This will create a link back to server1. Everything is the same except for the server name and alias. Login to the second server's Query Analyzer as "sa" and run:

USE master

GO

EXEC sp\_addlinkedserver

@server = 'server1',

@srvproduct = 'SQLServer OLEDB Provider',

@provider = 'SQLOLEDB',

@datasrc = 'dons13'

GO

EXEC sp\_addlinkedsrvlogin 'server1', 'false', 'sa', 'sa', 'secret'

GO

SELECT \*

FROM server1.EcmRepository.dbo.authors

If there were additional servers involved, each would require links to every other. The same type of design we had with the old “*NT multi-Domain security trusts*”. If there were four servers in our federation, it would have this landscape:

Server1 has a linked server to Server2, Server3, and Server4.

Server2 has a linked server to Server1, Server3, and Server4.

Server3 has a linked server to Server1, Server2, and Server4.

Server4 has a linked server to Server1, Server2, and Server3.

There is not an automated way to either create, or test these reciprocal links.

### Create Table

For this example, we will be working with a subset of the EcmRepository Authors table. We will imagine the table was very large and most of our lookups were done by last name. In this case, we could split the Authors table in half, with one server getting records with a last name beginning with A - M, and the other server getting records N - Z. From server1, create and load a test table.

CREATE DATABASE EcmRepository

GO

USE EcmRepository

GO

CREATE TABLE DataSourceAM(

Author varchar(40) NOT NULL,

File\_Name varchar(20) NULL,

CONSTRAINT CHK\_DataSourceAM CHECK (Author < 'N')

)

GO

INSERT INTO DataSourceAM

(Author, File\_Name)

SELECT Author, File\_Name

FROM EcmRepository..authors

***WHERE Author < 'N'***

From server2, almost identical code is run again. The check will be changed for our new range of last names. Notice the tables do not need to have the same name on each server. Server1s' table is called DataSourceAM, server2s' table is called DataSourceNZ. Our view will take care of sorting this out.

CREATE DATABASE EcmRepository

GO

USE EcmRepository

GO

CREATE TABLE DataSourceNZ(

Author varchar(40) NOT NULL,

File\_Name varchar(20) NULL,

CONSTRAINT CHK\_DataSourceNZ CHECK (Author not NULL)

)

GO

INSERT INTO DataSourceNZ

(Author, File\_Name)

SELECT Author, File\_Name

FROM EcmRepository..authors

***WHERE Author >= 'N'***

The critical section of code is the constraint. The check must insure that a row has only one exclusive table in which to be located. When our views are run, the Query Optimizer will use these constraints to determine which servers will receive the distributed work.

### Distributed Partitioned Views

The last step is the actual creation of the views. The UNION operator is used to merge the results from both tables into one result set. Please refer to my previous paper on "The Speed if the Union operator" for detailed Union explanations and rules. This approach will allow the AFFINITY of each server to run the submitted query in parallel in a distributed fashion – that is it is distributed across potentially many smaller databases rather than one huge one. The query will run fast and the reassembling of the different parts if the query record set will be performed by the issuing server. As an added bonus, the user load can also be distributed across these same servers as an administrator deems necessary. This will, in theory, keep the speed remarkably fast and the efficiency of the servers near optimum without concern for the ultimate size of the overall repository.

**It needs to be noted that this approach requires no modification to the code within ECM Library. This is what I do – I exercise creativity – if I can’t do that, then I may as well be a consultant where creativity is neither expected nor demanded and the pay is still good.**

**From server1:**

CREATE VIEW AllMyContent

AS

SELECT \*

FROM DataSource

UNION ALL

SELECT \*

FROM server2.EcmRepository.dbo.DataSourceNZ

GO

**From server2:** the code again is almost identical. Only the server names are changed.

CREATE VIEW AllAuthors

AS

SELECT \*

FROM DataSourceNZ

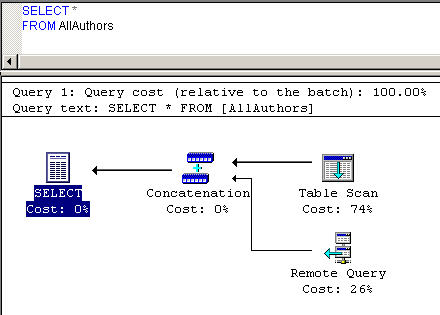
UNION ALL

SELECT \*

FROM server1.EcmRepository.dbo.DataSourceAM

GO

Running a simple select from server1 produced this Execution Plan:



We can see the local table scan on server1 is merged with a remote query on server1. Our view follows all the standard rules for views. So returning only subsets requires nothing more than a standard where clause:

SELECT Author, CreateDate, LastUpdateDate

FROM DataSource

WHERE CreateDate BETWEEN '1/1/2007' AND '7/1/2007'

### Conclusion

While setting up Federated servers is not a quick task, the performance gains produced by Distributed Partitioned Views on large tables can make it more than worthwhile. We will continue next month with the requirements for DML on Federated servers, and optimization techniques.