

Database Replication – an Overview of Replication Techniques in Common Database Systems

Patrik Rantanen

Bachelor's thesis
Degree Programme in Business
Information Technology
2010



Tietojenkäsittelyn koulutusohjelma

<p>Tekijä</p> <p>Patrik Rantanen</p>	<p>Ryhmä tai aloitusvuosi</p> <p>2007</p>
<p>Opinnäytetyön nimi</p> <p>Tietokantareplikointi – Yleiskuvaus replikointitekniikoiden toteutuksesta yleisimmissä tietokantajärjestelmissä</p>	<p>Sivu- ja liitesivumäärä</p> <p>59 + 27</p>
<p>Ohjaaja</p> <p>Martti Laiho</p>	
<p>Tämä opinnäytetyö on tehty HAAGA-HELIA:n tietokantateemaryhmän ja DBTech EXT projektin tilauksesta syksyllä 2010 HAAGA-HELIAssa järjestettävää workshopia varten. Opinnäytetyössä tutkittiin, mitä tietokantareplikointi on ja kuinka se toteutetaan.</p> <p>Tutkimustyö on jaettu kahteen osaan, jonka ensimmäinen osa on toteutettu kirjallisuustutkimuksena. Materiaali kirjallisuustutkimukseen on haettu eri tietokantatoimittajien manuaaleista, tutkimuksista ja oppaista. Näistä esitellään kolmen yleisimmän tietokantajärjestelmän replikointitekniikat ja niiden ominaisuudet. Vertailu on tehty ryhmittelemällä järjestelmät uudelleen toiminnallisuuksien mukaan. Toinen, empiirinen osa tutkii tietokantareplikointia käytännössä, ensimmäisestä osasta saatujen tietojen pohjalta. Lisäksi replikointiympäristön pystytys SQL Server 2008 ympäristöön on dokumentoitu.</p> <p>Kirjallinen osa osoitti, että eri toimittajilla on omanlaisensa toteutus replikoinnista. Näitä ovat erilaiset termit ja käsitteet, joiden vertailu oli hankalaa. Osan yhteenvedossa vertailu on tehty ryhmittelemällä ne uudelleen replikointityyppien ja ominaisuuksien mukaan. Ryhmät on myös uudelleen nimetty. Toisen osan testeistä saadun tiedon mukaan replikointi voidaan toteuttaa monella tavalla, ja se on tehokas tapa hajauttaa tietoa sinne, missä sitä tarvitaan. Lisäksi testit osoittivat suunnittelun tärkeyden.</p> <p>Opinnäytetyön johtopäätöksenä voidaan todeta tietokantareplikoinnin olevan monimutkainen järjestelmä, joka vaatii riittävää teoreettista ja käytännön osaamista niin suunnittelu, - toteutuskuin ylläpitovaiheessakin.</p>	
<p>Asiasanat</p> <p>Relaatiotietokannat, hajautetut järjestelmät, tietovarastot</p>	

Degree programme in Business Information Technology

<p>Author</p> <p>Patrik Rantanen</p>	<p>Group or year of entry</p> <p>2007</p>
<p>The title of thesis</p> <p>Database Replication – an Overview of Replication Techniques in Common Database Systems</p>	<p>Number of pages and appendices</p> <p>59 + 27</p>
<p>Supervisor</p> <p>Martti Laiho</p>	
<p>This thesis was requested by the HAAGA-HELIA thesis group on databases and the DBTech EXT project for a workshop where one of the topics is database replication. Its purpose was to test and document database replication techniques in most common database systems.</p> <p>The thesis was done in two parts. The first part was a literature research. It was constructed from vendor manuals, handbooks, studies and white papers. Additionally, re-grouping by functionality was done to help out differentiate replication features of vendor database systems. The second part, the empirical part consisted of series of tests, where database replication was tested in practice using gathered information from the first part. Furthermore, a setup of replication environment using SQL server 2008 was documented.</p> <p>The literature review showed that all database vendors have exclusive replication products. All vendors have their own concepts and terms, which made comparison of features challenging. In the recap of the first part, replication systems were compared by type and functionality rather than per vendor basis. Furthermore, groups were re-named. According to the empirical tests, database replication can be done in various ways and it proved to be effective system to distribute data where it's needed. Moreover, the tests showed the importance of planning.</p> <p>The study concludes that, due to the subject complexity and differences in vendor concepts, sufficient theoretical and empirical background is required to plan, implement and maintain replication systems.</p>	
<p>Key words</p> <p>Relational databases, distributed systems, data warehouses</p>	

Table of Contents

Glossary of terms	1
1 Introduction.....	3
1.1 Background and research problem.....	3
1.2 About the requestor.....	4
2 Database replication	5
2.1 What is replication?.....	5
2.2 Components of database replication.....	6
2.3 Replication environments	7
2.4 Common replication usage situations	8
3 Microsoft SQL Server replication system.....	9
3.1 Servers and their roles	10
3.2 Replication agents.....	11
3.3 SQL Server replication types	12
3.3.1 Snapshot replication.....	13
3.3.2 Transactional replication	13
3.3.3 Merge replication.....	14
4 IBM DB2 replication system.....	15
4.1 SQL Replication	15
4.2 Q Replication.....	17
5 Oracle replication solutions.....	19
5.1 Advanced replication	20
5.1.1 Materialized View replication	20
5.1.2 Multimaster replication.....	22
5.2 Oracle streams replication.....	23
6 Conclusion of part I	26
7 Part II – Replication examples.....	29
7.1 Replication test environment.....	29
7.2 Test network setup.....	30
7.3 Snapshot replication test scenario.....	31
7.3.1 Configuring the Distributor.....	32
7.3.2 Configuring the Publisher	34
7.3.3 Configuring subscribers.....	36
7.4 Transactional and merge test scenario configurations.....	38

7.5	Snapshot, transactional and merge replication test run	40
7.6	Transactional peer-to-peer replication configuration.....	49
7.6.1	Transactional peer-to-peer replication test	52
8	Discussion	55
9	Conclusion	55
	Bibliography	57
	Index	59
	Appendices	60
	Appendix 1. Description of DBTech EXT project	60
	Appendix 2. BANK database script	61
	Appendix 3. BANK data generation script	63
	Appendix 4. Distributor script	64
	Appendix 5. Snapshot publication script.....	65
	Appendix 6. Snapshot subscriber ARLANDA script.....	66
	Appendix 7. Snapshot subscriber SCHIPHOL script	67
	Appendix 8. Transactional publication script.	69
	Appendix 9. Transactional subscriber SCHIPHOL script.....	70
	Appendix 10. Transactional subscriber ARLANDA script	71
	Appendix 11. Updatable transactional publication script.....	72
	Appendix 12. Updatable transactional subscription ARLANDA script.....	74
	Appendix 13. Updatable transactional subscription SCHIPHOL script	75
	Appendix 14. Merge publication script.....	76
	Appendix 15. Merge subscriber ARLANDA script.....	78
	Appendix 16. Merge subscriber SCHIPHOL script	79
	Appendix 17. Merge subscriber MALPENSA script.....	80
	Appendix 18. Bank database backup script.....	81
	Appendix 19. Bank database restore script	82
	Appendix 20. Distributor and publisher MALPENSA script	83
	Appendix 21. Distributor and Publisher SCHIPHOL script	84
	Appendix 22. Distributor and publisher BARAJAS script	85
	Appendix 23. Peer-to-peer publication MALPENSA script	86

Glossary of terms

AD	Active Directory
API	Application programming interface
ASNCLP	ASN Command line processor used with IBM DB2 replication
BI	Business Intelligence
CCD table	Consistent change table
CD table	Change data tables are used in DB2 SQL replication by capture and apply programs. Also known as staging table
DBA	Database administrator. A person who maintains database operations under the guidelines provided by vendor and/or employer
DBMS	Database Management System
DDL	Data Definition language
DML	Data Manipulation language
FTP	File Transfer Protocol
LCR	Logical Change Record. Used in Oracle Streams replication
ODBC	Open Database Connectivity. A software API, independent of programming languages, operating systems and DBMS
OLAP	Online analytical processing
OLTP	Online transaction processing

MSDTC	Microsoft Distributed Transaction Coordinator. A service component in Microsoft operating systems
RDBMS	Relational Database Management System
RPC	Remote Procedure Call
SCN	System Change Number
T-SQL	Transact-SQL. A proprietary extension to standard SQL

1 Introduction

In a modern world the amount of information is increasing constantly. Governments and businesses are compiling their documents, invoices, registries, catalogues and records in electronic form. As business is becoming globalized, so is information they own and share. Data has to be available anywhere in the world at any time. Cost for such network solution would be significant. Traditionally lack of network bandwidth and cost has been the main reason for database replication. Spanning data across geographic locations allows it to be closer to the users, therefore providing gains in performance, reliability and lack of network congestion. However, networks and processors are getting faster and more capable, thus shifting the bottleneck towards the database management software and the operating systems. Databases and their performance, availability, response times and throughput are consequently facing new and challenging requirements. Since network bandwidth has increased and costs reduced, also applications that use replication have changed. It is still feasible to use replicas to localize network access, but what have become increasingly important are custom applications. Therefore replication solutions and the reasons behind are diverse.

1.1 Background and research problem

There are techniques to distribute databases for faster access and to reduce network and server loads. Replication is one, often used and many times not properly understood by the people who study databases. Even those who administer databases may find the terms controversial and complex. This is to a certain extent due to database vendor differences in terms and concepts, as well as documentation which is sometimes complex. Purpose of this study is to examine and document some of the many features of replication in most common databases, Microsoft SQL Server, IBM DB2 and Oracle. It seeks to answer the main question “what is database replication?” The sub questions are “what should a DBA know about replication?”, “what is the best practice of planning and implementation?” and “what are the benefits of using replication?” Further objective was to test in practice, how replication is done in the most common replication scenarios and types. By no means is it a complete guide, but an overview and one-off compiled document from abundant of sources.

Thesis has been divided into two parts. First part was carried out as literature research from vendor manuals, tutorials, IT- field books as well as whitepapers and other studies done in Universities, corporations and their collaboration. Second part, the empirical part of the thesis was done according to the request of DBTechNet. DBTechNet organized a workshop and a

seminar in autumn 2010 at HAAGA-HELIA University of Applied Sciences in Helsinki, Finland. One of the subjects discussed in the seminar was replication. A thesis was requested as a baseline to be used in the workshop.

Interest for the topic arose during a team project, which was completed in autumn 2008. It was part of a course combination specializing on databases. In the project, replication was done as an additional task. As a DBA in the making I had hoped for a subject to my thesis, which concentrated in a specific area of database administration. Something I could really get deeply into and gain substantial knowledge about. Many DBA tasks have been automated these days, but there is still plenty of work administering databases by hand. Implementing and maintaining database replication requires specific knowledge and if possible experience as well. Thus when this topic was proposed to me, I quickly decided this to be the subject for my thesis.

1.2 About the requestor

DBTechNet, the European database teacher network, is an initiative set up by European database lecturers in collaboration with Universities and IT-companies. It was founded in 1997 and funded by the EU Leonardo Da Vinci Programme. One of the key drivers for the creation of DBTechNet has been the lack of database professionals in the European labor market. DBTechNet aims to help higher level educational establishments to promote database teaching. It has three achieving goals, development of internet based tools with training materials and references, design and develop virtual workshop type course modules and promote entrepreneurship. They have had numerous activities with guest lecturers during the first years of its existence.

DBTech EXT is a continuation to DBTech PRO project, which was also a European Commission funded project. PRO project held many workshops covering many important topics during its run from 2002 to 2005.

2 Database replication

2.1 What is replication?

Semantic definition of replication is copying. Database replication also means copying, a replica presentation of the original data stored in a database. Its purpose is to propagate database objects to replicating server to share the load of a centralized database. A replica can consist of many common database objects, namely tables, indices, triggers, views and functions etc. depending on maker and their definition of objects. Database replication means that the same data is available at multiple locations. (Oracle 2009a.) This data can be read-only or it can be updatable or it can be combination of both. Furthermore, the data does not need to be complete copy of the original. A subset of a relation can be replicated instead. It is also possible to copy combination of relations to target table with aggregate functions (IBM 2009). Even though replication uses similar technique as distributed database and is distributed database as such, database replication is not and should not be confused with distributed databases (Oracle 2009a). Distributed databases may be subjected to a different study. Likewise, mirroring and clustering techniques, albeit being distributed systems and contain close associations to database replication, are left out of this thesis.

The benefits of replicating data over to different servers are many and varied but the main reason replication is done is performance as Wiesmann (2000, 11) describe. By providing local access to servers, access to data is distributed to number of servers instead of centralized server. On the other hand, Bernstein says (1987, 265) the main reason for replication is to increase availability. The thought behind this is that the more copies there are, more likely a copy of the database is found in case of a failure. A server-to-server scenario is typically used for this type of availability, for instance when data is gathered from several sites for warehousing (Thomas & McLean 2006, 526). Reliability that replication provides, means that if a site goes down, a replica can provide the same data to the user or application. Other clear benefits are network costs. By putting replicas in diverse geographic locations, closer to the user, network response times and network loads are reduced. Also it means that when load of the servers is shared, server load and response times reduce. Furthermore, replication provides opportunity for local custom applications when a subset of a data is used for local purposes. (Connolly & Begg 2005, 782.)

Downside of replication is the reality that it introduces more complex environment. A replication environment could be different in every location and the servers might not be managed

by same administrators. During updates, whether transactional or scheduled, access to database can be reduced. Also, recovery and data concurrency are more complex than that of a single server environment. (Connolly & Begg 2005, 781-783.)

2.2 Components of database replication

In any DBMS, when a replication is implemented, several programs are started and they work together to form the replication system. Replication is done between the source and target data, or so called master and slave. In the middle, there needs to be a way to distinguish the data and a way to communicate this information to all the parties involved in replication process. Conflict prevention, resolving mechanisms, agents, schedulers, transaction manager and service programs use history logs, transaction logs, system log and replication protocols are used to prevent data corruption. This is the metadata that is used by the programs handling all replications tasks. An example in Figure 1 is a replication system of Postgres Plus Advanced server, which is based on open source developed database.

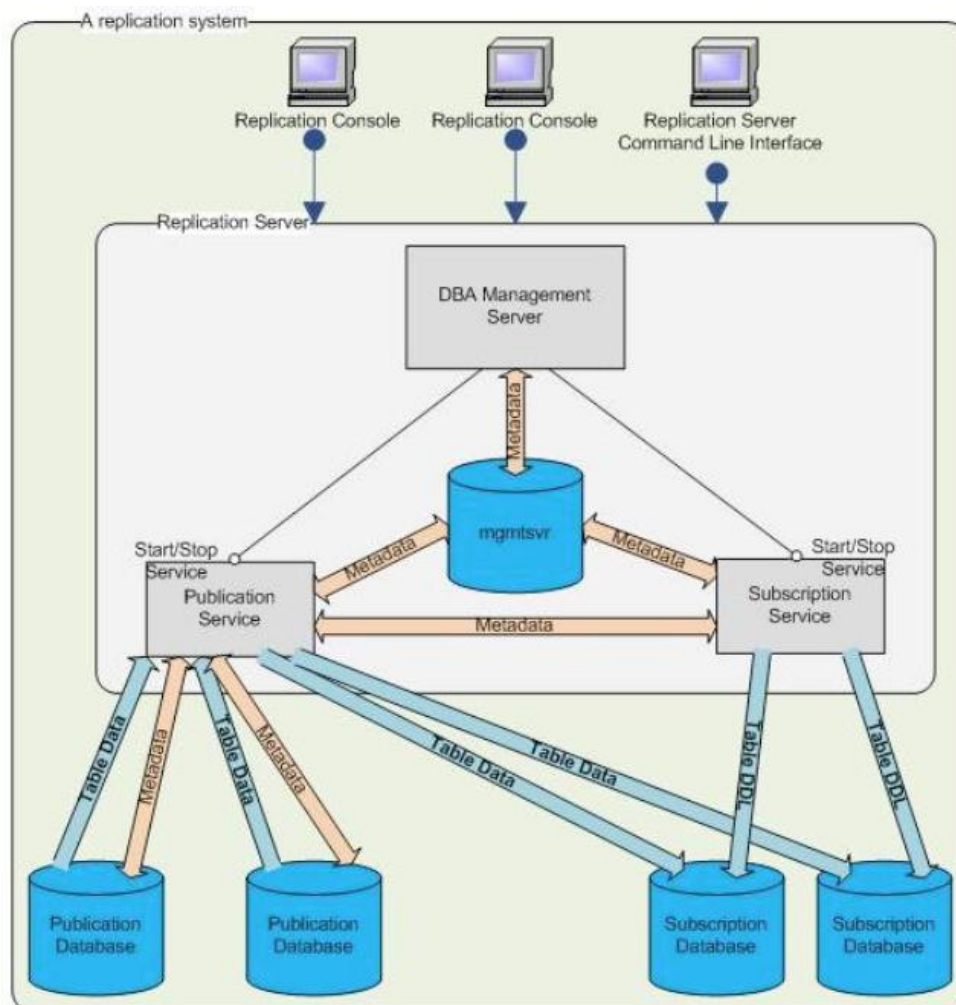


Figure 1. A replication system logical model

2.3 Replication environments

Synchronous replication

In a replication environment, a number of database copies are expected to behave like a one database. That is, when one copy is updated, all other copies need to be updated as well. This is known as synchronous replication. By means of locking protocols a conflict can be avoided and 1-copy serializability is guaranteed. (Wiesmann et. al 2000, 14.) Synchronous replication comes with a cost but it guarantees that all copies have the same data. In write-all approach (Bernstein et. al, 1987, 266), if a transaction tries to write to all copies of replicated system and one copy is not available, the transaction can't commit. This doesn't provide availability and prevents replication occurring. To avoid situation like this, a technique called voting is used. When voting is used, the majority of copies must be able to write and read. For instance, 7 out 10 copies must be able to write and 4 read. Each copy gets a version number for a transaction. The highest version number is the most current and therefore other will use this copy for update. (Ramakrishnan & Gehrke 2002, 620.) Even such, synchronous replication is used mostly in situation where replicas are close, and network bandwidth and availability is guaranteed like database mirroring techniques or failover clusters (Microsoft 2009). According to Oracle, in synchronously linked multimaster replication, "there is never a period of time when the data at any of the master sites does not match". (Oracle 2009).

Asynchronous replication

Replicating data continuously is not feasible at all times. The mechanics of synchronous replication might prevent execution of an update, leaving rows locked for a period of time. Asynchronous replication might be the preferred solution. (Ramakrishnan & Gehrke 2002, 621.) With asynchronous replication in a multimaster of peer-to-peer environment, an update is done in one database, and when it is committed the transaction is propagated to other masters. This way it requires less networking and hardware resources than synchronous replication. A conflict might occur, as for a period of time, data might be different in different master sites. In Oracle for example, for transaction a remote procedure call is created by a trigger and it's placed on deferred transaction queue. On a scheduled or on-demand interval, the deferred transactions are propagated to destinations. At the destination server the transaction is applied and checked for conflict. If conflict is detected, Oracle has resolution methods available. (Oracle 2009.)

2.4 Common replication usage situations

Data distribution

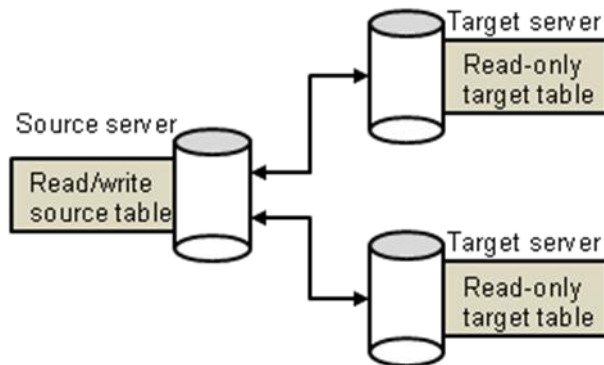


Figure 2. Data distribution (Gu et. al 2002, 4)

Businesses have many ways and needs for data distribution depending on situation and application. The most basic situation for replication is when a master production database is replicated to remote location as read-only copy. Read operations are most common and therefore makes sense to have a local replica of the master database. Transactions for insert, delete and update operations are always done on master data. Web applications might use master database in similar way as most of the transactions are reads. (Microsoft 2009.)

Another situation is when data is copied from the master server for use with an OLAP, BI or a reporting tool. This provides additional security and reduces load on the production server. (Gu et. al 2002, 3.)

Data consolidation

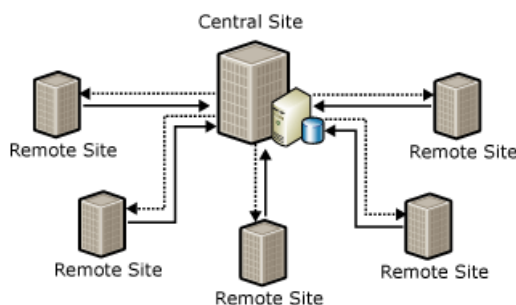


Figure 3. Data consolidation (Microsoft 2009)

Regional offices on a remote locations use read-only sales catalogues distributed from central master database. However, all offices send their collected business data back to central servers

for data warehousing. These include local transactions like sales and warehouse operations. Travelling salespeople carry an offline copy of the database and at the end of the day or week, synchronize transactions with central database. (Microsoft 2009.) Analysis and reporting tools gather data from many locations to central databases, where parts of the data is further distributed to tools and systems that use them, such as OLAP and data mining applications (Gu et. al 2002, 4).

Peer-to-peer

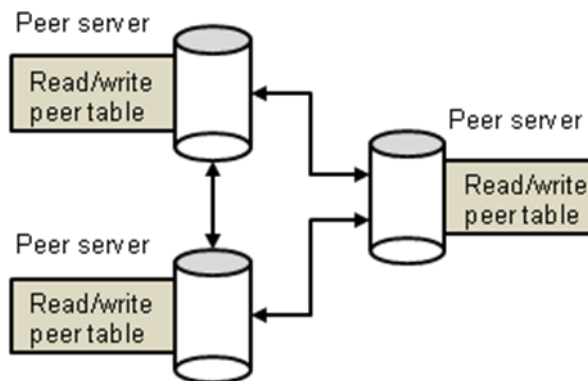


Figure 4. Peer-to-peer replication topology (Gu et. al 2002, 7)

A situation might turn up in a company, where centralized database might not be the best solution. Centralized database may be under heavy utilization load and no room available for fine tuning. For example part of production might be transferred to another country in a different continent, or a company has multinational presence where local data is needed. In such case data might still be location specific and is updated locally. As data communication networks are costly and they might not be upgradable in given situation, the solution might be transport of the database or parts of it instead. This type solution is called be peer-to-peer replication. Updates would be done on the geographically nearest location, but the data would still be available in any other location. (Oracle 2009; Microsoft 2009.)

3 Microsoft SQL Server replication system

In SQL Server replication environment, a terminology derived from the publishing industry is used. This magazine model as it is known, defines concepts such as a distributor, publisher and subscriber. Likewise, the publisher produces publications, which contain articles. Publisher may use the distributor or it can distribute itself. Publications are received by subscribers

according to their subscriptions. (Microsoft 2009.) The three servers outline the core of SQL Server replication. All components are needed to form a complete replication system.

Even though it may be helpful to understand replication with the magazine model, it may not be useful in real applications, for example in a real world the subscriber cannot update a publication. The components of the replication processes have different meaning to that of their magazine example counterparts and their association can be seen in Figure 5, where the information path is displayed. (Microsoft 2009.)

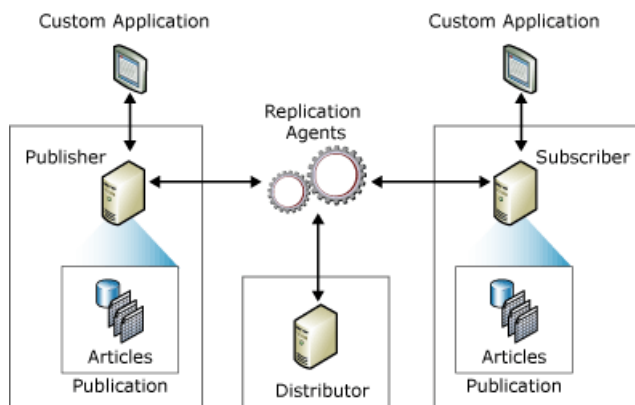


Figure 5. Replication system topology (Microsoft 2009)

Articles can consist of tables, views, stored procedures and other objects. Publication is a collection of articles. (Microsoft 2009.) It is a unit under same administration, similar to replication groups defined in other DBMS. As a group, articles are easier to maintain than individually. Articles are not available to subscribers by themselves, only through publications to which they subscribe (Thomas & McLean 2006, 532).

3.1 Servers and their roles

Publisher

Publisher is a server that holds the data that is to be replicated. It publishes information through publications, which are composed of database objects. These objects are known as articles. (Microsoft 2009.)

Distributor

Distributor's role as a server is to host exactly the same data as the publication in a distribution database, along with metadata about the publication. It interacts in between the publisher and the subscribers. Before data is sent to subscribers, it is picked up by the distributor. If the

distributor is located in a same database server instance, it's called local distributor. Likewise, distributor located in remote server instance is called remote distributor. If local publisher server is an OLTP server, it is better to use remote distributor. Since subscriber is involved in a two-step process, it also requires plenty of performance power. Likewise, as the distributor holds another copy of the publications, it is also important to take into consideration the disk space usage. (Microsoft 2009.)

Subscriber

Subscriber is a server receiving the publication. It can host replicated data from multiple publishers and publications. Subscriber can be another instance of SQL Server or another ODBC compliant database such as Oracle or IBM DB2. Two types of subscriptions exist, specifically push and pull. The difference between the two is usage situation. Whereas pull subscription is used in situations where changes are made at the publisher, push subscriptions are for real-time data movement. Another difference is the location of the agents. With push type, the distribution or merge agent runs at distributor server and with pull subscription it is running at subscriber server. Articles are database objects to be replicated. (Microsoft 2009.) In merge replication, according to Cotter (2009, 1), pull type subscription is considered less resource consuming and therefore better choice over low-capacity network links. Pull subscriber specifically requests updates according to its distribution agent schedule.

3.2 Replication agents

When replication is configured on the server, a number of background processes are enabled. These are stored procedures and standalone programs, which run controlled under the SQL Server Agent as scheduled jobs. (Microsoft 2009.) Replication agents as they are called, perform tasks according to their server location, specifically distributing data and tracking changes. Agents form the core of replication engine, which is separate of running SQL Server engine. (Mackin & Hotek 2007, 389.)

Snapshot agent

Snapshot agent runs at the distributor. It is used in all replication types initially. In distribution database, it creates a snapshot of data files of published tables and other objects. Also information about synchronization is stored by the snapshot agent. (Microsoft 2009.) Scheduling snapshot agent is an important part of planning a replication system. Generating snapshots can cause lack of disk space very quickly.

Log reader agent

Log reader agent also runs at the distributor. By reading publisher transaction log it differentiates transactions marked for replication and stores them to the distribution database sequentially as they were applied in the publisher. Log reader agent is used only in transactional replication. (Microsoft 2009; Mackin & Hotek 2007, 389.)

Distribution agent

Distribution agent runs at the subscriber for pull subscriptions and for push subscriptions it is ran at the distributor. Its task is to supply the initial snapshot. In case of transactional replication, it applies transactions to subscribers which were written to the distribution database by the log reader agent. (Microsoft 2009.)

Merge agent

Merge agent is used only with merge replication. Similar to distribution agent, it depends on subscription type for location. It sets the initial snapshot and applies data changes to the subscriber. For every subscriber a dedicated merge agent is needed. (Microsoft 2009.)

Queue reader agent

Queue reader agent is used with transactional replication where update option is chosen. It runs at distributor and manages changes made at subscriber back to publisher. Only one agent manages all publisher and publications as per distribution database. (Microsoft 2009.)

3.3 SQL Server replication types

Microsoft SQL Server supports three types of replication, snapshot replication, merge replication and transactional replication. When replication is considered for deployment, a special consideration has to be taken for selecting replication architecture. (Microsoft 2009.) If Data to be replicated is something that doesn't update frequently, there is no need for transactional replication. For example, a product catalogue deployed regionally might not be updated often enough to justify constant updates. A snapshot propagated to replicating servers after a change is sufficient. However, sales teams completing sales transactions need them to be committed immediately, therefore warranting a transactional replication.

3.3.1 Snapshot replication

Snapshot replication is in all replication types as initial base copy of the database to subscribers. A snapshot is a copy of a database in a single point of time. Entire snapshot is initially copied from the distributor to subscribers by the distribution agent. From thereon, the data in the published tables is refreshed periodically. This can be a long process as database snapshots can be large in size and can take lot of resources. Therefore careful planning and scheduling is in order. Generally, snapshot replication is used mostly in read-only data distribution setting, where data is updated infrequently. It is the simplest replication type SQL Server offers.

(Thomas & McLean 2006, 531.)

3.3.2 Transactional replication

Transactional replication, as explained in previous paragraph, typically starts out with base snapshot. It is one-directional flow from snapshot generated by the snapshot agent and distributed to subscribers. Afterwards transactions are gathered to the distribution database in batches by the log reader agent. Unlike snapshot replication, transactional replication transmits only incremental changes to subscribers, not whole snapshot (Ashok & Randall, 1). Data is either pushed from the distributor to the subscriber, or the subscriber pulls it from the distributor. If subscriber type is push subscriber, the distribution agent monitors changes at the distribution database and pushes those changes to subscriber. Pull subscriber works conversely, distribution agent runs at the subscriber and it pulls changes from the distribution database according to its own schedule. (Microsoft 2009.) Transactional replication means, that all transactions done on the publisher are immediately propagated to all subscribers.

Transactional replication supports updatable subscriptions. This is accomplished with the use of triggers. Two types of updatable subscriptions (see Figure 6) are available. Immediately updating subscriptions propagate changes directly to the publisher using MSDTC. A column is added to the publication table, which contains a version number for the transaction. At the subscriber, identity and timestamp columns are added. In case of queued updating subscriptions, the change is queued first and then applied to the publisher by the Queue reader agent. A system table is added to the subscription database which contains the changes. Transactional replication can be updated periodically or continuously. Typically it is used for read-only purposes, updatable configurations are not common. For example, a copy of a master database is kept updated with transactional replication just for backup purposes. (Mackin & Hotek 2007, 401-403.)

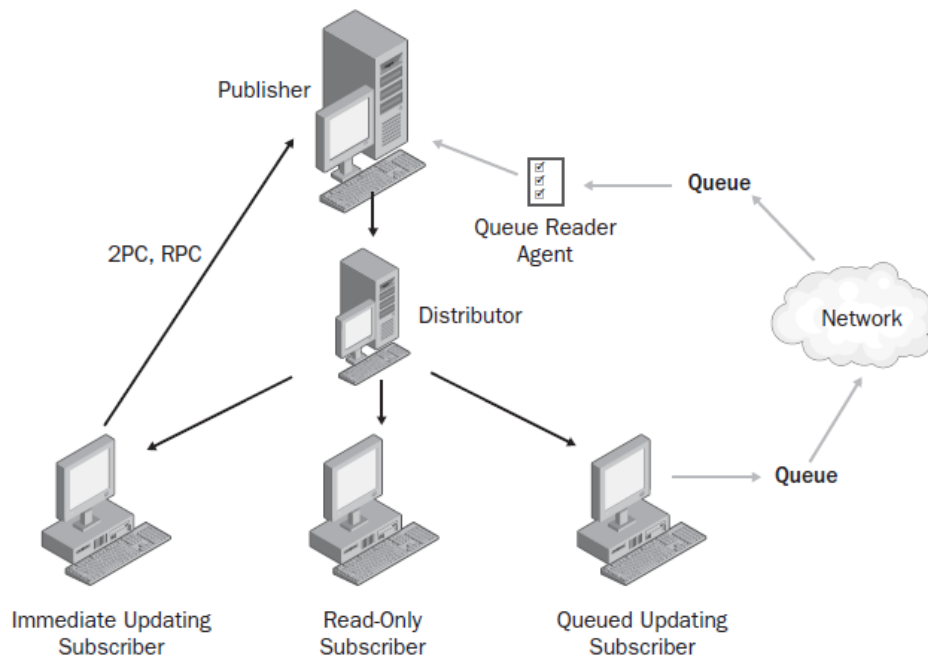


Figure 6. Transactional replication (Mackin & Hotek 2007, 402)

Transactional peer-to-peer replication

Transactional peer-to-peer replication is another implementation of transactional replication. It is used in situations where each peer is reading or updating data. This is accomplished by creating publication and subscribing it to a host using transactional replication. On the subscriber, a publication is created from the same data and replicating it back to origin using transactional replication. (Mackin & Hotek 2007, 404.)

3.3.3 Merge replication

Merge replication as its name implies, is designed to incorporate data from changes made at either publisher or subscriber (Nielsen et. al. 2009, 816). Most common use for this replication type, according to Mackin and Hotek is mobile and disconnect processing (2007, 392). After initial snapshot is generated and propagated to all parties, subsequent changes are tracked with triggers. In both publication and subscription databases, there are system tables which store metadata about transactions. (Microsoft 2009.)

Since merge replication allows concurrent updates at any location, there is a possibility of data conflict. Merge agent tracks conflicts from the stored metadata. If a conflict is detected the agent launches conflict resolver. Thus, the winning data is written to both subscriber and publisher (Thomas & McLean 2006, 553). Conflict resolve process is affected by the type of sub-

scription, whether it is a client or a server. This is specified when subscription is created. A priority value is assigned to subscription. For client type it is 0.00, but when change is synchronized the value of the publisher is assigned. For the server type it is assignable by the user and it ranges from 0.00 to 99.99. The top-level publisher always has priority value of 100. (Microsoft 2009.) When client types both update and get assigned with the publisher priority value, it means that first update wins, since both have identical values.

Conflicts are tracked into a special table called MSmerge_conflicts_info on the distribution database. In this table there's a column for conflict type. A total of 14 types are specified. When a conflict occurs, the data from the losing row is stored. (Thomas & McLean 2006, 551.)

Merge replication has a retention period for subscriptions. The default is 14 days. This is the time that within subscriber must be synchronized with the publisher or it has to be reinitialized. This is due the metadata being flushed by the clean-up process. (Microsoft 2009.)

4 IBM DB2 replication system

In DB2 database system there are two types of replication available, SQL replication and Q replication. SQL replication is a feature included in all DB2 editions. Q replication is only available for DB2 Enterprise Server Edition. Q replication is a new type and it is used for replication of large volumes of data with low latency. Both are managed by the InfoSphere Replication Server. Additionally, there's a feature called Event Publishing, which converts changes in tables or events to XML or comma delimited messages. These are then fed to the Websphere MQ message queue for use with applications or API's. (IBM 2009.) This feature is somewhat similar to replication and also uses Websphere MQ for its message delivery, but Event Publishing is not further discussed in this thesis. Unlike SQL Server, there's no magazine model or similar to illustrate the replication process. Terminology is different compared to the other DBMS, as are the fundamentals in DB2 replication system. However, applying targets to source is known as subscribing which is a term, used for such process in SQL Server.

4.1 SQL Replication

SQL replication allows data source to be DB2 or non-DB2 source such as SQL Server, Oracle or Sybase. A Source can be a subset of a table data, a view of single table or it can be a view

created from inner join of table group. Source tables are selected for replication in a process called registering. Information of the registered sources is stored in capture control tables. Control tables are created in replication center tool or with ASNCLP command line program. Usually, capture control tables are stored at the source server. Likewise, apply control tables reside at the target server. When tables are registered, a method for replication is also specified. There are three methods for table replication. (IBM 2009.)

Full-refresh replication

Full-refresh replication method does not capture changes made to tables. Instead it is invoked by specified intervals. Data is not captured and therefore does not involve capture program. The apply program reads the registration information from the control tables, deletes the contents of targets and replaces it with content of registered tables. Since capture program is not started it is also a performance benefit. (Gu et. al 2002, 144; IBM 2009.)

Change-capture replication

Change-capture replication method type1 is used when columns are selected as registered columns for replication. This method reduces the amount of data to be replicated. All changes to one or more of the registered columns are captured. If a change occurs in a column that is not registered, no data is captured. Change-capture replication method type 2 is similar to type 1 with the exception that changes to any of the columns are captured. If a change occurs in any of the registered columns of a table, the change is captured. However, this is only for the row where it occurred and only for the registered column value of that specific row. (Gu et. al 2002, 143; IBM 2009.)

Subscription sets

Whenever source-target pair is formed, it becomes a member of subscription set. These subscription sets are used by the named apply program which can process one or more subscription sets. There are rules which apply to replication of each subscription set. Subscription member sources must be on the same server. Targets of a subscription set member must also locate on same server. Replication timing depends on selected method and is selected on subscription set basis. Continuous replication lets apply program replicate as frequently as it can depending on load. (IBM 2009.)

Target types

Target table can be different type than the source. For user copy table the whole or a subset of the original can be created. It requires primary key or unique index column to uniquely describe each row. A point-in-time target adds a timestamp column that indicates the time the data was committed. Aggregate target is a computed value of the sources selected. It uses functions such as AVG, COUNT, MIN, MAX and SUM. Timestamp is also added for tracking changes. CCD table contains columns for further tracking. By default these columns include a sequence number that ascend when transaction takes place, a column for operation type (INSERT, UPDATE, DELETE), timestamp and a sequence for each row in transaction. Other columns are also available. All of these abovementioned targets are read-only type. A replica type can be used with update-anywhere replication and it allows writes. In this case, the source acts as a master table and if conflict occurs the replica will be rejected. (IBM 2009.)

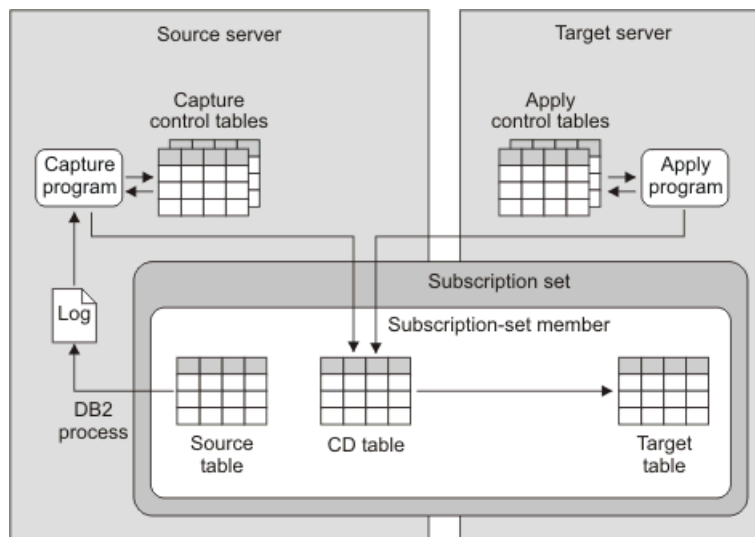


Figure 7. DB2 SQL replication process (IBM 2009)

4.2 Q Replication

Q replication is essentially different from SQL replication. Data is not stored in staging tables like in SQL replication, nor is it sent as data over to replicas. Transactional, committed data is read from the recovery log by the Q Capture program, converted and sent over to Websphere MQ as a message. Likewise, the messages are converted to transactional data at the target or sent to application as is. Q replication uses control tables in a similar way as they are used in SQL replication. Both the capture and apply store information of the Q subscriptions, used queue, queue manager and tables to apply transactions. Dissimilar to SQL replication, no sources are registered for subscription. A Q subscription is used to map a source to a single target or multiple targets by creating own subscription to each. Replication queue map is used to group Q subscriptions. (IBM 2009.)

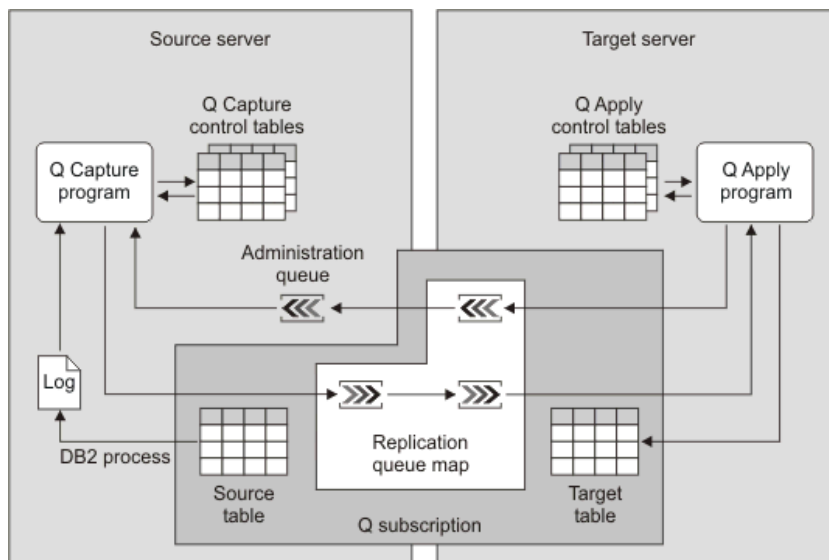


Figure 8. DB2 Q replication (IBM 2009)

Q capture process

When Q capture program reads and detects a change in recovery log, it adds the transaction into its memory. After COMMIT statement, transaction is converted to message and put into send queue. A replication queue map is used to differentiate queues. Multiple instances of capture program can be run simultaneously. They have their own schema and control tables. The benefit of running parallel program is higher throughput. Consequently, it requires more database connections. (IBM 2009.)

Q apply process

The apply program receives messages from the capture program in its receive queue. Messages are then converted to SQL by the apply program. If transactions are not depending on one another, they can be applied concurrently by the apply program due its multithreading capabilities. Usually one apply program is used per server even with multiple queues or with high volume transactions, but it's not restricted. The apply program can have multiple instances which are separated by schemas. This way data can be processed differently from other receiving queues. (IBM 2009.)

Q replication types

Q replication supports three different types of replication. For replication between two servers, there's unidirectional and bidirectional replication types, used depending on situation. The differences are verbatim. However, in two-server scenario where both parties need to update data, a choice can be made between peer-to-peer or bi-directional replication types. Peer-to-

peer works best if conflicts are expected to happen, since it uses column versioning. (IBM 2009.)

Unidirectional replication

Changes made to source are replicated to target. Tables at the target are read-only. Replication from target back to source is not possible. Any rows and columns or a subset of them can be replicated. Q subscriptions allowed are one source to one target, one source to multiple targets, multiple sources to one target and multiple sources to multiple targets. (IBM 2009.)

Birectional replication

As its name implies, bidirectional replication is done between two servers. Source tables identified for replication have same structure at the target server. Any changes made to either of the tables on any server are replicated vice versa. A table can be configured to take precedence if a conflict occurs during update. (IBM 2009.)

Peer-to-peer replication

Peer-to-peer replication can be applied to two or more servers. All servers involved have identical table structure. That is they must have same number of columns, column names and data types which can have different names. All changes to a table on any server are replicated to all parties in peer-to-peer replication environment. Column versions and triggers are used for conflict detection and resolving. (IBM 2009.)

5 Oracle replication solutions

Oracle has many replication options available. The simplest way to get data replicated is to use Oracle basic replication. As its name implies, it is really straightforward solution. Basic replication is always done with one-way read-only snapshots. Only data is possible to replicate with basic replication, not objects such as procedures or indexes. (Orafaq 2009.) For read-only query access is local and updates are always done on the master table. Similar to Basic replication, Materialized view replication is improved over basic replication with updatable views, and is also based on snapshots. Materialized view is part of Oracle Advanced Replication. Other replication types are Streams replication, TimesTen replication and after acquisition of GoldenGate yet another replication solution. TimesTen and GoldenGate are not discussed in this study further. (Oracle 2009a.)

5.1 Advanced replication

Oracle advanced replication comes integrated with Oracle database server: It is not a separate feature. It is constructed from different components that make up the replication system. The concepts are defined for Oracle replication. Hence they differ to that of their rival products. (Oracle 2009a.)

Replication objects are database objects, which are copied to other site when an update is made. Oracle objects consist of tables, including partitioned tables, views and object views, packages and package bodies, procedures, functions, user defined types and type bodies, triggers, synonyms, indexes and index types and user defined operators. (Oracle 2009a.) It's not possible to replicate sequences, as they need to be generated on each site. For LONG and LONG RAW a LOB type should be used instead as Oracle advanced replication doesn't support these data types. (Orafaq 2009.)

For easy administration purposes, replication objects are pooled within replication groups. An object can be a member of only one replication group. Every site has also a replication catalog, which consist of administrative information about replication groups and objects. (Oracle 2009a.)

Replication groups belong to replication sites. There are two types of replication sites, master sites and materialized view sites. Replication group that belongs to master site is called master group and group belonging to materialized view site is called materialized view group. For every master group there's one master definition site. This is the control center that manages all replication groups and objects. There are restrictions however. If a site is a master site for a replication group, it cannot be materialized view site for the same group. (Oracle 2009a.) This means that all master sites, despite it being a single or multimaster site, holds complete data of replicated objects. In materialized view sites only a part of the data may be held, moreover it can be different subset in another materialized view site.

5.1.1 Materialized View replication

Known also as a snapshot replication in the past, a materialized view contains partial or complete copy of the master database in a single point of time. A materialized view can be based on a master materialized view, which is known as multitier materialized view. There are benefits for using materialized views over master database. Queries are local and therefore faster.

This also takes the load of from the master database. Security is another benefit. A master database is safe as only selected subset might be replicated. With read-only type conflicts are not possible since updates are only done at the master. (Oracle 2009a.)

Read-only materialized view replication

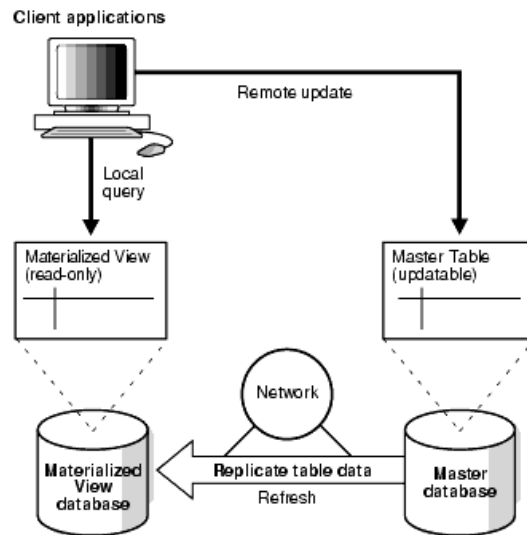


Figure 9. Oracle read-only materialized view replication (Oracle 2009a)

Local read-only access is given to client applications to a materialized view of the master data-base. In this configuration if application needs to update a table, it has to do it on the master database. Materialized view is updated from master with scheduled refreshes. This reduces network load and can be scheduled to low-traffic periods. However, during refreshes, users cannot make queries or DML statements. (Oracle 2009a.)

Updatable materialized view replication

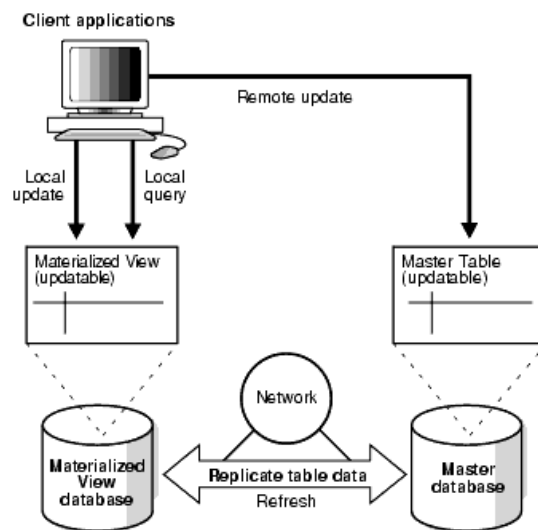


Figure 10. Updatable materialized view replication (Oracle 2009a)

A materialized view can be made updatable. This allows INSERT, UPDATE and DELETE operations of the master table to be performed at the materialized view. During refreshes, changes made in materialized view are applied to the master. In rare cases, materialized views can also be made writable, but this is not common. Writable materialized views can't be a part of a materialized view group. (Oracle 2009a.)

5.1.2 Multimaster replication

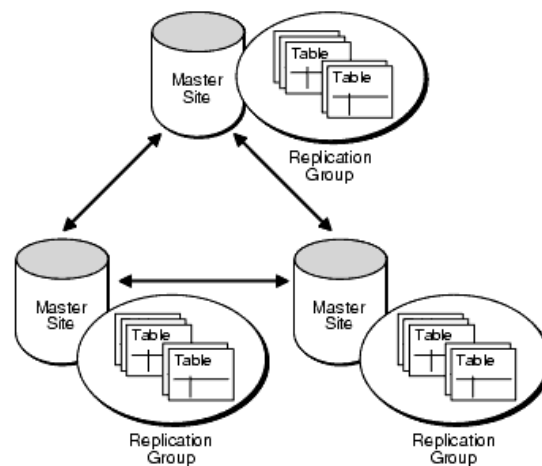


Figure 11. Oracle multimaster replication logical model (Oracle 2009a)

Multimaster replication, which is also known as peer-to-peer or n-way replication, consist of multiple sites. These master sites have two way communications amongst the whole multimaster group in an update anywhere model. When update occurs, it is propagated to every master

site equally as portrayed in Figure 11. Every master site has a copy of each replicated table. One master site acts as a master definition site in master group. (Oracle 2009a.)

Oracle (2009a) defines multimaster replication to be useful in balancing heavy application load. Each master site provides local update access point to the database. Master sites support disconnected materialized views. For example a salesperson can store a subset into a laptop or other type of disconnect device. (Oracle 2009a.)

Multimaster replication process is of either type, synchronous or asynchronous. It affects the way information is processed. Asynchronous mode also called store and forward, propagates changes in regular intervals. This means that for a period of time, data is inconsistent. Synchronous means that data is applied real time, as all masters involved in the transaction must commit, or the whole transaction is rolled back. In multimaster replication, it is possible to have environment where mixed modes are used. Some sites propagate synchronously and some asynchronously, all within one master group. (Oracle 2009a.)

5.2 Oracle streams replication

Oracle has many replication solutions. One of them is Oracle streams replication. It was developed to meet the business needs of growing information, which require multiple databases and multiple solutions. Each solution targeted a specific problem, thus adding complexity of information systems. Oracle Streams aims to simplify these tasks by providing a unified solution to information sharing. Streams captures events, updates and messages from applications. These are then distributed for use with replication and, event notification, data warehouse loading and many other applications. (Oracle 2005.)

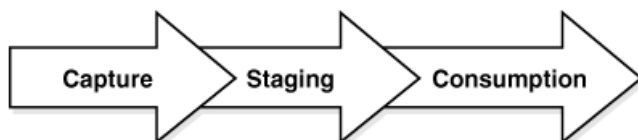


Figure 12. Oracle Streams elements

In Streams, an information unit is called a message. These messages are shared in streams, which are propagated within a database or to another database. (Oracle 2009b.) Oracle streams is based on three elements, capture, staging and consumption. These elements are controlled by the Oracle rules engine. (Oracle 2005.) This is somewhat similar to DB2 Q rep-

lication as it changes messages rather than data directly. Again, processes that handle messages are known as capture and apply, yet another resemblance to DB2.

Event capture

Streams captures events into so called staging area in two ways, explicitly or implicitly. Explicit capture is used by applications which create the events and puts them in staging area. Implicit capture on the other hand has two distinguish capture methods. Capture process catches changes by reading the redo log. These changes are converted to special messages, captured LCR's. Synchronous capture messages are called persistent LCR's and they are captured and formatted to messages using internal technique. The difference between the two is the rules, which tell where and which changes are captured. Capture process can capture changes made at remote database called a downstream database, or it can capture local changes. The information of the database is generated in redo log. Synchronous capture can only capture changes made where it was configured. LCR's or messages are then put in to queue. At regular intervals, capture process is recorded to a checkpoint. Checkpoints are identified with SCN numbers. Checkpoints are held for a period of time called checkpoint retention time, before they are automatically purged. Time is measured in days. Checkpoints are important for recovery. (Oracle 2009b.)

LCR's consist of information from the changed table. It stores new and old values of the columns that the change affects. Also, key column information is stored to correctly identify the rows in question. A single DML statement can generate multiple LCR's and one transaction usually consists of many DML statements. Therefore it is vital to differentiate LCR's origin. This is accomplished by using tags. Tags are used in consumption phase when LCR's are transferred to source queue to destination queue (Figure 13). (Oracle 2005.)

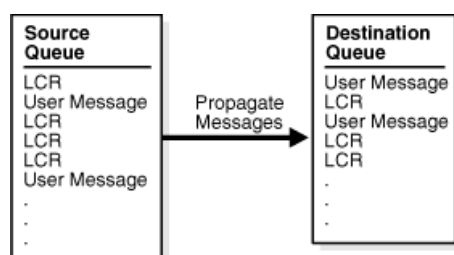


Figure 13. Streams queues (Oracle 2009)

Staging and propagation

Message queue, where LCR's and messages from the capture process and synchronous capture is put is called anydata queue. Those messages from applications are put into typed queue,

which can be only one specific type. Queues reside in memory until they are consumed by the subscribers. A subscriber can be an apply process, another staging area or a user application. In case the subscriber is an apply process, the messages are consumed by it. User applications dequeues messages from the staging area and consume them. (Oracle 2005.)

Events of other databases can be propagated in staging area to reduce network traffic, thus they do not need to be directly connected. Streams can pass events through by routing them. Routing can be specified by rules, a feature of publish and subscribe. (Oracle 2005.)

Consumption

Consuming in Streams means that messages are dequeued, consumed or discarded. This is done by the apply engine. It consist of several server processes, an apply coordinator, a reader server and apply servers. Like capture, the apply engine also has implicit and explicit options. Similar to its capture counterpart, implicit apply is automated process. It dequeues different types of LCR's from the anydata queue and applies it directly or passes it to a procedure. If LCR contains DML or DDL statements, they are automatically applied. The apply engine also has conflict detection mechanism. (Oracle 2005.)

6 Conclusion of part I

Foreword on literature review and comparison

After reading through the manuals, it was quickly discovered that if any comparison of the database systems was to be done, grouping them somehow was in order. Therefore with the influx of concepts and terms, it was not wise to recap the features with dissimilar information. The databases were grouped by functionality; they are easier to comprehend, along with common terms and concepts.

Summary of vendor documentation

Replication is complex system and requires careful planning in implementation phase. When in production, a DBA must monitor alerts, conflicts and performance. Even though lot of these tasks are automated, human eye can spot the alerts and fix things by hand. Automation can't yet do these tasks. Without waste knowledge implementing and maintaining replication is challenging. There are so many features, so many details to configure and rules to apply. The vendor manuals provide all these with examples.

SQL Server, for example has a good logical model that follows the flow of a newspaper publication. T-SQL has good stored procedures supporting replication and they are documented well in Books Online, which is the main source of information of SQL Server. It contains also tutorials and examples, which show some basics of replication. The concepts are logical and easy to understand, although at first they seemed too alike. The whole publisher model opens up after enough detail of the system is gained.

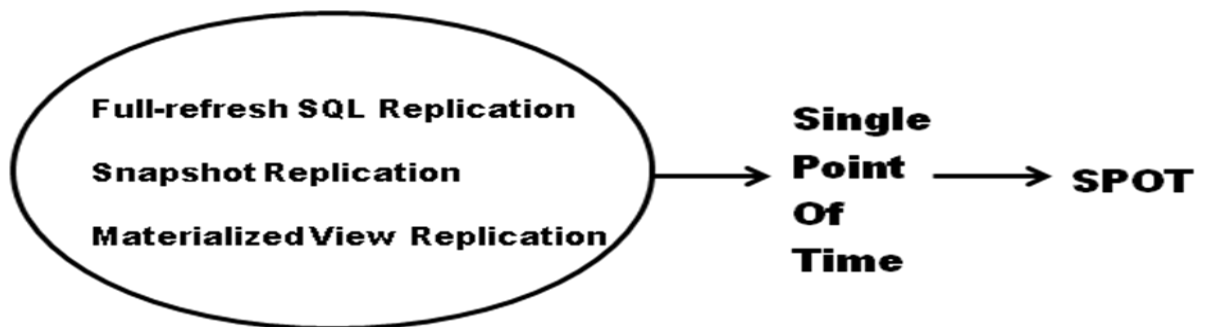
IBM documentation along with the Redbooks is also good way to discover and learn new things, such as replication. There are tutorials and examples, and all details are documented with vast amount of options. This is however also its weak point. There's so much of information, and so many things to tune and trim before the actual replication can take place. The concepts are specific for replication, yet complex and easy to forget. The idea of separating the "simple" SQL replication and the more complex Q replication is good. They serve different purpose and it's noted in documentation.

Oracle provides good documentation with pictures, examples and code to copy and edit for own purposes. Every detail is explained with much information. There's however so many terms and their derivatives that it is very complex and therefore hard to understand. Oracle has

also many replication options that surpass each other, maybe a cleanup of obsolete solutions would be in order.

Summary and comparison of the systems

First replication type; all vendors have this type of replication in some form. SQL server calls it Snapshot Replication. IBM has similar type with its SQL replication, more specifically Full-refresh SQL Replication. Oracle has renounced the name Snapshot, and uses the name Materialized View Replication instead. Anyhow, all these have one key thing in common. They represent data in a single point of time. Thus, 1st group is called SPOT type replication.



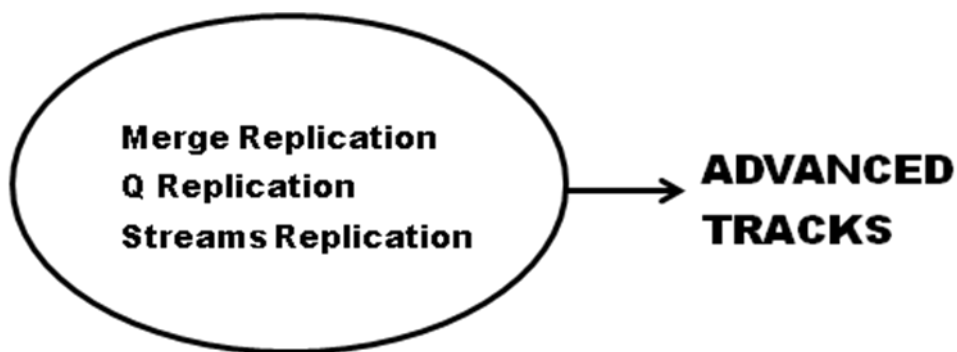
This type of replication is one-way data flow, master to copy replication. Updates are done on the master and a complete re-write of the existing data in replicating server is done periodically. This schedule is controllable and is the one thing along with disk space that requires careful planning.

2nd group involves many of the intermediate replication types. They share commonality in that they allow updatable copies. Or the updates are done only in master data and then propagated to the copy. All the types allow this decision to be made beforehand.



The name derives from the fact, that all types in this group track changes and propagate them to copies. Even more, all can do peer-to-peer type replication or updatable copies.

3rd group consist of the advanced replication types. Q replication and Oracle Streams are both message queue based, and have many features that are highly configurable. Merge replication is not similar in that it doesn't use message queues as such, but it does have sophisticated conflict detection and resolution system. Therefore it is included in this group.



This group is named as advanced tracks because all have advanced features. All types can do everything the previous group can more efficiently. All types support updates everywhere and have conflict detection. Additionally, all can be used in any situation with the exception of merge replication, which doesn't support peer-to-peer replication.

Comparing replication on paper is hard. All vendors appear to have solution for any kind of situation. Database replication explained in one sentence would be; consistent, atomic information that retains integrity, available in multiple locations at almost same time. As for the question "What should a DBA know about replication", most important things to know are what to replicate and what kind of replication type is required to accomplish that. These are the requirements in planning and implementation phase. Maintaining replication system also requires theoretical knowledge and experience. As for who benefits from replication? Many do, but most likely the consumers, employees, DBA's and the decision makers that need the data available at their beckon of call. Moreover, companies that reduce their network costs and gain markets by re-using data in new applications. After all, replication means copying.

7 Part II – Replication examples

In this part of the thesis, different replication types and scenarios were tested. All examples were composed in SQL Server 2008 environment. Throughout the examples the same setup was used. For sake consistency and simplicity, default settings were used where appropriate. All tests we're done using the DBTechNet BANK database. BANK database creation script is included in appendix 2 and data generation script in appendix 3. The data generation script creates by default 100 000 rows into the Account table, 10 rows into Branch table and 100 rows into Teller table. Other tables are used for DBTechNet TPCA benchmark program for testing performance, by measuring throughput of transactions.

7.1 Replication test environment

An example of configuring replication is given with the setup of snapshot replication. For other replication types, setup was mostly the same. When there is a difference for the specific type, it's noted and illustrated. Peer-to-peer replication is another example which requires many changes to the basic setup. Therefore, it's been introduced in its own.



Figure 14. BANK database logical diagram

The hardware and software setup used:

- 3 standard PC's with 1 Gigabytes of RAM memory + 1 virtual machine with 512Mbytes of RAM running in VMware player 3.0.0 build-203739
- Cisco Catalyst 3750 switch
- Microsoft Windows Server 2003 standard with SP2
- Microsoft SQL Server 2008 Developer edition SP1
- DBTechNet bank database

7.2 Test network setup

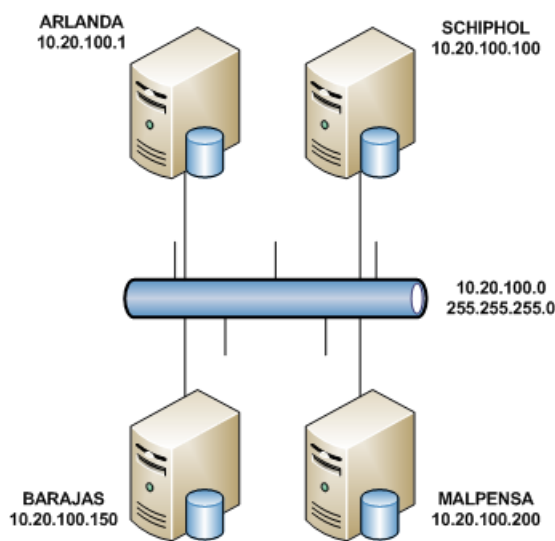


Figure 15. Test network environment

The network environment used for the tests was a simple LAN setup with Windows AD domain. All servers located in same network segment with 100Mbits/s full duplex access speed. Host name logic follows that of European airfield names and domain was named to ATC. All hosts have network names to IP-addresses mapping done on hosts-file.

SQL Servers were configured with same setup. Special username was added to AD administrators group, sqlRoot. This account was also added to local server administrators group to be used over the network shares and for different replication agents. For all SQL servers, mixed mode authentication with both windows and SQL were chosen as authentication method. The login sa was enabled. Windows firewall service was disabled to avoid problems with network security. Furthermore, all servers needed to have SQL server agent started and TCP/IP communication protocol enabled. For these tasks, SQL Server configuration manager was used.

The test scenario as such does not confirm good security practices recommended by Microsoft. This should be taken into consideration if this test is duplicated or used as a baseline. However, Microsoft recommends using a network share for storing snapshots for pull subscription type subscribers. This was part of the reason behind AD domain setup.

Additional steps were needed to get replication working. All SQL servers were configured to “SET REMOTE_PROC_TRANSACTIONS OFF” to solve the SQL server error 7391 and 3933. Another task was to set MSDTC to allow remote transactions connections, as explained in Microsoft support articles <http://support.microsoft.com/kb/839279> and <http://support.microsoft.com/kb/306212/>.

7.3 Snapshot replication test scenario

First test was the easiest and the most simple to setup, snapshot replication. For this test server MALPENSA was assigned to play the role of the publisher, BARAJAS was the remote distributor and servers ARLANDA and SCHIPHOL were assigned the roles of the subscribers with ARLANDA being pull subscriber and SCHIPHOL push subscriber. Changes to default names for the databases were introduced to comply with the BANK database. Therefore distribution database was named to distributionBANK, publication to publicationSnapBANK, subscription database to subscriptionSnapPullBANK for SCHIPHOL and subscriptionSnapPushBANK for ARLANDA. In Figure 16, the test setup logical plan is illustrated.

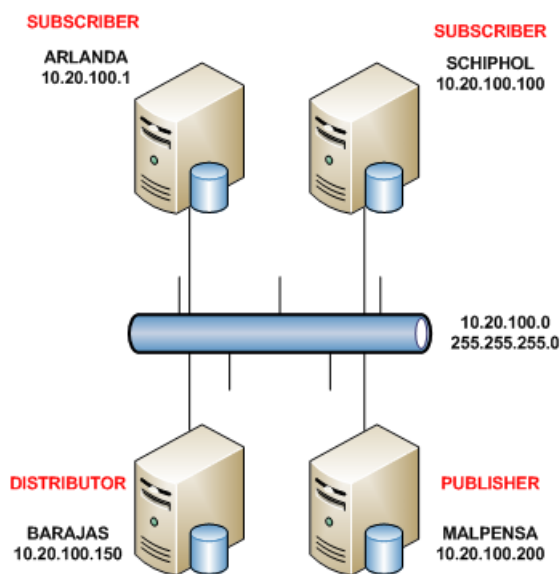


Figure 16. Snapshot replication test setup

7.3.1 Configuring the Distributor

All replication scenarios require a Distributor to exist before the actual replication can be configured. The distributor role was assigned to server BARAJAS. Wizards provided by the SQL Server were used in all configurations. The scripts generated by the wizard were added to appendices. The setup wizard starts by right-clicking the Replication folder and selecting option “Configure Distribution” in the object explorer in SQL Server Management Studio.

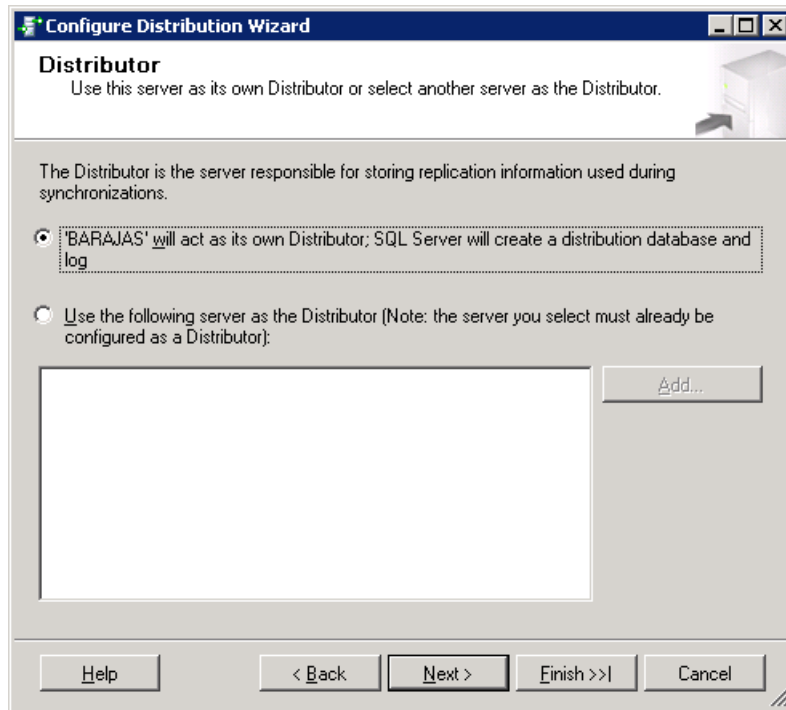


Figure 17. Distributor location selection

Step 1 was to select the Distributor location, in which the options were local or remote. Since this scenario was planned beforehand and configuring took place on server BARAJAS, the local Distributor was selected. From the Publisher point of view however, this is a remote distributor.

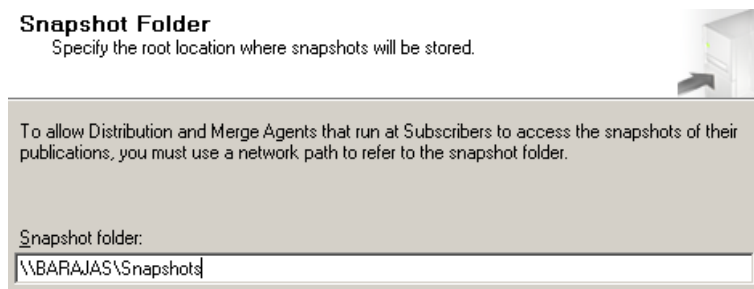
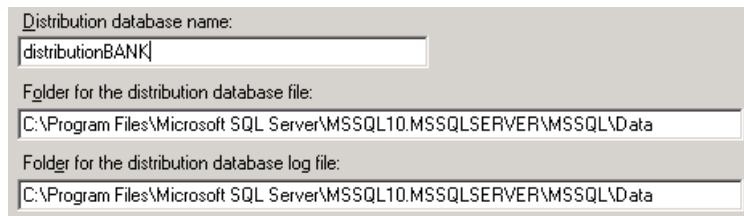


Figure 18. Snapshot store folder

Step 2, a folder where snapshots are stored was specified. This folder is used by the distribution agent for deploying snapshots to subscribers. A folder was previously created for storing snapshots.



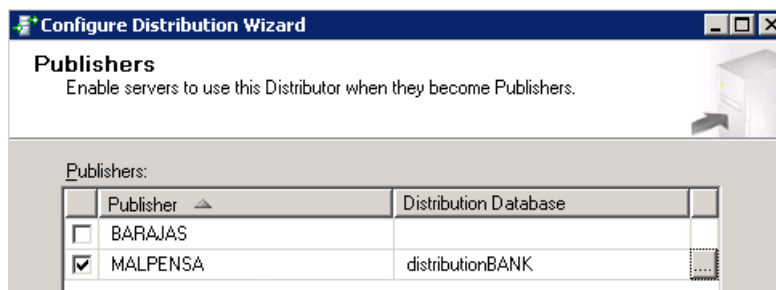
Distribution database name:
distributionBANK

Folder for the distribution database file:
C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Data

Folder for the distribution database log file:
C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Data

Figure 19. Distribution database name and store location

Step 3, the distribution database was named to distributionBANK and selected to be stored locally on the distributor. This data store can locate also in different resource other than local.



Configure Distribution Wizard

Publishers
Enable servers to use this Distributor when they become Publishers.

Publishers:

Publisher	Distribution Database
<input type="checkbox"/> BARAJAS	
<input checked="" type="checkbox"/> MALPENSA	distributionBANK

Figure 20. Publisher selection for the Distributor

Step 4, MALPENSA was selected to use BARAJAS as the Distributor. This also required a password for the remote Publishers to be used for connect to the Distributor. The password used in this context is only used by the connection between remote publisher and the distributor. If a local publisher is chosen, no password is required.

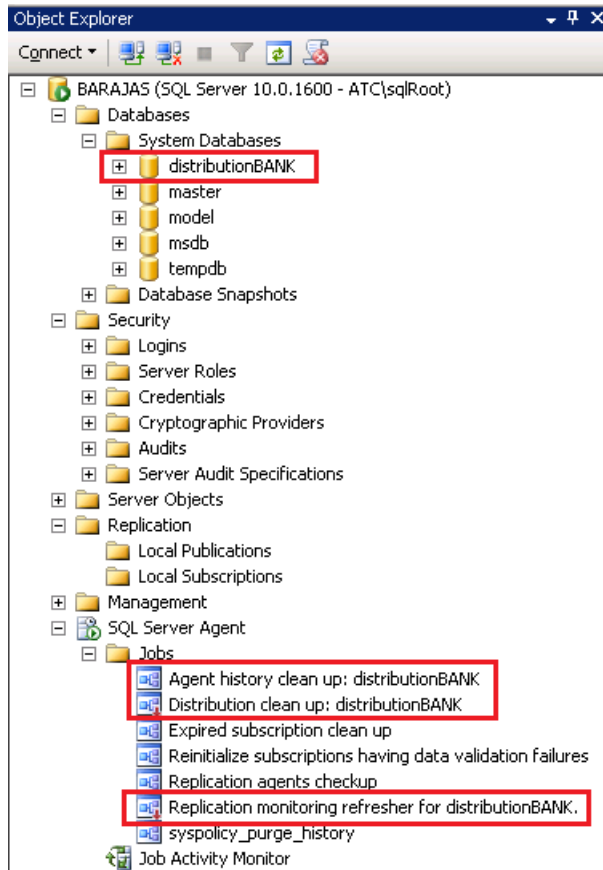


Figure 21. Distributor database objects

Last **step 5** in the wizard was to select the option to create a script for the distributor. Script is in Appendix 4. Jobs for the SQL Server Agent were added, as seen in Figure 21. The distribution database distributionBANK was also created by the wizard.

7.3.2 Configuring the Publisher

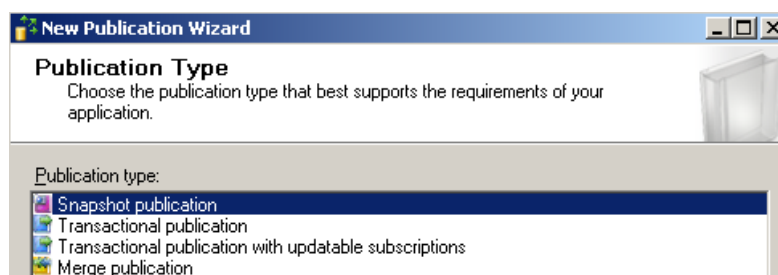


Figure 22. Publication type selection

Step 1, publisher configuration starts by right-clicking folder “Local Publications” and selecting “New Publication”. The wizard starts with the selection of the database and publication type. Snapshot publication was selected as publication type.

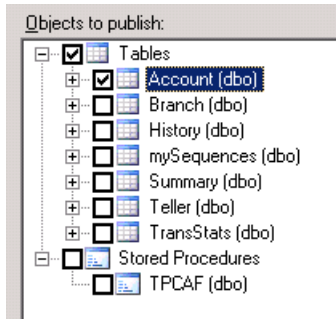


Figure 23. Selection of articles for publication

Step 2, selection of the articles for the publication. Account table from the bank database was chosen for this test and no table rows were filtered.

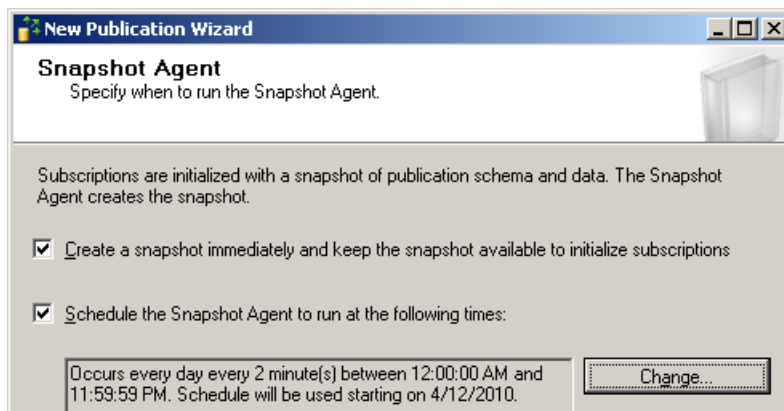


Figure 24. Snapshot agent schedule

Step 3, the snapshot agent was scheduled to run every 2 minutes. Running snapshot agent as often is not recommend action in live databases. This would mean that database is copied every 2 minutes and would fill disks very quickly. For the test however, this was selected to observe changes in relatively short period of time.

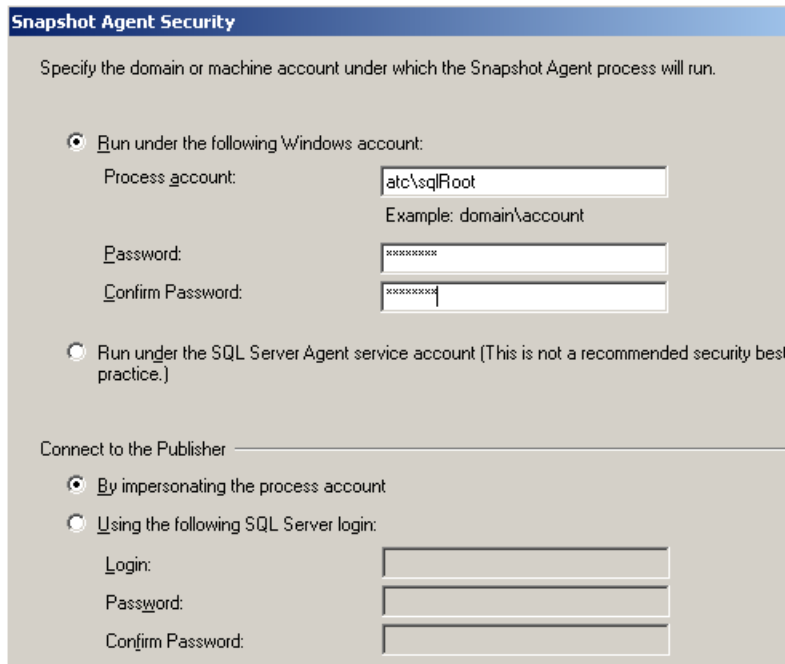


Figure 25. Snapshot agent security details

Step 4, Snapshot agent security details were entered. All agents and connections were set to run under AD login sqlRoot.

Last steps were the creation and naming the publication, and script generation (see Appendix 5). Publication was named publicationSnapBANK.

7.3.3 Configuring subscribers



Figure 26. Publisher selection on subscriber wizard

In the test setup, two subscribers were used. The setup wizard starts by right-clicking folder “Local Subscriptions” and selecting “New Subscriptions”. The publisher was selected from the drop down menu to point to server MALPENSA, the test setup publisher server.

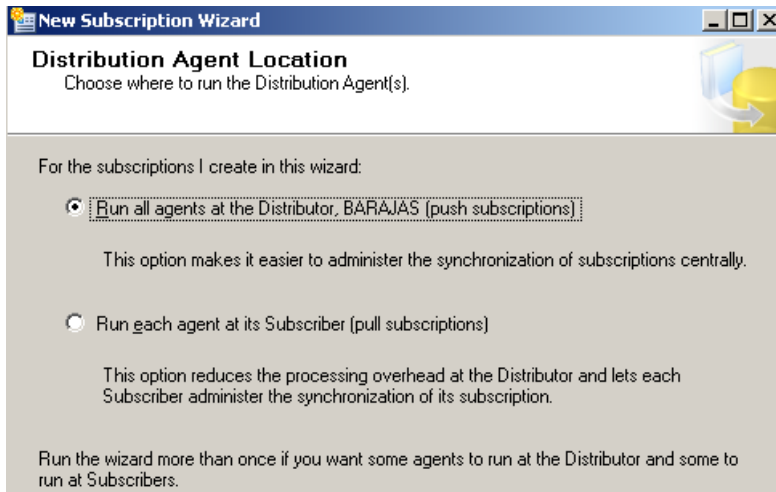


Figure 27. Distribution Agent location selection

Step 2 was to select where the distribution agent will run. Push subscription was selected for the subscriber server ARLANDA, thus location was at the distributor. For server SCHIPHOL pull subscription was chosen. This selection affects the location of the Distribution Agent. Therefore, in case of pull subscription, the agent runs at the subscriber and for push subscription, it runs at the distributor.

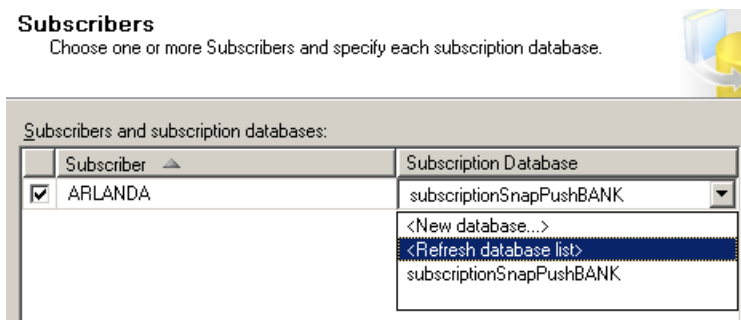


Figure 28. Subscriber database selection

Step 3, a new database was created for each subscriber. For push subscriber at server ARLANDA it was named as subscriptionSnapPushBANK, and for pull subscriber at server SCHIPHOL name used was subscriptionSnapPullBANK.

Step 4 involved similar setting for the distribution agent as with publication configuration illustrated in Figure 25. Same AD login was used for both subscriber configurations.

Step 5 was synchronization schedule. Both were configured to run continuously. It is possible to schedule synchronization to happen in its own schedule, say after snapshot generation.

With the test setup this was not required. Both subscribers will receive the new snapshot after snapshot agent generates it in every 2 minutes.

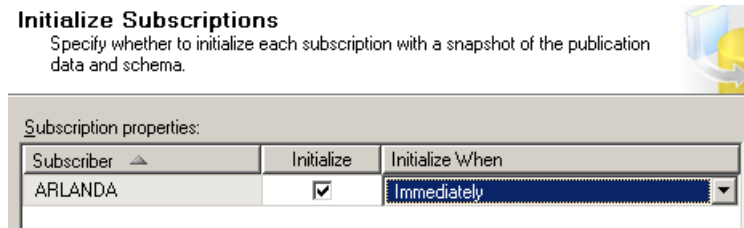



Figure 29. Subscriber initialization selection

Step 6 was selection of when to initialize subscriptions. For both subscribers, snapshots were selected to initialize immediately. This means that the wizard will copy the latest snapshot to the subscribers as a last step. Other option is to allow initialization during next synchronization. The wizard exits with the options to create scripts and these were added as Appendix 6 and 7.

7.4 Transactional and merge test scenario configurations

Transactional replication

Configuration for the transactional replication was similar to the snapshot scenario. On **Step 1** on publication setup, the replication type selected was transactional. It was named as publicationTranBANK. Basically the only difference in contrast to snapshot publication was articles selection. History table, due it not having a primary key column wasn't selectable.

 This table cannot be published because it does not have a primary key column. Primary key columns are required for all tables in transactional publications.

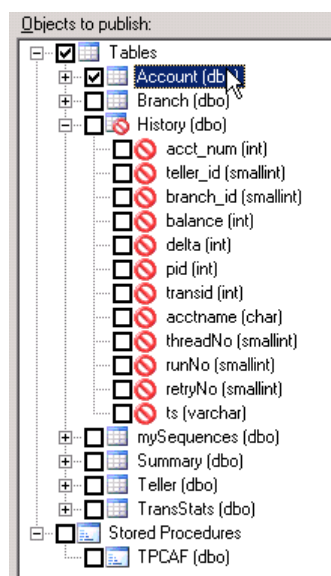


Figure 30. Articles selection for transaction replication

Publication script is included in Appendix 8. Subscribers were configured for transactional replication by selecting the correct publication from the publisher in similar way as in Figure 26. Both were also selected to synchronize continuously. Script for configuring subscriber with pull subscription, SCHIPHOL is included in Appendix 9, and script for push subscriber ARLANDA is included in Appendix 10.

Transactional replication with updatable subscriptions

Only difference to transactional replication setup was to select how transactions would commit at the publisher. For server ARLANDA, queued updating was selected. Dissimilar for server SCHIPHOL, simultaneously committing updates was selected. Scripts are included as Appendix 11 for publication, Appendix 12 for subscriber ARLANDA and Appendix 13 for subscriber SCHIPHOL.

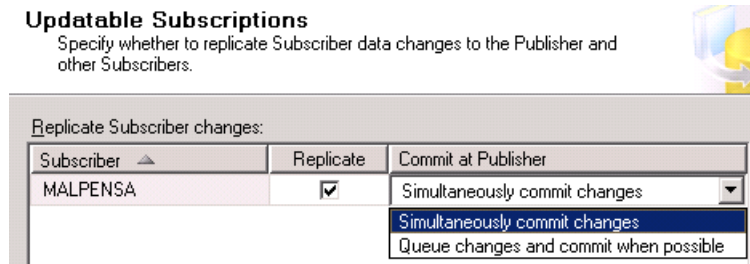


Figure 31. Commit selection of updatable subscriptions

Merge replication

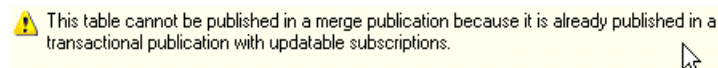
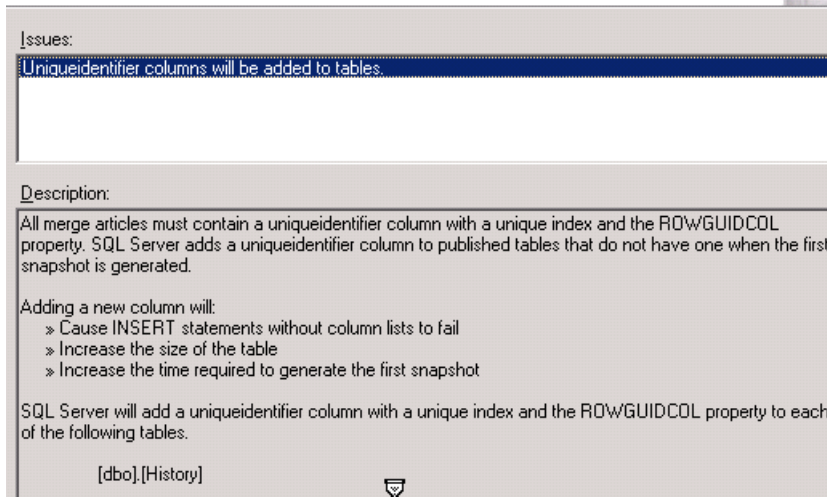


Figure 32. Error selecting articles for merge replication

In merge replication setup, a new copy of the Bank database was required as only table available for replication was the History table. New copy was named as BankCopy and it was only used to test merge replication. Script for creating merge publication is in Appendix 14.

Article Issues

The following issues may require changes to your application to ensure it continues to operate as expected.



Issues:

Uniqueidentifier columns will be added to tables.

Description:

All merge articles must contain a uniqueidentifier column with a unique index and the ROWGUIDCOL property. SQL Server adds a uniqueidentifier column to published tables that do not have one when the first snapshot is generated.

Adding a new column will:

- » Cause INSERT statements without column lists to fail
- » Increase the size of the table
- » Increase the time required to generate the first snapshot

SQL Server will add a uniqueidentifier column with a unique index and the ROWGUIDCOL property to each of the following tables.

[dbo].[History]

Figure 33. Merge replication article selection

The wizard added unique identifier column for all merge replication articles. This was to ensure the uniqueness of all subscriber data. These were the only differences compared to the setup of Snapshot replication. Scripts for creating merge replication subscribers are in Appendix 15 for subscriber ARLANDA, 16 for subscriber SCHIPHOL and 17 for subscriber MALPENSA.

7.5 Snapshot, transactional and merge replication test run

The data generation script sets the value of the balance column in Account table for every row as 1000. On the test, an update operation is used to update the first two rows. On both targets, the subscribing servers, a SELECT statement was applied before and after update to test the snapshot propagation. Before update, the situation was as seen on both subscribers with the following SELECT operation. For the test the snapshot replication was setup so, that the snapshot agent runs every 2 minutes and synchronization is constant.

```
DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
SELECT @@SERVERNAME AS "ServerName",@ts AS "Time"
SELECT acct_num, balance FROM Account
WHERE acct_num < 4
```

	Timestamp	Server
1	17:20:35	ARLANDA

	acct_num	balance
1	1	1000
2	2	1000
3	3	1000

Figure 34. ARLANDA before snapshot update

	Timestamp	Server
1	17:20:34	SCHIPHOL

	acct_num	balance
1	1	1000
2	2	1000
3	3	1000

Figure 35. SCHIPHOL before snapshot update

Next, an update operation was done on the publisher MALPENSA with the following SQL statement.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
SELECT @ts AS "Timestamp", @@SERVERNAME AS "ServerName"
-- Timestamp --
UPDATE Account
SET balance = 1111
WHERE acct_num = 1

UPDATE Account
SET balance = 2222
WHERE acct_num = 2
-- Update --

```

	Timestamp	ServerName
1	17:40:37	MALPENSA

Figure 36. Update operation done on server MALPENSA

This operation updates the first two rows of the table Account. Additionally, a timestamp of the operation was added for comparison with the snapshot synchronization. After update the select operation was run again on both subscribers and refreshed manually every few seconds. Finally, after few minutes both updates were applied.

Results		Messages	
	Timestamp	Server	
1	17:42:07	SCHIPHOL	

	acct_num	balance
1	1	1111
2	2	2222
3	3	1000

Figure 37. SCHIPHOL after update

Results		Messages	
	Timestamp	Server	
1	17:42:10	ARLANDA	

	acct_num	balance
1	1	1111
2	2	2222
3	3	1000

Figure 38. ARLANDA after update

Agent History					
View: <input type="text" value="The last 100 synchronizations"/>					
Sessions of the Snapshot Agent:					
	Status	Start Time	End Time	Duration	Error Message
✓	Completed	4/28/2010 5:42:02 PM	4/28/2010 5:42:04 PM	00:00:02	
✓	Completed	4/28/2010 5:40:02 PM	4/28/2010 5:40:04 PM	00:00:02	
✓	Completed	4/28/2010 5:38:02 PM	4/28/2010 5:38:04 PM	00:00:02	
✓	Completed	4/28/2010 5:36:02 PM	4/28/2010 5:36:05 PM	00:00:03	

Figure 39. Snapshot agent log

As we can see from results of both SELECT operations, transactions were successful and updated at almost same time. The update was seen at 17:42:10 on server ARLANDA and at 17.42.07 for server SCHIPHOL. This is comparable to the synchronization history log, which can be seen from the replication monitor tool. Figure 39 is taken from the snapshot agent log. It shows the start and stop times for the agent. Snapshot was generated at 17:42:04.

Transactional replication test

Same test-run was used to test the transactional replication. Update was done on server MAL-PENSA with similar update statement than in previous test.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
SELECT @ts AS "Timestamp", @@SERVERNAME AS "ServerName"
-- Timestamp --
UPDATE Account
SET balance = 3333
WHERE acct_num = 1

UPDATE Account
SET balance = 4444
WHERE acct_num = 2
-- Update --

```

	Timestamp	ServerName
1	18:13:02	MALPENSA

Figure 40. MALPENSA timestamp after update

This produced a timestamp for comparing results on subscribing servers. Both subscribers were refreshed manually, and at 10:13:08 showed the transaction complete.

	Timestamp	Server
1	18:13:08	SCHIPHOL

	acct_num	balance
1	1	3333
2	2	4444
3	3	1000

Figure 41. SCHIPHOL after update

	Timestamp	Server
1	18:13:08	ARLANDA

	acct_num	balance
1	1	3333
2	2	4444
3	3	1000

Figure 42. ARLANDA after update

Agent History					
View: The last 100 synchronizations					
Sessions of the Log Reader Agent:					
	Status	Start Time	End Time	Duration	Error Message
	Running	4/28/2010 12:45:18 PM	-	05:27:47	

Actions in the selected session:	
Action Message	Action Time
2 transaction(s) with 4 command(s) were delivered.	4/28/2010 6:13:05 PM
Initializing	4/28/2010 12:45:18 PM
Starting agent.	4/28/2010 12:45:18 PM

Figure 43. Log reader agent log

From log reader agent log, the two transactions can be seen committed at 18:13:05. This is comparable with the time it was committed at the publisher MALPENSA, as seen in Figure 40.

Transactional replication with updatable subscribers test

In this test it was mandatory to experiment with update propagation from the subscribers to publisher, and to other subscribers. Original publication was created with default options, thus it had “Conflict resolution policy” set as “Keep the publisher change”. This means that whichever subscriber updates first, gets to keep the update. This update will then be applied to publication and to all subscribers.

For that reason, another publication was created with conflict resolution set to “Keep Subscriber Change”. This was the only way to test and distinguish updates from subscribers as one publication can only be either.

Situation before update at publisher server MALPENSA was as seen in Figure 42.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
SELECT @@SERVERNAME AS "ServerName",@ts AS "Time"
SELECT acct_num, balance FROM Account
WHERE acct_num < 4

```

Results		Messages	
ServerName	Time		
1	MALPENSA	15:47:44.593	

	acct_num	balance
1	1	3333
2	2	4444
3	3	1000

Figure 44. MALPENSA before update

Next, both subscribers were updated manually with conflicting data.

Results		Messages	
Timestamp	Server		
1	15:47:47.447	SCHIPHOL	

	acct_num	balance
1	1	3166
2	2	4221
3	3	1000

Figure 45. Update done on subscriber SCHIPHOL

Subscriber SHIPHOL was updated with the following:

```

DECLARE @ts time(3)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account
SET balance = balance * 0.95 WHERE acct_num < 3
-- Update -
SELECT @ts AS "Timestamp", @@SERVERNAME AS "Server"
SELECT acct_num, balance
FROM Account WHERE acct_num < 4
-- Results -

```

Results		Messages	
Timestamp	Server		
1	15:47:51.463	ARLANDA	

	acct_num	balance
1	1	3499
2	2	4666
3	3	1000

Figure 46. Update done on subscriber ARLANDA

Subscriber ARLANDA was updated with the following:

```

DECLARE @ts time(3)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account
SET balance = balance * 1.05 WHERE acct_num < 3
-- Update -

```

```

SELECT @ts AS "Timestamp", @@SERVERNAME AS "Server"
SELECT acct_num, balance
FROM Account WHERE acct_num < 4
-- Results -

```

	ServerName	Time
1	MALPENSA	15:48:17.830

	acct_num	balance
1	1	3499
2	2	4666
3	3	1000

Figure 47. Publisher MALPENSA after update

As seen in Figure 47, subscriber ARLANDA won the update. This update was then propagated to all subscribers including SCHIPHOL, which had the conflicting update. This can be seen on Figure 48.

	Timestamp	Server
1	15:48:04.857	SCHIPHOL

	acct_num	balance
1	1	3499
2	2	4666
3	3	1000

Figure 48. SCHIPHOL after update propagation

Merge replication test

On the merge replication test the merge agent was set to run every 3 minutes. Since all subscribers were set as servers, it was possible to distinguish updates and data from one another. Subscriber SCHIPHOL was assigned with value 74.00, ARLANDA 75.00 and another subscriber with a value of 76.00 was added to MALPENSA server. Before the update, all servers had no changes made to the database. Moreover, this database was exactly like it was setup by the BANK scripts.

SCHIPHOL with the lowest priority of 74.00 updated first with following DML clause.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account
SET balance = balance * 0.95 WHERE acct_num < 3
-- Update --
SELECT @ts AS "Timestamp", @@SERVERNAME AS "Server"
SELECT acct_num, balance
FROM Account WHERE acct_num < 4
-- Results --

```

Timestamp	Server
17:22:36	SCHIPHOL
17:22:39	SCHIPHOL
17:27:41	SCHIPHOL

acct_num	balance
1	1000
2	1000
3	1000

acct_num	balance
1	950
2	950
3	1000

acct_num	balance
1	500
2	500
3	1000

Figure 49. SCHIPHOL before update, update and after synchronization

Next, ARLANDA with priority 75.00 updated with the following SQL statement.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account
SET balance = balance * 1.05 WHERE acct_num < 3
-- Update --
SELECT @ts AS "Timestamp", @@SERVERNAME AS "Server"
SELECT acct_num, balance
FROM Account WHERE acct_num < 4
-- Results --

```

Timestamp	Server
17:22:44	ARLANDA
17:22:48	ARLANDA
17:27:50	ARLANDA

acct_num	balance
1	1000
2	1000
3	1000

acct_num	balance
1	1050
2	1050
3	1000

acct_num	balance
1	500
2	500
3	1000

Figure 50. ARLANDA before, update and after synchronization

MALPENSA with priority 76.00 was updated last with the following SQL statement.

```

DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account
SET balance = balance * 0.5 WHERE acct_num < 3
-- Update --
SELECT @ts AS "Timestamp", @@SERVERNAME AS "Server"
SELECT acct_num, balance
FROM Account WHERE acct_num < 4

```

Timestamp	Server
17:22:55	MALPENSA

acct_num	balance
1	1000
2	1000
3	1000

Timestamp	Server
17:23:00	MALPENSA

acct_num	balance
1	500
2	500
3	1000

Timestamp	Server
17:27:45	MALPENSA

acct_num	balance
1	500
2	500
3	1000

Figure 51. MALPENSA before update, update and after synchronization

What can be seen from the Figures is that although the server with lowest priority updated first, it lost its update. If all subscribers were configured as clients or servers with default value of 75.00, the situation would have been completely opposite. SCHIPHOL's update would have been applied to the publication, and swarmed over to the other subscribers. Merge agent log from SCHIPHOL verifies this action. (Figure 52)

Status	Start Time	End Time	Duration	Uploaded Commands	Downloaded Commands	Error Message
Completed	4/30/2010 5:33:00 PM	4/30/2010 5:33:01 PM	00:00:01	0	0	
Completed	4/30/2010 5:30:00 PM	4/30/2010 5:30:01 PM	00:00:01	0	0	
Completed	4/30/2010 5:27:00 PM	4/30/2010 5:27:01 PM	00:00:01	0	2	
Completed	4/30/2010 5:24:00 PM	4/30/2010 5:24:00 PM	00:00:00	2	0	

Article	% of Total	Duration	Inserts	Updates	Deletes	Conflicts	Retries	Schema Changes
Initialization	100	00:00:01	-	-	-	-	-	-
Schema changes and bulk inserts	0	00:00:00	0	-	-	-	-	0
Upload changes to Publisher	0	00:00:00	0	0	0	0	0	-
Download changes to Subscriber	0	00:00:00	0	2	0	2	0	-
Article: Account	0	00:00:00	0	2	0	2	0	-

Figure 52. Merge agent log

What happened can be seen from logs. During synchronization (Figure 53) at 17:24:00 merge agent logged changes and uploaded them to the publisher. At 17:27:00 the agent discovered 2 conflicting updates and updated subscribers accordingly.

Status	Start Time	End Time	Duration	Uploaded Commands	Downloaded Commands
Completed	4/30/2010 5:24:00 PM	4/30/2010 5:24:00 PM	00:00:00	2	0

Article	% of Total	Duration	Inserts	Updates	Deletes	Conflicts	Retries	Schema Changes
Initialization	100	00:00:00	-	-	-	-	-	-
Schema changes and bulk inserts	0	00:00:00	0	-	-	-	-	0
Upload changes to Publisher	0	00:00:00	0	2	0	0	0	-
Article: Account	0	00:00:00	0	2	0	0	0	-
Download changes to Subscriber	0	00:00:00	0	0	0	0	0	-

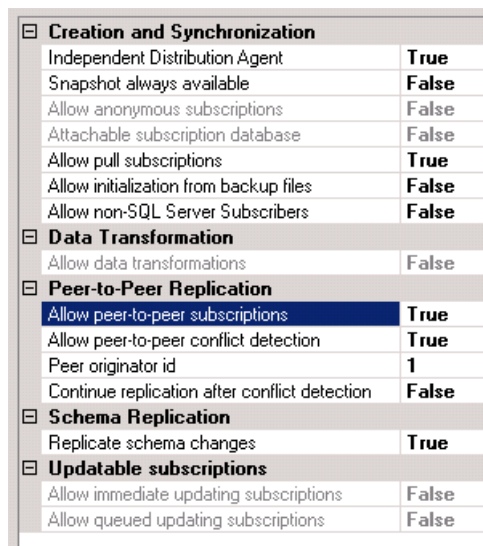
Figure 53. Merge agent log

7.6 Transactional peer-to-peer replication configuration

For this test the all previous publications, subscribers and distributor was removed. A backup off the BANK database at server MALPENSA was created (Appendix 18). This backup was copied and restored on both servers BARAJAS and SCHIPHOL manually (Appendix 19).

Step 1, all 3 servers were configured to be their own distributor and publisher (Appendices 20 for server MALPENSA, 21 for server SCHIPHOL and 22 for server BARAJAS).

Step 2. On server MALPENSA, a new publication was created from the BANK database. Publication type selected was transactional with Account table selected as replicated article. No snapshots were generated nor scheduled to run. Publication was named as publication-PeerBANK. Publication creation script is in Appendix 23.



Creation and Synchronization	
Independent Distribution Agent	True
Snapshot always available	False
Allow anonymous subscriptions	False
Attachable subscription database	False
Allow pull subscriptions	True
Allow initialization from backup files	False
Allow non-SQL Server Subscribers	False
Data Transformation	
Allow data transformations	False
Peer-to-Peer Replication	
Allow peer-to-peer subscriptions	True
Allow peer-to-peer conflict detection	True
Peer originator id	1
Continue replication after conflict detection	False
Schema Replication	
Replicate schema changes	True
Updatable subscriptions	
Allow immediate updating subscriptions	False
Allow queued updating subscriptions	False

Figure 54. Peer-to-peer replication setup

Step 3. When publication was ready, the actual peer-to-peer configuration was started by right-clicking publication and selecting properties. On the subscription options page the option “Allow peer-to-peer subscriptions” was set to true.

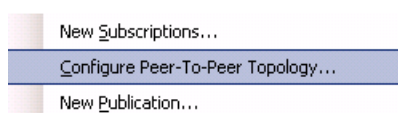


Figure 55. New menu item

Step 3. After the peer-to-peer replication was set to true on subscription options, a new option was added to right-click menu.

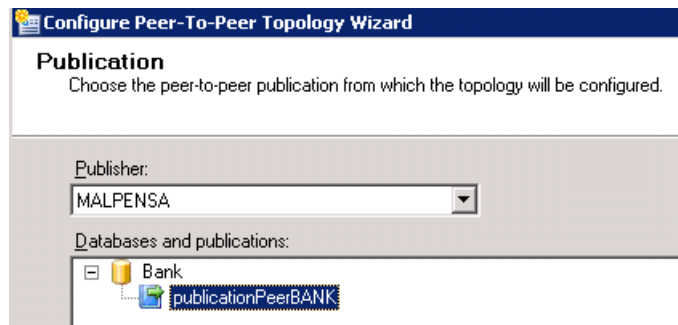


Figure 56. Publication selection

Step 4. A new Wizard starts to begin the peer-to-peer replication configuration. The publisher selected was local server MALPENSA and the newly created publication.

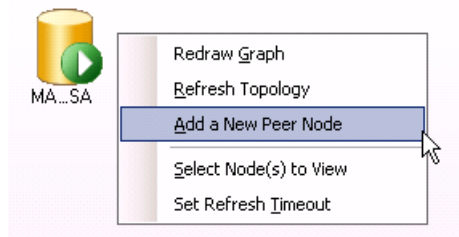


Figure 57. New node for peer-to-peer setup

Step 5. A new node was added to the setup. First one added was server BARAJAS.

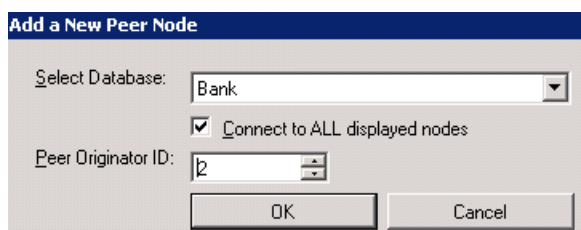


Figure 58. Peer node details

Step 6. After connection was made to BARAJAS, a selection for database was displayed. Also, every peer required a unique ID and connection to all other nodes in topology.



Figure 59. Peer-to-peer topology with one node added

Step 7. The new node was added to the peer-to-peer topology. Next, **steps 8 and 9** was to select log reader agent and distribution agent logins.

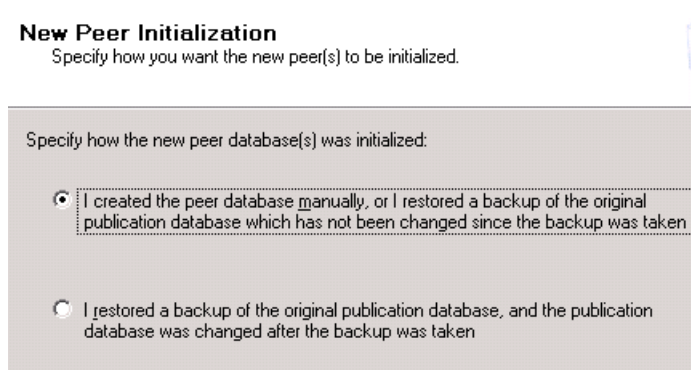


Figure 60. Peer initialization selection

Step 10, the wizard required details of how new peer are to be initialized. In test setup, databases were restored from a backup manually. This was the last step in first node addition and then the wizard exited.

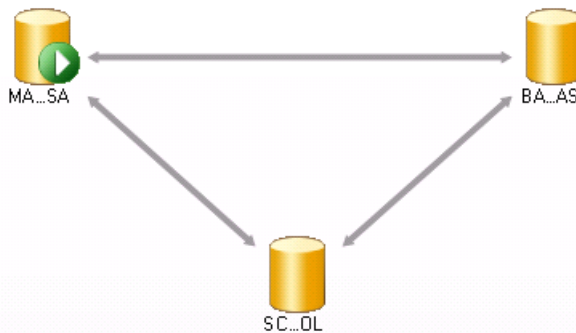


Figure 61. Final peer-to-peer replication topology

Step 11, another node was added to topology in a similar fashion as before. The final setup looked like in Figure 62.

7.6.1 Transactional peer-to-peer replication test

First test was simple update to observe replication was in working condition. Before any updates, the situation in all servers was checked with following SELECT clause.

```
DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
SELECT @@SERVERNAME AS "ServerName",@ts AS "Time"
SELECT acct_num, balance FROM Account
WHERE acct_num < 4
-- Results
```

ServerName	Time
1 MALPENSA	12:03:53

Server	Timestamp
1 SCHIPHOL	12:03:54

ServerName	Time
1 BARAJAS	12:03:57

acct_num	balance
1	1000
2	1000
3	1000

acct_num	balance
1	1000
2	1000
3	1000

acct_num	balance
1	1000
2	1000
3	1000

Figure 63. Peer status before updates

Next, the following update was applied on server MALPENSA.

```
DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
UPDATE Account SET balance = 1111 WHERE acct_num = 1
UPDATE Account SET balance = 2222 WHERE acct_num = 2
-- Update --
SELECT @@SERVERNAME AS "ServerName",@ts AS "Time"
SELECT acct_num, balance FROM Account
WHERE acct_num < 4
-- Results
```

ServerName	Time
1 MALPENSA	12:11:48

ServerName	Time
1 BARAJAS	12:12:01

Server	Timestamp
1 SCHIPHOL	12:12:05

acct_num	balance
1	1111
2	2222
3	1000

acct_num	balance
1	1111
2	2222
3	1000

acct_num	balance
1	1111
2	2222
3	1000

Figure 64. Peer status after update

The update was propagated to other peers from server MALPENSA.

Peer-to-Peer Replication	
Allow peer-to-peer subscriptions	True
Allow peer-to-peer conflict detection	True
Peer originator id	1
Continue replication after conflict detection	True

Figure 65. Peer-to-peer replication configuration

In peer-to-peer replication, peer with highest ID always wins conflict situation. In the next test, a conflict was created between the peers. Before the test, an option to allow replication to continue after conflict detection was set to true. Otherwise the replication would have stopped and manual resolution would've required.

A conflict situation was done so that on sever MALPENSA following script was applied in two parts. Before committing transaction, server SCHIPHOL was updated with conflicting data.

```
BEGIN TRANSACTION
UPDATE Account SET balance = 1111 WHERE acct_num = 1
UPDATE Account SET balance = 2222 WHERE acct_num = 2
-- Update --
DECLARE @ts time(0)=(CURRENT_TIMESTAMP)
-- Timestamp --
SELECT @@SERVERNAME AS "ServerName",@ts AS "Time"
SELECT acct_num, balance FROM Account
WHERE acct_num < 4
-- Wait for update on SCHIPHOL before COMMIT ... --
COMMIT
```

SCHIPHOL was updated with following and MALPENSA transaction was committed.

```
UPDATE Account SET balance = 2112 WHERE acct_num = 1
```

ServerName	Time	acct_num	balance
1	MALPENSA	1	2112
		2	2222
		3	1000

Server	Timestamp	acct_num	balance
1	SCHIPHOL	1	2112
		2	2222
		3	1000

ServerName	Time	acct_num	balance
1	BARAJAS	1	2112
		2	2222
		3	1000

Figure 66. Peer status after conflicting update

SCHIPHOL had higher ID than MALPENSA, thus the update winner was SCHIPHOL. The update for the second row was updated by MALPENSA since there was no conflict.

Microsoft Replication Conflict Viewer for 'Bank' -- '[dbo].[Account]'			
File View Records Help			
Database:	Bank	Publication:	publicationPeerBA
		Table:	[dbo].[Account]
Conflict peers (winner/loser)	Conflict type	Reason	Date
SCHIPHOL/MALPENSA	1(Update conflict)	1(Resolved)	5/3/2010 1:17:16 PM
An update-update conflict between peer 3 (incoming) and peer 1 (on disk) was detected and resolved. The incoming update was applied			
Column name	Conflict winner	Conflict loser	
acct_num	1	1	...
name	...	Account 1	...
branch_id	...	1	...
balance	2112	1111	...
address	...	User Address 1	...
temp1	...	Temp 1	...
msrepl_tran_version	...	8de9a494-5042-42c5-aae3-24bad574ba77	...

Figure 67. Replication conflict viewer

The conflict was seen also with replication conflict viewer. The affected peers and the conflict winner/loser values are recorded, along with the reason code and timestamp.

8 Discussion

The reliability of the test setup was validated in the tests. It was documented thoroughly, thus if tests are duplicated using information of this test setup, it will work. With this in mind, there's always other things involved in a networked system, therefore there might come a situation what has not been tested and fixed in this study.

Foreword on the tests

Due to tight schedule, tests were not as complete as originally planned. A number of things, namely performance testing, DML operations and fault simulation were left out. Also, tests were carried out only in SQL server, whereas literature review was conducted also from IBM and Oracle replication systems. In an ideal situation, the same hardware would've been used with Linux operating system, and DB2 along with Oracle replication systems would have tested. That was the original, albeit ambitious plan.

9 Conclusion

Results from the tests

First test was to test snapshot replication. A publisher with remote distributor, pull and push subscribers. With this replication type, along with table subsets and filters, the only things to customize are snapshot agent schedule and synchronization schedule. For this, the snapshots were generated every 2 minutes and synchronized constantly. It was easy to setup, and it was a successful test. The updates done on the publisher were replicated to subscribers.

Second was transactional replication with approximately the same setup. Push and pull subscribers were initialized from a snapshot and synchronized constantly. An update done on the publisher was replicated to subscribers. It took 3 seconds for the log reader agent to react and apply the update, while on subscribers the update was seen 5 seconds afterwards.

Third test was transactional replication with updatable subscribers. Again same setup, however a new publication from the Bank database was created to test the rules for this replication type. An update was applied on both subscribers with conflicting data, affecting same rows. The subscriber updating publication with "Keep Publisher Change" option updated first and subscriber with option "Keep Subscriber Change" second. For a period of time, there was a situation where subscribers had inconsistent data.

Fourth, merge replication was tested. This time a local subscriber on the publishing server was added to the test. Otherwise the setup was same. Subscribers were configured as servers, thus had assignable priority values. A conflicting update was applied from all the subscribers, which was noted and fixed by the merge agent according to the priority values.

Last test was peer-to-peer transactional replication test. This was done with a 3 peer setup. A conflicting update was applied from one server, while another was in a middle of transaction. The conflict resolver was able to catch and resolve the conflict, thus all servers were updated with the winning update.

Overall results

The literature review provides a base to understand database replication, the inner workings and techniques that are needed to form a replication system. Different scenarios and replication types are presented. Along with the test results, it can be used as an introduction to database replication. There's also a good list of references for anyone who's interested to gain more in-depth knowledge about it.

What next? Topics to research in future

Topics that have close resemblance, or are used together or in replication environments are High availability solutions. One of them is mirroring, which provides fail-over mirror that in case of failure can take over automatically. Clustering is another high availability technique. Similar to mirror system, it involves number of servers that continues to provide services while one or more server is down. As a personal preference, it would be interesting to study one these techniques.

Bibliography

Ashok G. & Randal S. P. SQL Server Replication: Providing high using Database mirroring. 2008. URL: <http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/ReplicationAndDBM.docx>. Quoted 13.03.2010

Bernstein A. P., Hadzilacos V., Goodman N. Concurrency Control and Recovery in Database Systems. 1987. Microsoft Research. URL: <http://research.microsoft.com/en-us/people/philbe/chapter8.pdf>. Quoted: 12.04.2010

Connolly, T. & Begg, C. 2005. Database Systems. A Practical Approach to Design, Implementation, and Management. 4th edition. Addison-Wesley

Cotter, H. Deploying Merge Replication Subscribers in SQL Server. 2009. URL: <http://www.informit.com/articles/article.aspx?p=1392520>. Quoted: 01.05.2010

EnterpriseDB Corporation 2009. Postgres Advanced Server Replication Server User's Guide. URL: http://www.enterprisedb.com/products/pdf_docs.do. Quoted: 14.03.2010

Gu, L., Budd, L., Cayci, A., Hendricks, C., Purnell, M., Rigdon, C. 2002. A Practical Guide to DB2 UDB Data Replication V8. IBM. URL: <http://www.ibm.com/redbooks>. Quoted 27.04.2010

IBM 2009. IBM DB2 Database for Linux, UNIX and Windows Information Center. URL: <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>. Quoted 27.04.2010

Mackin, J.C., Hotek, M. Designing a Database Server Infrastructure Using SQL Server 2005. 2007. Microsoft Press

Microsoft 2009. SQL Server 2008 Books online. URL: <http://msdn.microsoft.com/en-us/library/ms130214.aspx>. Quoted: 05.05.2010

Nielsen, P., White, M., Parui, U. SQL Server 2008 Bible. 2009. Wiley Publishing, Inc.

Orafaq 2009. Advanced Replication. URL:

http://www.orafaq.com/wiki/Advanced_Replication. Quoted: 05.05.2010

Oracle 2009a. Oracle Database Advanced Replication 11g Release 2 (11.2) URL:

http://download.oracle.com/docs/cd/E11882_01/server.112/e10706/toc.htm. Quoted:
05.05.2010

Oracle 2009b. Oracle Streams Streams Concepts and Administration 11g release 2 (11.2) URL:

http://download.oracle.com/docs/cd/E11882_01/server.112/e10704/toc.htm. Quoted:
05.05.2010

Oracle 2005. Oracle Database 10g : Oracle Streams Replication. URL :

http://www.oracle.com/technology/products/dataint/pdf/twp_streams_replication_10gr2.pdf
df Quoted : 05.05.2010

Raheem, M. Sonkin, D. D'Hers, T. LeMonds, K. Inside SQL Server 2005. Addison Wesley
Professional

Ramakrishnan, R & Gehrke, J. Database Management Systems. 2nd edition 2002. Mcgraw-Hill

Thomas, O., McLean, I. Optimizing and Maintaining a Database Administration Solution by
Using SQL Server 2005. 2006. Microsoft Press.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G. 2000. Understanding Replica-
tion in Database and Distributed Systems. URL:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.9166&rep=rep1&type=pdf>.
Quoted: 15.04.2010

Index

1-copy serializability	7	mirroring.....	5
aggregate function	5	peer-to-peer replication	14
Articles	10	persistent LCR.....	24
Asynchronous replication	7	Publication.....	10
captured LCR.....	24	Publisher	10
Change-capture replication	16	Queue reader agent	12
Checkpoint	24	SCN number	24
clustering.....	5	Snapshot agent.....	11
DBTechnet.....	3	Snapshot replication.....	12
distributed database.....	5	SQL replication.....	15
distributed system.....	5	SQL Server Agent	11
Distribution agent	12	streams replication	23
Distributor.....	10	Subscriber	11
Full-refresh replication	16	subscription set.....	16
Log reader agent	11	synchronous replication	7
Merge agent.....	12	Synchronous replication.....	7
Merge replication.....	14	Transactional replication	13

Appendices

Appendix 1. Description of DBTech EXT project



DBTech EXT – The new ICT project in EC LLP Transversal Programme

Databases are needed everywhere in the modern information society, building the basis of all business critical applications requiring reliable, secure and scalable data access. The lack of knowledgeable database professionals in the European labour market is already a productivity problem that has led to failed and delayed application development projects in the industry.

The new application of DBTechNet, the European database teacher network (www.DBTechNet.org), for extending the work of our successful LdV project DBTech Pro in years 2002-2005, has been accepted for Commission funding in LLP Transversal Programme KAY3: ICT. Project is scheduled from 1 Jan 2009 to 31 Dec 2010.

Partners in the project include 7 universities, 3 VET institutes, and 1 industry representative, from 6 European countries, including

- Alexander Technology Educational Institute (ATEI), Thessaloniki, Greece
- Finnish Business College, Helsinki, Finland
- Groupe ISAIP-ESAIP, Saint Barthélemy d'Anjou, France
- HAAGA-HELIA University of Applied Sciences, Helsinki, Finland
- I.E.S. Politécnico Jesús Marín, Málaga, Spain
- Omnia, The Joint Authority of Education in Espoo Region, Espoo, Finland
- Reutlingen University, Reutlingen, Germany
- Tieto Public Industry Finland
- University of Macedonia Economic and Social Sciences, Thessaloniki, Greece
- University of Málaga, Málaga, Spain
- University of the West of Scotland, Paisley, UK

and we co-operate with many Outer Circle partners

Description of the Action

Aside the theoretical studies the project addresses the needs of the industry on application development and use of the mainstream database systems used by the European industry. The project extends the results of the DBTech Pro project, the DBTech Framework of courses and the series of hands-on workshops for learning by doing and verifying theories is used to guide the development of learning modules, to cover the latest development in the practical field of database technologies.

Beside the contents the results are extended and exploited to link the education of VET (vocational education and training) and HE (higher education) institutions up to the level needed for professionals in the European labour market and to PhD programmes for the needs of European research addressing also the lifelong learning strategies for the European database professionals. Internet-based virtual laboratories, digital video lessons and online tests will disseminate and exploit the tutorials and workshops to industry (outer circle) partners supporting the lifelong learning of European database professionals by means of ICT.

Target impacts of the project include:

- Integration of VET and HE education curricula for LLP of database professionals
- Recognizing of prior formal and informal learning
- European education and training in database technologies meet the developments in database technologies and the needs of the European labour market
- Improved quality in ICT and co-operation of VET, HE, and the European industry

For more information contact the coordinator of the project

HAAGA-HELIA University of Applied Sciences

att: DBTech / M Laiho

Ratapihantie 13

00520 HELSINKI, Finland

web : www.haaga-helia.fi

Appendix 2. BANK database script

```
USE master
GO

CREATE DATABASE Bank
    ON (NAME=Bank, FILENAME='Bank.mdf',          SIZE = 40MB)
    LOG ON (NAME = 'BankLog', FILENAME='Bank.ldf', SIZE = 10MB)
GO

USE Bank ;

-- Create the tables

CREATE TABLE Branch
(
    branch_id    SMALLINT    NOT NULL,
    branch_name  CHAR (20)   NOT NULL,
    balance      INT         NOT NULL,
    area_code    CHAR (4)    NOT NULL,
    address      CHAR (30)   NOT NULL,
    temp1       CHAR (40)   NOT NULL,
    PRIMARY KEY (branch_id)
) ;

CREATE TABLE Account
(
    acct_num     INT         NOT NULL,
    name         CHAR (20)   NOT NULL,
    branch_id    SMALLINT    NOT NULL,
    balance      INT         NOT NULL,
    address      CHAR (30)   NOT NULL,
    temp1       CHAR (40)   NOT NULL,
    PRIMARY KEY (acct_num),
    FOREIGN KEY (branch_id) REFERENCES Branch (branch_id)
) ;

CREATE TABLE Teller
(
    teller_id    SMALLINT    NOT NULL,
    teller_name  CHAR (20)   NOT NULL,
    branch_id    SMALLINT    NOT NULL,
    balance      INT         NOT NULL,
    teller_code  CHAR (3)    NOT NULL,
    address      CHAR (30)   NOT NULL,
    temp1       CHAR (40)   NOT NULL,
    PRIMARY KEY (teller_id),
    FOREIGN KEY (branch_id) REFERENCES Branch (branch_id)
) ;

CREATE TABLE History
(
    acct_num     INT         NOT NULL,
    teller_id    SMALLINT    NOT NULL,
    branch_id    SMALLINT    NOT NULL,
    balance      INT         NOT NULL,
    delta        INT         NOT NULL,
    pid          INT         NOT NULL,
    transid      INT         NOT NULL,
    acctname     CHAR (20)   NOT NULL,
    threadNo     SMALLINT,
    runNo        SMALLINT,
    retryNo      SMALLINT,
    ts           VARCHAR(40) --timestamp
) ;

CREATE TABLE TransStats (
    clientNo     INTEGER    NOT NULL,
    threadNo     SMALLINT    NOT NULL,
    runNo        SMALLINT    NOT NULL,
```

```

        retryNo      SMALLINT NOT NULL,
        icommit      SMALLINT NOT NULL,
        irollback    SMALLINT NOT NULL,
        iretry       SMALLINT NOT NULL,
        elapsed      FLOAT     NOT NULL,
        PRIMARY KEY (clientNo, threadNo, runNo, retryno)
    ) ;

CREATE TABLE Summary (
    clientNo      INTEGER NOT NULL,
    CSlogic       SMALLINT NOT NULL,
    connects      SMALLINT NOT NULL,
    isolation     SMALLINT NOT NULL,
    threads       SMALLINT NOT NULL,
    runs          SMALLINT NOT NULL,
    commits       SMALLINT NOT NULL,
    rollbacks    SMALLINT NOT NULL,
    retries       SMALLINT NOT NULL,
    maxRetryNo    SMALLINT NOT NULL,
    failPercent   FLOAT     NOT NULL,
    TPS           FLOAT     NOT NULL,
    avgRespTime  SMALLINT NOT NULL,  -- msec
    elapsed       FLOAT     NOT NULL,
    PRIMARY KEY (clientNo)
) ;

CREATE TABLE mySequences (
    seqid  CHAR(20) NOT NULL PRIMARY KEY,
    seqno  INTEGER  NOT NULL
) ;
INSERT INTO mySequences (seqid,seqno) VALUES ('TPCACLIENTNO',0);

-- Grant privileges on the tables to PUBLIC

GRANT SELECT, UPDATE ON Account TO PUBLIC ;
GRANT SELECT, UPDATE ON Branch TO PUBLIC ;
GRANT SELECT, UPDATE ON Teller TO PUBLIC ;
GRANT SELECT, INSERT ON History TO PUBLIC ;

```

Appendix 3. BANK data generation script

```

DECLARE      -- local variables for the block
@chBranch  VARCHAR(20) ,
@chTeller  VARCHAR(20) ,
@chAcct    VARCHAR(20) ,
@intBranch INTEGER ,
@intTeller INTEGER ,
@intAcct   INTEGER ,
@intLoop   INTEGER ;

BEGIN      -- of the Transact-SQL block
SET NOCOUNT ON ;
-- delete the possible old contents

TRUNCATE TABLE ACCOUNT ;
TRUNCATE TABLE TELLER ;
DELETE FROM BRANCH ;

-- and generate the new test material:
SET @chBranch = '' ;
SET @chTeller = '' ;
SET @chAcct = '' ;
SET @intTeller = 1 ;
SET @intAcct = 1 ;

SET @intBranch = 1 ;
WHILE (@intBranch <= 10)
BEGIN      -- outer loop generating the bank branches
    SET @chBranch = CAST (@intBranch AS VARCHAR) ;      -- casting integer
branch_id to varchar2
    INSERT INTO BRANCH (BRANCH_ID,BRANCH_NAME,BALANCE,AREA_CODE,ADDRESS,TEMP1)
    VALUES (@intBranch,'Branch' + @chBranch, 10000000, @chBranch,
            'Branch Address ' + @chBranch, 'Temp ' + @chBranch) ;

    SET @intLoop = 1 ;
    WHILE (@intLoop <= 10)      -- 10 tellers for every branch
    BEGIN
        SET @chTeller = CAST (@intTeller AS VARCHAR) ;
        INSERT INTO TELLER
(TELLER_ID,TELLER_NAME,BRANCH_ID,BALANCE,TELLER_CODE,ADDRESS,TEMP1)
        VALUES (@intTeller,'Teller ' + @chTeller,@intBranch, 10000, @chTeller,
            'TellerAddress ' + @chTeller, 'Temp ' + @chTeller) ;
        SET @intTeller = @intTeller + 1 ;
        SET @intLoop = @intLoop + 1 ;
    END ; -- of generating Tellers

    SET @intLoop = 1 ;
    WHILE (@intLoop <= 10000)      -- 10000 accounts on every branch
    BEGIN      -- <=10000 would generate 100000
accounts in total
        SET @chAcct = CAST (@intAcct AS VARCHAR) ;
        INSERT INTO ACCOUNT (ACCT_NUM,NAME,BRANCH_ID,BALANCE,ADDRESS,TEMP1)
        VALUES (@intAcct,'Account ' + @chAcct, @intBranch, 1000,
            'User Address ' + @chAcct, 'Temp ' + @chAcct) ;
        SET @intAcct = @intAcct + 1 ;
        SET @intLoop = @intLoop + 1 ;
    END ; -- of generating Accounts

    SET @intBranch = @intBranch + 1 ;
END ;      -- of intBranch

END;      -- of the Transact-SQL Block

--- The End ---

```

```

use master
exec sp_adddistributor @distributor = N'BARAJAS'
, @password = N''
GO
exec sp_adddistributiondb @database = N'distributionBANK'
, @data_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_file_size = 2
, @min_distretention = 0
, @max_distretention = 72
, @history_retention = 48
, @security_mode = 1
GO

use [distributionBANK]
if (not exists
(select * from sysobjects where name = 'UIProperties' and type = 'U ')
)
create table UIProperties(id int)
if (exists
(select * from ::fn_listextendedproperty('SnapshotFolder'
, 'user'
, 'dbo'
, 'table'
, 'UIProperties'
, null
, null)
)
)
EXEC sp_updateextendedproperty N'SnapshotFolder'
, N'\\BARAJAS\Snapshots'
, 'user'
, dbo
, 'table'
, 'UIProperties'
else
EXEC sp_addextendedproperty N'SnapshotFolder'
, N'\\BARAJAS\Snapshots'
, 'user'
, dbo
, 'table'
, 'UIProperties'
GO

exec sp_adddistpublisher @publisher = N'MALPENSA'
, @distribution_db = N'distributionBANK'
, @security_mode = 0, @login = N'sa'
, @password = N''
, @working_directory = N'\\BARAJAS\Snapshots'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO

```

```

use master
exec sp_replicationdboption @dbname = N'Bank'
, @optname = N'publish'
, @value = N'true'
GO

use [Bank]
exec sp_addpublication @publication = N'publicationSnapBANK'
, @description = N'Snapshot publication of database 'Bank' from Publisher
'MALPENSA'..'
, @sync_method = N'native', @retention = 0
, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'true'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21
, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
, @repl_freq = N'snapshot'
, @status = N'active'
, @independent_agent = N'true'
, @immediate_sync = N'true'
, @allow_sync_tran = N'false'
, @autogen_sync_procs = N'false'
, @allow_queued_tran = N'false'
, @allow_dts = N'false'
, @replicate_ddl = 1
GO

exec sp_addpublication_snapshot @publication = N'publicationSnapBANK'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 0
, @frequency_subday = 4
, @frequency_subday_interval = 2
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1

use [Bank]
exec sp_addarticle @publication = N'publicationSnapBANK'
, @article = N'Account'
, @source_owner = N'dbo'
, @source_object = N'Account'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509D
, @identityrangemanagementoption = N'manual'
, @destination_table = N'Account'
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
GO

```

Appendix 6. Snapshot subscriber ARLANDA script

```
-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationSnapBANK'
, @subscriber = N'ARLANDA'
, @destination_db = N'subscriptionSnapPushBANK'
, @subscription_type = N'Push'
, @sync_type = N'automatic'
, @article = N'all'
, @update_mode = N'read only'
, @subscriber_type = 0
exec sp_addpushsubscription_agent @publication = N'publicationSnapBANK'
, @subscriber = N'ARLANDA'
, @subscriber_db = N'subscriptionSnapPushBANK'
, @job_login = N'atc\sqlRoot'
, @job_password = null
, @subscriber_security_mode = 1
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 2
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100423
, @active_end_date = 99991231
, @enabled_for_syncmgr = N'False'
, @dts_package_location = N'Distributor'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----
```

Appendix 7. Snapshot subscriber SCHIPHOL script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationSnapBANK'
, @subscriber = N'SCHIPHOL'
, @destination_db = N'subscriptionSnapPullBANK'
, @sync_type = N'Automatic'
, @subscription_type = N'pull'
, @update_mode = N'read only'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'SCHIPHOL'-----
use [subscriptionSnapPullBANK]
exec sp_addpullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationSnapBANK'
, @publisher_db = N'Bank'
, @independent_agent = N'True'
, @subscription_type = N'pull'
, @description = N''
, @update_mode = N'read only'
, @immediate_sync = 1

exec sp_addpullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'Bank'
, @publication = N'publicationSnapBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 3
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100423
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlRoot'
, @job_password = null
, @publication_type = 0
GO
-----END: Script to be run at Subscriber 'SCHIPHOL'-----

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationSnapBANK'
, @subscriber = N'SCHIPHOL'
, @destination_db = N'subscriptionSnapPullBANK'
, @sync_type = N'Automatic'
, @subscription_type = N'pull'
, @update_mode = N'read only'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'SCHIPHOL'-----
use [subscriptionSnapPullBANK]
exec sp_addpullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationSnapBANK'
, @publisher_db = N'Bank'
, @independent_agent = N'True'

```



```

, @subscription_type = N'pull'
, @description = N''
, @update_mode = N'read only'
, @immediate_sync = 1

exec sp_addpullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'Bank'
, @publication = N'publicationSnapBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 3
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100423
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlRoot'
, @job_password = null
, @publication_type = 0
GO
-----END: Script to be run at Subscriber 'SCHIPHOL'-----

```

Appendix 8. Transactional publication script.

```
exec sp_replicationdboption @dbname = N'Bank'
, @optname = N'publish'
, @value = N'true'

exec [Bank].sys.sp_addlogreader_agent @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1
, @job_name = null

exec sp_addpublication @publication = N'publicationTranBANK'
, @description = N'Transactional publication of database 'Bank' from Publisher
'MALPENSA'..'
, @sync_method = N'concurrent', @retention = 0
, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'true'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
, @repl_freq = N'continuous'
, @status = N'active'
, @independent_agent = N'true'
, @immediate_sync = N'true'
, @allow_sync_tran = N'false'
, @autogen_sync_procs = N'false'
, @allow_queued_tran = N'false'
, @allow_dts = N'false'
, @replicate_ddl = 1
, @allow_initialize_from_backup = N'false'
, @enabled_for_p2p = N'false'
, @enabled_for_het_sub = N'false'

exec sp_addpublication_snapshot @publication = N'publicationTranBANK'
, @frequency_type = 1
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1

exec sp_addarticle @publication = N'publicationTranBANK'
, @article = N'Account'
, @source_owner = N'dbo'
, @source_object = N'Account'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'Account'
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboAccount'
, @del_cmd = N'CALL sp_MSdel_dboAccount'
, @upd_cmd = N'SCALL sp_MSupd_dboAccount'
```

Appendix 9. Transactional subscriber SCHIPHOL script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationTranBANK'
, @subscriber = N'SCHIPHOL'
, @destination_db = N'subscriptionTranPullBANK'
, @sync_type = N'Automatic'
, @subscription_type = N'pull'
, @update_mode = N'read only'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'SCHIPHOL'-----
-
use [subscriptionTranPullBANK]
exec sp_addpullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationTranBANK'
, @publisher_db = N'Bank'
, @independent_agent = N'True'
, @subscription_type = N'pull'
, @description = N''
, @update_mode = N'read only'
, @immediate_sync = 1

exec sp_addpullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'Bank'
, @publication = N'publicationTranBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 64
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100428
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publication_type = 0
GO
-----END: Script to be run at Subscriber 'SCHIPHOL'-----

```

Appendix 10. Transactional subscriber ARLANDA script

```
-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationTranBANK'
, @subscriber = N'ARLANDA'
, @destination_db = N'subscriptionTranPushBANK'
, @subscription_type = N'Push'
, @sync_type = N'automatic'
, @article = N'all'
, @update_mode = N'read only'
, @subscriber_type = 0
exec sp_addpushsubscription_agent @publication = N'publicationTranBANK'
, @subscriber = N'ARLANDA'
, @subscriber_db = N'subscriptionTranPushBANK'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @subscriber_security_mode = 1
, @frequency_type = 64
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100428
, @active_end_date = 99991231
, @enabled_for_syncmgr = N'False'
, @dts_package_location = N'Distributor'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----
```

Appendix 11. Updatable transactional publication script

```

use [Bank]
exec sp_replicationdboption @dbname = N'Bank'
, @optname = N'publish'
, @value = N'true'
GO
use [Bank]
exec [Bank].sys.sp_addqreader_agent @job_login = N'atc\sqlroot'
, @job_password = null
, @job_name = null
, @frompublisher = 1
GO
-- Adding the transactional publication
use [Bank]
exec sp_addpublication @publication = N'publicationTranUpdatableBANK'
, @description = N'Transactional publication with updatable subscriptions of
database 'Bank'
from Publisher 'MALPENSA'.'
, @sync_method = N'concurrent'
, @retention = 0, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'true'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21
, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
, @repl_freq = N'continuous'
, @status = N'active'
, @independent_agent = N'true'
, @immediate_sync = N'true'
, @allow_sync_tran = N'true'
, @autogen_sync_procs = N'true'
, @allow_queued_tran = N'true'
, @allow_dts = N'false'
, @conflict_policy = N'pub wins'
, @centralized_conflicts = N'true'
, @conflict_retention = 14
, @queue_type = N'sql'
, @replicate_ddl = 1
, @allow_initialize_from_backup = N'false'
, @enabled_for_p2p = N'false'
, @enabled_for_het_sub = N'false'
GO

exec sp_addpublication_snapshot @publication = N'publicationTranUpdatableBANK'
, @frequency_type = 1
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1

use [Bank]
exec sp_addarticle @publication = N'publicationTranUpdatableBANK'
, @article = N'Account'
, @source_owner = N'dbo'
, @source_object = N'Account'

```

```
, @type = N'logbased'  
, @description = null  
, @creation_script = null  
, @pre_creation_cmd = N'drop'  
, @schema_option = 0x0000000008035CDF  
, @identityrangemanagementoption = N'manual'  
, @destination_table = N'Account'  
, @destination_owner = N'dbo'  
, @status = 16  
, @vertical_partition = N'false'  
GO
```

Appendix 12. Updatable transactional subscription ARLANDA script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationTranUpdatableBANK'
, @subscriber = N'ARLANDA'
, @destination_db = N'subscriptionTranUpdatablePushBANK'
, @subscription_type = N'Push'
, @sync_type = N'automatic'
, @article = N'all'
, @update_mode = N'queued failover'
, @subscriber_type = 0
exec sp_addpushsubscription_agent @publication = N'publicationTranUpdatableBANK'
, @subscriber = N'ARLANDA'
, @subscriber_db = N'subscriptionTranUpdatablePushBANK'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @subscriber_security_mode = 1
, @frequency_type = 64
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100428
, @active_end_date = 99991231
, @enabled_for_syncmgr = N'False'
, @dts_package_location = N'Distributor'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'ARLANDA'-----
use [subscriptionTranUpdatablePushBANK]
exec sp_link_publication @publisher = N'MALPENSA'
, @publisher_db = N'Bank'
, @publication = N'publicationTranUpdatableBANK'
, @distributor = N'BARAJAS'
, @security_mode = 0
, @login = N'sa'
, @password = null
GO
-----END: Script to be run at Subscriber 'ARLANDA'-----

```

Appendix 13. Updatable transactional subscription SCHIPHOL script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [Bank]
exec sp_addsubscription @publication = N'publicationTranUpdatableBANK'
, @subscriber = N'SCHIPHOL'
, @destination_db = N'subscriptionTranUpdatablePullBANK'
, @sync_type = N'Automatic'
, @subscription_type = N'pull'
, @update_mode = N'failover'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'SCHIPHOL'-----
-
use [subscriptionTranUpdatablePullBANK]
exec sp_addpullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationTranUpdatableBANK'
, @publisher_db = N'Bank', @independent_agent = N'True'
, @subscription_type = N'pull', @description = N''
, @update_mode = N'failover', @immediate_sync = 1
exec sp_link_publication @publisher = N'MALPENSA'
, @publication = N'publicationTranUpdatableBANK'
, @publisher_db = N'Bank'
, @security_mode = 0
, @login = N'sa', @password = null
exec sp_addpullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'Bank'
, @publication = N'publicationTranUpdatableBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 64
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100428
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publication_type = 0
GO
-----END: Script to be run at Subscriber 'SCHIPHOL'-----

```



```

-- Enabling the replication database
use master
exec sp_replicationdboption @dbname = N'BankCopy'
, @optname = N'merge publish'
, @value = N'true'
GO

-- Adding the merge publication
use [BankCopy]
exec sp_addmergepublication @publication = N'publicationMergeBANK'
, @description = N'Merge publication of database ''BankCopy'' from Publisher
'MALPENSA'..'
, @sync_mode = N'native'
, @retention = 14, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'true'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21
, @ftp_subdirectory = N'ftp'
, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
, @dynamic_filters = N'false'
, @conflict_retention = 14
, @keep_partition_changes = N'false'
, @allow_synctoalternate = N'false'
, @max_concurrent_merge = 0
, @max_concurrent_dynamic_snapshots = 0
, @use_partition_groups = null
, @publication_compatibility_level = N'100RTM'
, @replicate_ddl = 1
, @allow_subscriber_initiated_snapshot = N'false'
, @allow_web_synchronization = N'false'
, @allow_partition_realignment = N'true'
, @retention_period_unit = N'days'
, @conflict_logging = N'both'
, @automatic_reinitialization_policy = 0
GO

exec sp_addpublication_snapshot @publication = N'publicationMergeBANK'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 0
, @frequency_subday = 4
, @frequency_subday_interval = 2
, @active_start_time_of_day = 500
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1

use [BankCopy]
exec sp_addmergearticle @publication = N'publicationMergeBANK'
, @article = N'Account'
, @source_owner = N'dbo'
, @source_object = N'Account'
, @type = N'table'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'

```

```
, @schema_option = 0x000000010C034FD1
, @identityrangemanagementoption = N'manual'
, @destination_owner = N'dbo'
, @force_reinit_subscription = 1
, @column_tracking = N'false'
, @subset_filterclause = null
, @vertical_partition = N'false'
, @verify_resolver_signature = 1
, @allow_interactive_resolver = N'false'
, @fast_multicol_updateproc = N'true'
, @check_permissions = 0
, @subscriber_upload_options = 0
, @delete_tracking = N'true'
, @compensate_for_errors = N'false'
, @stream_blob_columns = N'false'
, @partition_options = 0
GO
```

Appendix 15. Merge subscriber ARLANDA script

```
-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [BankCopy]
exec sp_addmergesubscription @publication = N'publicationMergeBANK'
, @subscriber = N'ARLANDA'
, @subscriber_db = N'subscriptionMergePushBANK'
, @subscription_type = N'Push'
, @sync_type = N'Automatic'
, @subscriber_type = N'Global'
, @subscription_priority = 75, @description = null
, @use_interactive_resolver = N'False'
exec sp_addmergepushsubscription_agent @publication = N'publicationMergeBANK'
, @subscriber = N'ARLANDA'
, @subscriber_db = N'subscriptionMergePushBANK'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @subscriber_security_mode = 1
, @publisher_security_mode = 1
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 5
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100430
, @active_end_date = 99991231
, @enabled_for_syncmgr = N'False'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----
```

Appendix 16. Merge subscriber SCHIPHOL script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [BankCopy]
exec sp_addmergesubscription @publication = N'publicationMergeBANK'
, @subscriber = N'SCHIPHOL'
, @subscriber_db = N'subscriptionMergePullBANK'
, @subscription_type = N'pull'
, @subscriber_type = N'global'
, @subscription_priority = 74
, @sync_type = N'Automatic'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'SCHIPHOL'-----
-
use [subscriptionMergePullBANK]
exec sp_addmergepullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationMergeBANK'
, @publisher_db = N'BankCopy'
, @subscriber_type = N'Global'
, @subscription_priority = 74
, @description = N''
, @sync_type = N'Automatic'
exec sp_addmergepullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'BankCopy'
, @publication = N'publicationMergeBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 3
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100430
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1
, @publisher_login = null
, @publisher_password = null
, @use_interactive_resolver = N'False'
, @dynamic_snapshot_location = null
, @use_web_sync = 0
GO
-----END: Script to be run at Subscriber 'SCHIPHOL'-----

```

Appendix 17. Merge subscriber MALPENSA script

```

-----BEGIN: Script to be run at Publisher 'MALPENSA'-----
use [BankCopy]
exec sp_addmergesubscription @publication = N'publicationMergeBANK'
, @subscriber = N'MALPENSA'
, @subscriber_db = N'MergeTest'
, @subscription_type = N'pull'
, @subscriber_type = N'global'
, @subscription_priority = 75
, @sync_type = N'Automatic'
GO
-----END: Script to be run at Publisher 'MALPENSA'-----

-----BEGIN: Script to be run at Subscriber 'MALPENSA'-----
-
use [MergeTest]
exec sp_addmergepullsubscription @publisher = N'MALPENSA'
, @publication = N'publicationMergeBANK'
, @publisher_db = N'BankCopy'
, @subscriber_type = N'Global'
, @subscription_priority = 75
, @description = N''
, @sync_type = N'Automatic'
exec sp_addmergepullsubscription_agent @publisher = N'MALPENSA'
, @publisher_db = N'BankCopy'
, @publication = N'publicationMergeBANK'
, @distributor = N'BARAJAS'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 4
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 1
, @frequency_subday = 4
, @frequency_subday_interval = 3
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20100430
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1
, @publisher_login = null
, @publisher_password = null
, @use_interactive_resolver = N'False'
, @dynamic_snapshot_location = null
, @use_web_sync = 0
GO
-----END: Script to be run at Subscriber 'MALPENSA'-----

```

Appendix 18. Bank database backup script

```
BACKUP DATABASE [Bank] TO DISK = N'C:\Temp\bankBACKUP' WITH NOFORMAT
, NOINIT
, NAME = N'Bank-Full Database Backup'
, SKIP
, NOREWIND
, NOUNLOAD
, STATS = 10
GO
```

Appendix 19. Bank database restore script

```
RESTORE DATABASE [Bank] FROM DISK = N'C:\Temp\bankBACKUP'  
WITH FILE = 1, NOUNLOAD, STATS = 10  
GO
```

Appendix 20. Distributor and publisher MALPENSA script

```

use master
exec sp_adddistributor @distributor = N'MALPENSA'
, @password = N''
GO

exec sp_adddistributiondb @database = N'distributionBANK'
, @data_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_file_size = 2
, @min_distretention = 0
, @max_distretention = 72
, @history_retention = 48
, @security_mode = 1
GO

use [distributionBANK]
if (not exists (select * from sysobjects where name = 'UIProperties' and type =
'U '))
    create table UIProperties(id int)
if (exists (select * from ::fn_listextendedproperty('SnapshotFolder'
, 'user'
, 'dbo'
, 'table'
, 'UIProperties'
, null
, null)))
    EXEC sp_updateextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
else
    EXEC sp_addextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
GO

exec sp_adddistpublisher @publisher = N'MALPENSA'
, @distribution_db = N'distributionBANK'
, @security_mode = 0
, @login = N'sa'
, @password = N''
, @working_directory = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO

```


Appendix 21. Distributor and Publisher SCHIPHOL script

```

use master
exec sp_adddistributor @distributor = N'SCHIPHOL'
, @password = N''
GO

exec sp_adddistributiondb @database = N'distributionBANK'
, @data_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_file_size = 2
, @min_distretention = 0
, @max_distretention = 72
, @history_retention = 48
, @security_mode = 1
GO

use [distributionBANK]
if (not exists (select * from sysobjects where name = 'UIProperties' and type =
'U '))
    create table UIProperties(id int)
if (exists (select * from ::fn_listextendedproperty('SnapshotFolder'
, 'user'
, 'dbo'
, 'table'
, 'UIProperties'
, null
, null)))
    EXEC sp_updateextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
else
    EXEC sp_addextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
GO

exec sp_adddistpublisher @publisher = N'SCHIPHOL'
, @distribution_db = N'distributionBANK'
, @security_mode = 0
, @login = N'sa'
, @password = N''
, @working_directory = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO

```

Appendix 22. Distributor and publisher BARAJAS script

```

use master
exec sp_adddistributor @distributor = N'BARAJAS'
, @password = N''
GO

exec sp_adddistributiondb @database = N'DistributionBANK'
, @data_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data'
, @log_file_size = 2
, @min_distretention = 0
, @max_distretention = 72
, @history_retention = 48
, @security_mode = 1
GO

use [DistributionBANK]
if (not exists (select * from sysobjects where name = 'UIProperties' and type =
'U '))
    create table UIProperties(id int)
if (exists (select * from ::fn_listextendedproperty('SnapshotFolder'
, 'user'
, 'dbo'
, 'table'
, 'UIProperties'
, null
, null)))
    EXEC sp_updateextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
else
    EXEC sp_addextendedproperty N'SnapshotFolder'
, N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, 'user'
, dbo
, 'table'
, 'UIProperties'
GO

exec sp_adddistpublisher @publisher = N'BARAJAS'
, @distribution_db = N'DistributionBANK'
, @security_mode = 0, @login = N'sa'
, @password = N''
, @working_directory = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\ReplData'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO

```

```

use [Bank]
exec sp_replicationdboption @dbname = N'Bank'
, @optname = N'publish'
, @value = N'true'
GO
exec [Bank].sys.sp_addlogreader_agent @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1
, @job_name = null
GO
exec sp_addpublication @publication = N'publicationPeerBANK'
, @description = N'Transactional publication of database 'Bank' from Publisher
'MALPENSA'..'
, @sync_method = N'concurrent'
, @retention = 0
, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'false'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21
, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
, @repl_freq = N'continuous'
, @status = N'active'
, @independent_agent = N'true'
, @immediate_sync = N'false'
, @allow_sync_tran = N'false'
, @autogen_sync_procs = N'false'
, @allow_queued_tran = N'false'
, @allow_dts = N'false'
, @replicate_ddl = 1
, @allow_initialize_from_backup = N'false'
, @enabled_for_p2p = N'false'
, @enabled_for_het_sub = N'false'
GO
exec sp_addpublication_snapshot @publication = N'publicationPeerBANK'
, @frequency_type = 1
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = N'atc\sqlroot'
, @job_password = null
, @publisher_security_mode = 1
exec sp_addarticle @publication = N'publicationPeerBANK'
, @article = N'Account'
, @source_owner = N'dbo', @source_object = N'Account'
, @type = N'logbased', @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'Account'
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboAccount'
, @del_cmd = N'CALL sp_MSdel_dboAccount'
, @upd_cmd = N'SCALL sp_MSupd_dboAccount'

```