Advanced Database Group Project - Distributed Database with SQL Server

Hung Chang, Qingyi Zhu

Erasmus Mundus IT4BI

1. Introduction

1.1 Motivation

Distributed database is vague for us. How to differentiate the database system is the distributed database? Do they have the degree of distribution? For such a case in Taiwan is the distributed database because of different locations and the query ability? First, Five SQL Server databases located in five different flower markets and transmitted data in CSV file format to one data warehouse as figure 1.1. The transaction data generated by Auction Clock won't update after 12 A.M owing to transactions in all markets finished before 12 A.M. Second, the market managers and the flower growers query the data originally in different location through OLAP.

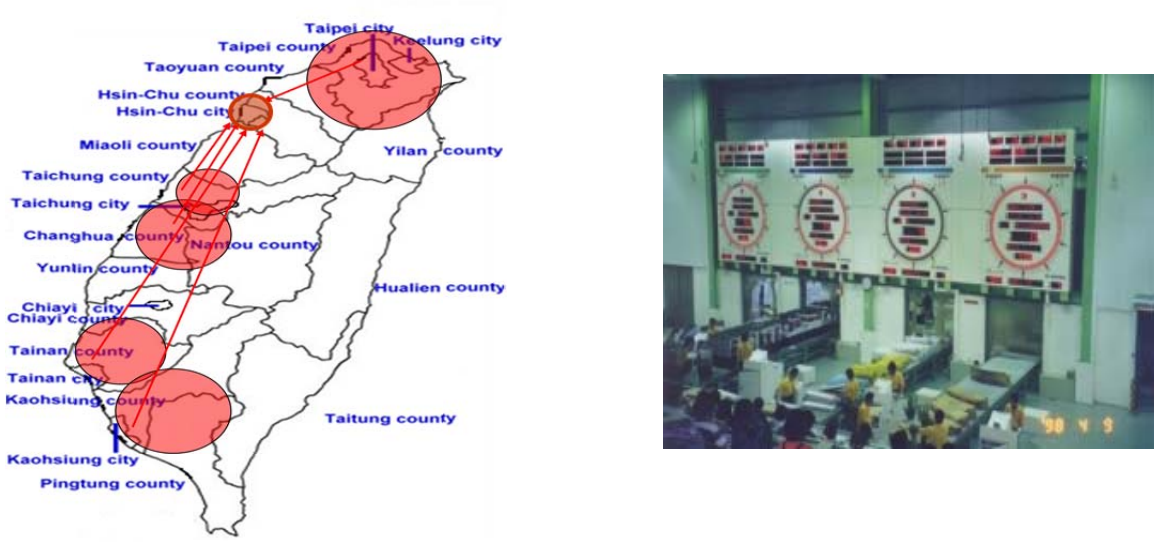

Figure 1.1 : Location of Five Databases and Auction Clocks [1]

In this case, five databases didn't connect together directly through the internet due to the privacy issues. They only uploaded their data through internet. For example, the SQL Server in Kaohsiung city couldn't query the database in Taipei city. They could only query with OLAP. Therefore, are they distributed database? What level of distribution did this case have? Could they distribute better?

1.2 Research Process

We studied the distributed database from both theory and practical. First, reading the book "Principles of Distributed Database System"[2] to know the concepts and definitions, and then searching the keywords in MSDN[3] to understand the distributed database with SQL Server. However, many different terminologies between SQL Server and the book cause the difficulties. The reasons come from a lot of distributed database product share the market based on figure 1.3 [4]. A note of distributed SQL Server [5] exits but without replication section and terminologies matching.

This article describes the different levels of distributed database to solve our confusion, and shows the ability of distributed SQL Server. Meanwhile, discussing the terminologies in "Principles of Distributed Database System" and SQL Server is also the article main task.

Follows are the structure of the article. Section 2 reviews the transparency. Section 3 describes the way SQL Server becomes distributed. Section 4 shows the distributed transaction with SQL Server. Section 5 demonstrates fragmentation with SQL Server. Section 6 demonstrates the replication with SQL Server. Section 7 describes the difficulties when setting up the distributed SQL Server. Section 8 is the conclusion.

Vendor	Product	Important Features
IBM	• DB2 Data Propagator • Distributed Relational Database • Architecture (DRDA) • DB2 Information Integrator	• Works with DB2; replicates data to "regional transactional" databases • Primary site and asynchronous updates • Read-only sites subscribe to primary site • Supports both distribution and consolidation • Heterogeneous databases • Integrates data from multiple types of sources
Sybase	• Replication Server • SQL Anywhere Studio	• Primary site and distributed read-only sites • Update to read-only site as one transaction • Hierarchical replication • Data and stored procedures replicated • Databases located outside the traditional data center
Oracle	• Oracle Streams • Oracle Advanced Replication	• Sharing data with both Oracle and non-Oracle data stores • Hub-and-spoke replication • Synchronous capture for table replication • Multi-master replication supporting distributed applications
MySQL	• Built-in replication capabilities	• Scale-out solutions • Analytics • Long-distance data distribution
Microsoft	• SQL Server 2008	• Multiple types of replication: transactional, merge, and snapshot • Microsoft Sync Framework

Figure 1.3 : Distributed Database Software[4]

2. Transparency Management

Transparent Management means how good the database distributed. With a fully transparent DBMS the developer needs not to pay much attention to deal with the kind of query which needs a distributed, fragmented and replicated database. The transparency services can be provided at three distinct layers. The first layer at which transparency can be provided is the access level. It provides transparent access to data resources. The second layer at which transparency can be provided is the operating system level. State-of-the-art operating systems provide some level of transparency to system users. The third layer at which transparency can be provided is the DBMS level. It provides translations from the operating system to the higher-level user interface. A hierarchy of these transparencies is shown in Figure 2.1. This suggests SQL Server only can provide transparency of replication, fragmentation and language. The Section 4 discusses the Language Transparency.

Replication Transparency means the existence of replicas and the manner of distributing the replicas across the network is transparent. The user applications need to determine how many copies to have and where to put these replicas. It will have an effect on the performance, reliability and availability of user applications.

Fragmentation Transparency means the fragmentation of database is transparent. Since a global query is divided into several fragment queries, the query processing becomes the fundamental issue of dealing with fragmentation transparency.

Considering of the ease of use and the difficulty and the cost of providing high levels of transparency, we need to decide the level of transparency.

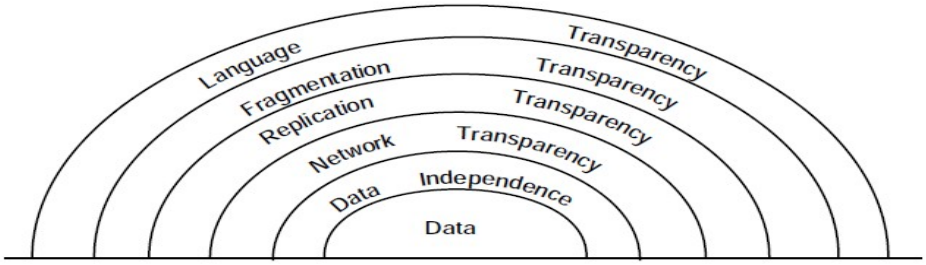

Figure 2.1 : Layers of Transparency

3. Linked Servers and Distributed Database

The book describes distributed database is different databases running on different servers. For instance, Figure 2.1 is not a distributed database because the DBMS only at one site (server). Figure 2.2 is a distributed database because the DBMSs are running on different sites (servers).

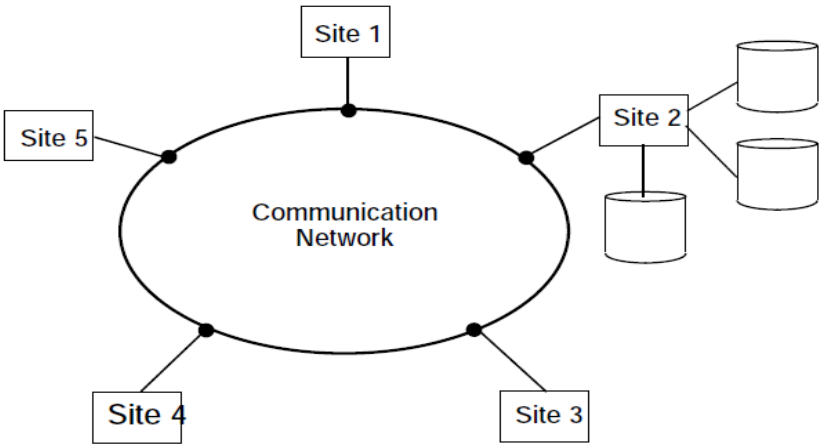

Figure 2.1 : Not Distributed Database[2]

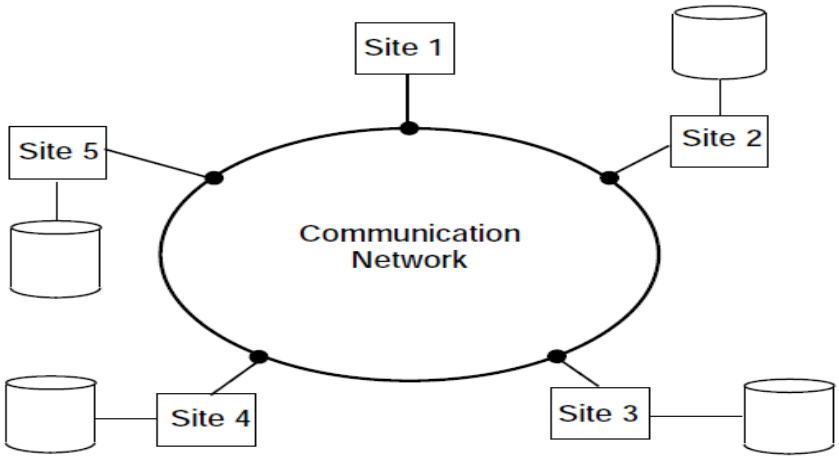

Figure 2.2 : Distributed Database[2]

MSDN names distributed SQL Server as Linked Server. The distributed query on SQL Server explains why Linked Server has the same definition as the book in the next section. Here firstly showing the way to build the distributed database on SQL Server. Two methods which are the Transact-SQL and the configuration of SQL management studio can generate Linked Server. For example, three databases on

different servers link together using a Transact-SQL.

Server 1 : Tainan
Server 2: Taipei
Server 3: Taichung

```
EXEC sp_addlinkedserver
@server=' Tainan',
@provider='SQLNCLI',
@datasrc='TinanDB'

EXEC sp_addlinkedserver
@server =' Taipei',
@provider = 'SQLNCLI',
@datasrc = 'TaipeiDB'

EXEC sp_addlinkedserver
@server = ' Taichung',
@provider = 'SQLNCLI',
@datasrc = 'TaichungDB'
```

Three store procedures created the linked servers at each three servers. sp_addlinkedserver is the name of the procedure. @server is the name of the linked server. @provider is the OLE DB provider. If @provider = 'MSDAORA', the distributed database is heterogamous due to this is an oracle provider. If all @provider = 'SQLNCLI' , it is homogenous distributed database. @datasrc is the data source.

4. Language Transparency and Distributed Query

Transparency presents the level of database distribution. Liked Server is at the Language Transparency level because when querying the tables it must know the name of the partitions, tables and servers.

In SQL Server environment, after creating the linked servers, can query the tables in different servers. Assume the query executes at Tainan city. The Transact-SQL is as follows.

```
SELECT * FROM
flowerDB.dbo.supplier
UNION ALL
SELECT * FROM
Taipei.flowerDB.dbo. supplier
UNION ALL
SELECT * FROM
Taichung. flowerDB.dbo. supplier
```

This example describes the tables needed to query are at the server in Taipei city and server in Taichung city separately. The result returns all the suppliers in those cities.

4. MS DTC and Distributed Transaction

SQL Server database engine and MS DTC(Microsoft Distributed Transaction Coordinator) controls the distributed transactions in SQL Server. A transact-SQL statement starts a distributed transaction when adding the following statement at the beginning.

BEGIN DISTRIBUTED TRANSACTION

Then, the instance of the SQL Server executes the Transact-SQL will become the coordinator in this transaction, and perform local or remote distributed queries throw MS DTC. MS DTC enlists all linked servers involved in this transaction as the participants. Next, the controlling servers apply the Two Phase Protocol between the linked servers to manage the ROLLBACK and COMMIT. The Two Phase Protocol operated as figure 4.1[2]. At first, the local SQL server asks the linked server for preparing to commit. If the participant is ready, it will return Yes else NO. Then the coordinator knows the status of participant is YES and starts to commit. When the participant have committed, it sends the message to the participant and the participant writes end, and the whole process ends. Conversely, the rollback case also shows in figure 4.1.

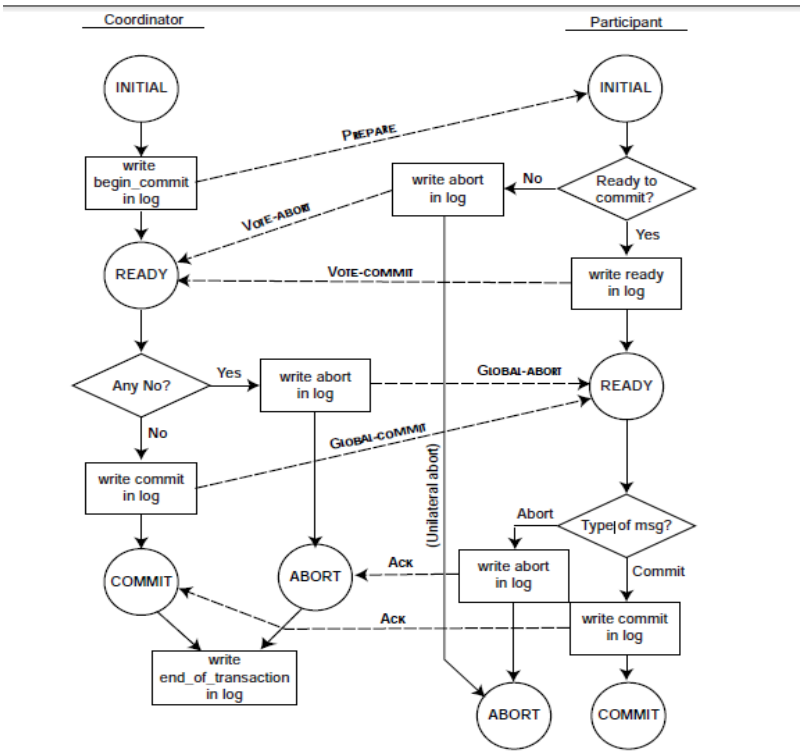

Figure 4.1 : Two Phase Protocol[2]

5. Distributed Partitioned Views and Horizontal Fragmentation

SQL Server uses Distributed Partitioned Views as the horizontal fragmentation. The fragmentation means a table is divided into many tables. One is vertical fragmentation and it separates the columns of a table into two tables which have part of columns in the original table. The other is horizontal fragmentation and it separates the row of a table into two tables which have part of rows in the original table. For example, a flower table has 4 columns and 4 rows as follows. The horizontal fragmentation is as Table 5.2 and Table 5.3 based on the location.

Table 5.1 : A Fragmentation Example

Flower

Name	Price	Quality	Quantity
Lily	30	High	80
Rose	40	Low	40
Phalaenopsis	60	Medium	200
Chrysanthemum	50	Medium	10

Table 5.1 : Horizontal Fragmentation 1

Name	Price	Quality	Quantity
Lily	40	High	80
Rose	60	Low	40

Table 5.2 : Horizontal Fragmentation 2

Name	Price	Quality	Quantity
Phalaenopsis	30	Medium	200
Chrysanthemum	50	Medium	10

SQL Server provides First, executing transact-SQL in one local database and the transact-SQL as follows. through Distributed Partitioned Views and CHECK constraints. Here is an example. Assume that Tainan stores the flower data from FlowerID 1-100, and Tapei stores the flower data from FlowerID 101-200, and Taichung stores the flower data from FlowerID 201-300.

First, create the Distributed Partitioned Views by executing the transact-SQL in each local database as follows.

```
-- On Tainan Server
CREATE VIEW AllFlower AS
SELECT * FROM
flowerDB.dbo.flower
UNION ALL
SELECT * FROM
Taipei. flowerDB.dbo.flower_200
UNION ALL
SELECT * FROM
Taichung. flowerDB.dbo.flower_300

-- On Taipei Server
CREATE VIEW AllFlower AS
SELECT * FROM
Tainan. flowerDB.dbo.flower_100
UNION ALL
SELECT * FROM
flowerDB.dbo.flower_200
UNION ALL
SELECT * FROM
Taichung.flowerDB.dbo.flower_300

-- On Taichung Server
CREATE VIEW AllFlower AS
SELECT * FROM
Tainan. flowerDB.dbo.flower_100
UNION ALL
SELECT * FROM
Taipei.flowerDB.dbo.flower _200
UNION ALL
SELECT * FROM
DBteacher.dbo.Teacher_300
```

The CHECK constraints ensures the rules in the partition tables as following transact-SQL

```
-- On Server1:
CREATE TABLE dbo.flower_100
(ID INTEGER PRIMARY KEY
CHECK (ID BETWEEN 1 AND 100)

-- On Server2:
CREATE TABLE dbo.flower_200
(ID INTEGER PRIMARY KEY
CHECK (ID BETWEEN 101 AND 200)

-- On Server3:
CREATE TABLE dbo.flower_300
(ID INTEGER PRIMARY KEY
CHECK (ID BETWEEN 201 AND 300),
```

The CHECK constraints help the optimization of distributed query to achieve the smallest data delivery between linked servers to minimize the cost of distributed query. Therefore, SQL Server refers the CHECK constraints to better execute the distributed query. For example, we want to know which flower ID is between 51 and 103. The SQL Server will not issue a query to server 3 according to the CHECK Constraints, only query server1 and server2.

```
SELECT *
FROM AllFlower
WHERE ID BETWEEN 100 AND 200
```

6. Replication

6.1 Theory

Replication improves the reliability of distributed database explaining the importance of it. Replication has different degree. For example, the database is fully replicated and stores the replicated databases in different sites, or the database is partially replicated and stores the replicated partitions in different sites. This shows three replication problems. First, duplication of data means choosing one of several copies to retrieve, and update each copy. Second, problem occurs when some sites or some communication links fail while executing an update. The system must ensure that it can recover from the failure. The third problem is about the synchronization. When executing the query, the system need to get data from different sites, so the difficulty is to satisfy the synchronization of transactions on multiple sites. Some protocols are designed to ensure the consistency of the copies of database. These protocols can be eager or lazy. The eager means that all the updates must be applied to all the copies before transaction completes. The lazy means that the updates of other copies can be done after the transaction of one copy is completed.

6.2 Replication with SQL Server

SQL Server calls the problems as the autonomy and latency, and provides the solution with three roles (as figure 6.1). An on-line bookstore illustrates how the SQL Server replication works. The boss of bookstore (publisher) can decide the books (article) to sell, but the books (article) must put in a shelf (publication) which the client (Subscriber) can choose from, and the boss asks express delivery (distributors) for sending the books to the client periodically. In fact, SQL Server Replication Agent controls the mechanism.

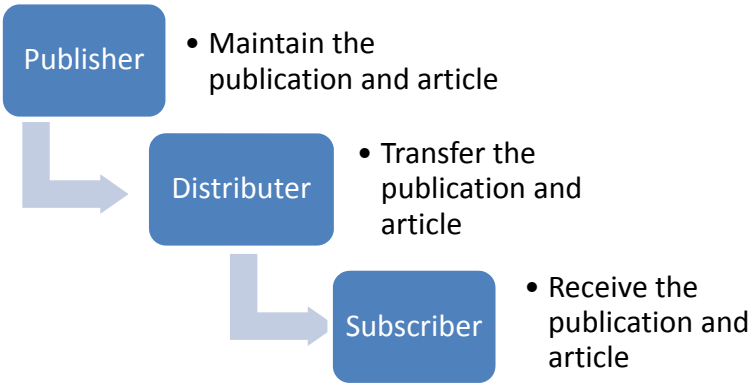

Figure 6.1 : The relation Between Three roles in SQL Server Replication

SQL Server provides three types of replication for use in distributed applications: Transactional replication, Merge replication, Snapshot replication as figure 6.2. The difference is the autonomy and latency issues.

The Snapshot replication distributes data at a specific moment in time without monitoring the updates of data. This type is better if data doesn't change frequently or is few. The Transactional replication responds each time of the change, instead of only respond to the final change while several changes happen. It begins with a publication of database objects and a snapshot of data. After creating the initial snapshot, data changes deliver to subscriber immediately when there are some data changes in publisher. The order of data changes is also the same to guarantee the transactional consistency within a publication. This type is better to deal with the synchronization problem. Besides, it assures low latency between the time changes made at the Publisher and the changes arrived at the Subscriber. The Merge replication uses triggers to track data changes in the Publisher and Subscribers. When subscriber connects to the network, it will synchronize with the publisher. If more than one Subscriber which update the same data at different time, and then spread the changes to server and other subscribers, it's better to use merge replication. Sometimes conflicts exist, and Merge replication deal with them in different ways.

The difference between Merge replication and Transactional replication is that Merge replication doesn't synchronize each change. So, though the data in Subscriber has changed several times, the publisher only reflects the final change.

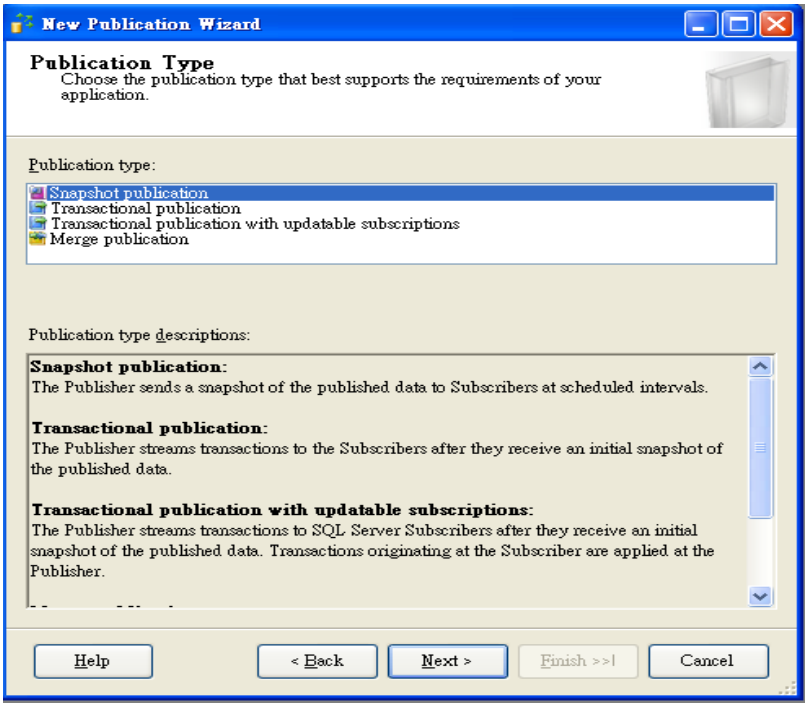

Figure 6.2 Three Main Method for Replication

7. Difficulties of Setting Distributed SQL Server

This section focuses on the trouble-shooting of setting distributed SQL Server. Two virtual machines simulate the distributed database scenario. It is the case because without connecting to the internet, the SQL Server can't connect to the other. If the SQL server on two virtual machines can connect each other without internet, it just represents a server with two databases.

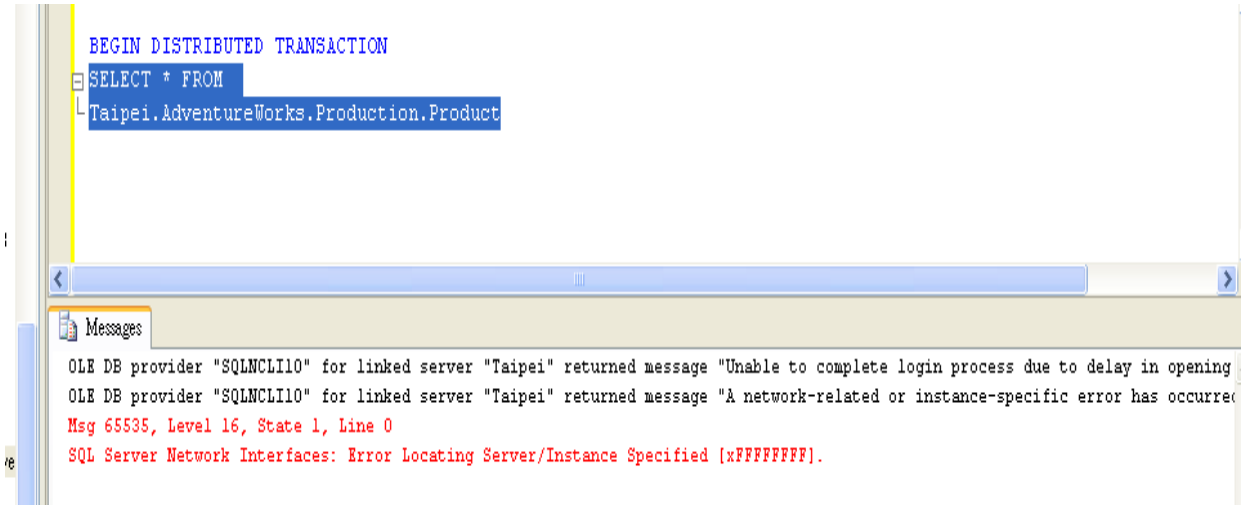

Figure 7.1 : Error Massage when without internet connection.

7.1 Fail to Build Linked Server

Sometimes the servers' links fail due to the network problem. To build linked servers, Firewall, SQL Server Browser, and SQL Server Network Configuration (TCP/IP, Named Pipes) in SQL Server Configuration must be suitable configured.

Figure 7.2 : SQL Server Configuration Manager

If the setting is suitable configured, browsing the network servers in Management Studio can see the database on the other sever. The example runs on VMware to represent two different servers.

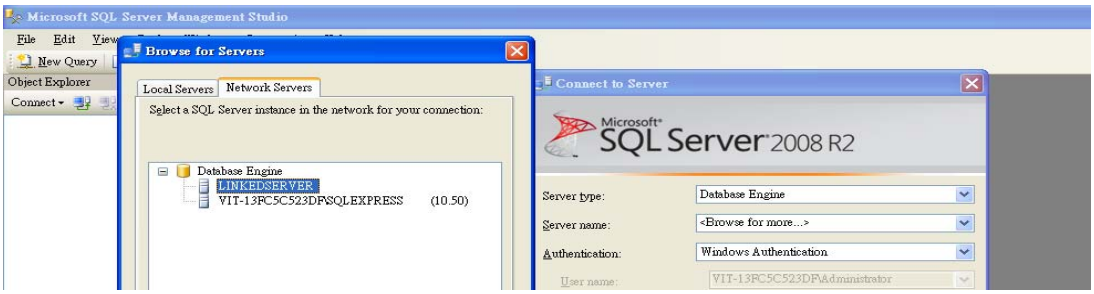

Figure 7.3 : Browsing the Database on the Other Server

Typing the Transact-SQL can add linked server successfully if the above internet setting of SQL Server has been done. The bottom left window shows Linked server has created, and the bottom window shows the distributed query executes successful.

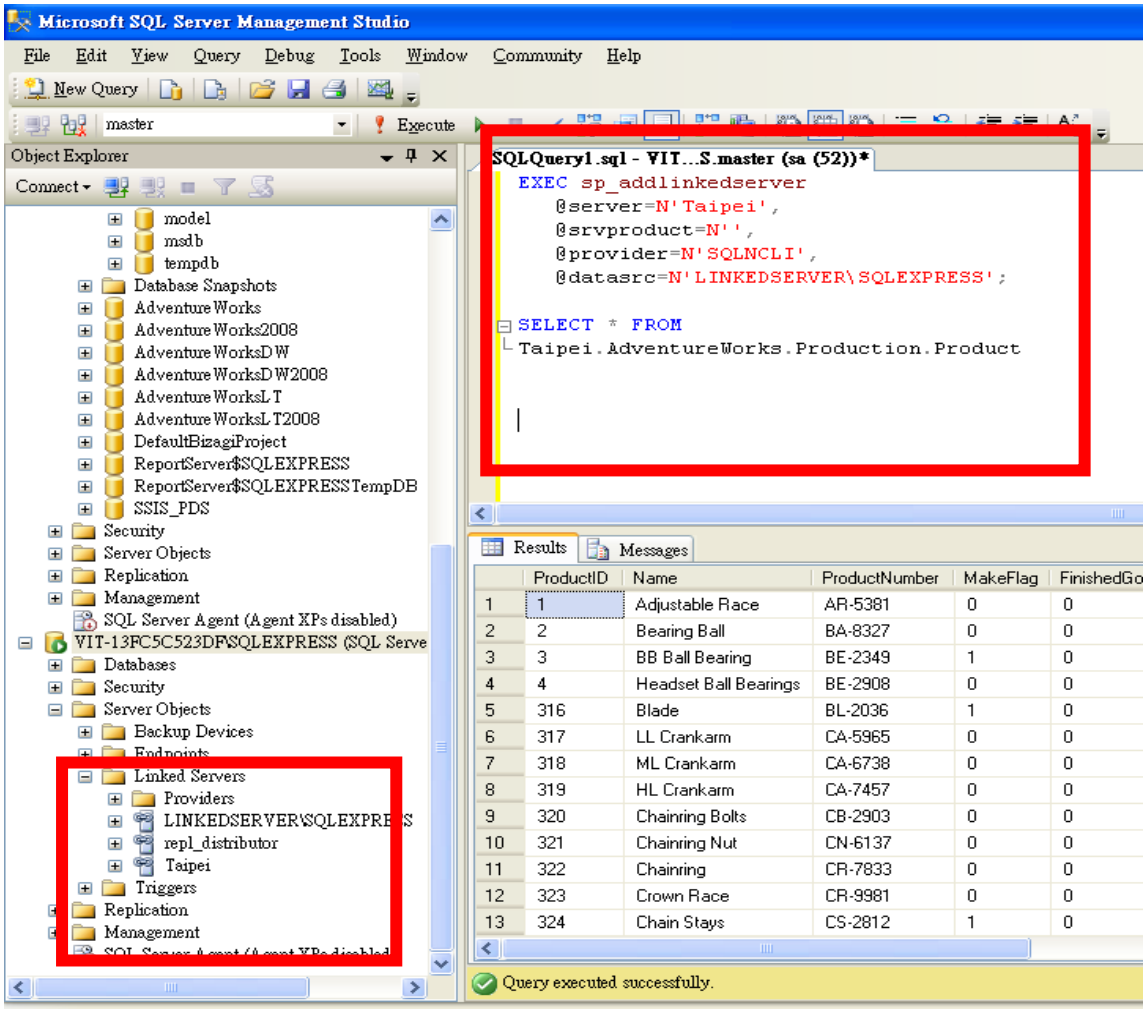

Figure 7.4 : Linked Server

7.2 Fail to Begin Distribute Transaction

To begin distributed transaction, it is still not enough. The following error massage shows the MS DTC is not activated. To start the MS DTC, Open START > SETTINGS > CONTROL PANEL > ADMINISTRATIVE TOOLS > SERVICES, and start the service called 'Distributed Transaction Coordinator'.

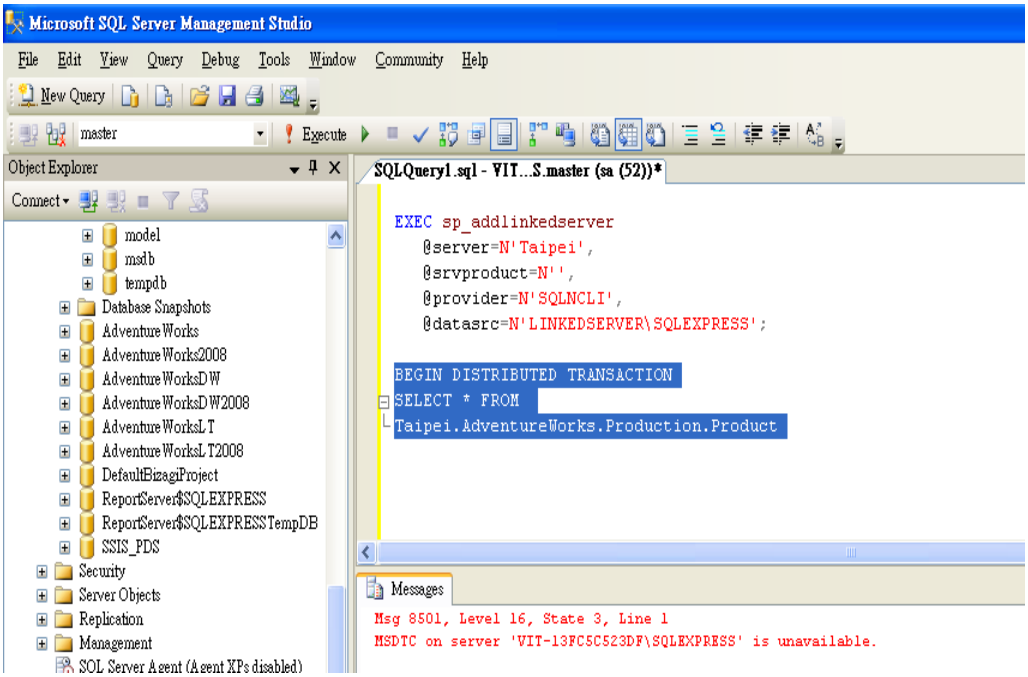

Figure 7.5 : Error of MS DTC

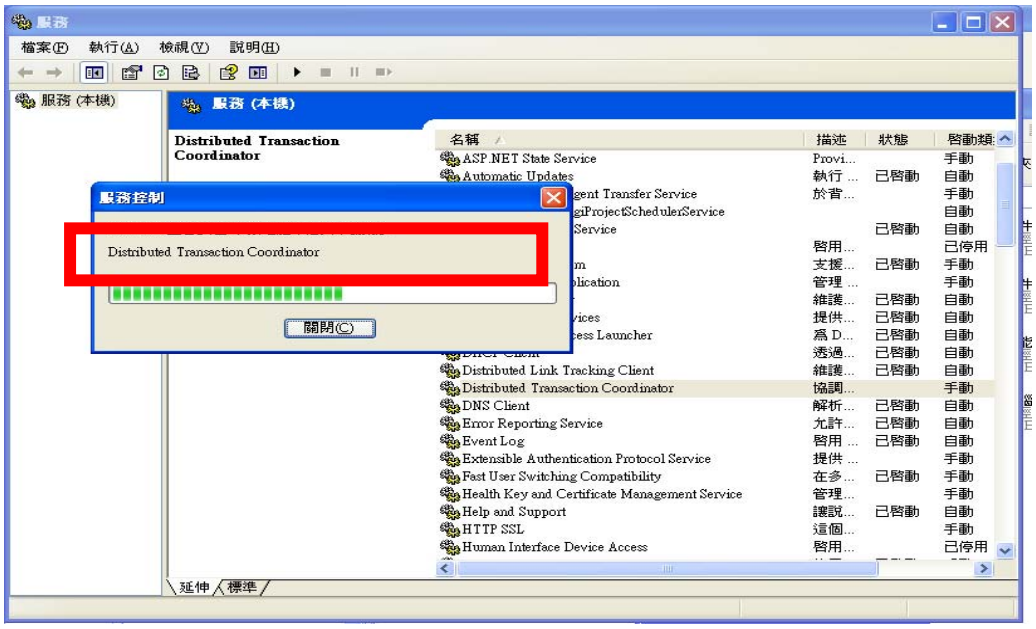

Figure 7.6 : MS DTC activates

8. Conclusion

The terminologies between SQL Server and the books are very confusing, for instance, Linked Server, Publisher and Subscriber. This implies originally each DBMS software doesn't provide distributed database, they add the features based on the Client/Server architecture. This also suggests no pure distributed database software exits.

On the other hand, SQL Server has the ability to implement a lot of promises of distributed database, such as distributed query, distributed transaction, fragmentation and replication, which can also set-up with Management Studio. The linked server provides the basic structure for distributed database. Based on linked server, the instances on SQL Server can execute distributed query, perform distributed transaction through MS DTC, and horizontal fragmentation, but the replication should set in the other way. Although the different terminologies between the book and SQL Server confuse a lot, it still provides the excellent way to implement the distributed database.

9. Reference

[1] G. R. Liang, "Incentive Driven Supply Chain," 2009.

[2] M. Tamer èOzsu and P. Valduriez, *Principles of Distributed Database Systems*: Springer, 2011.

[3] Microsoft. (2013). *MSDN*. Available: http://msdn.microsoft.com/en-US/

[4] J. A. Hoffer, R. Venkataraman, and H. Topi, *Modern Database Management, 11/E*: Prentice Hall, 2002.

[5] (2013). *Distributed Database*. Available: http://www.unife.it/ing/lm.infoauto/sistemi-informativi/allegati/