Chapter 10. October 2007

Welcome to the October 2007 edition of IBM Information Server Developer's Notebook. This month we answer the question;

How do I perform data replication with IBM Information Server?

To address the topic of data replication, we need first to define each of the following; What is data replication, why does one do data replication, and then in the larger part of this document, How does one perform data replication using IBM Information Server-

The topic of data replication also invites a discussion of relational database server transaction logging, and the data replication capabilities offered by these systems.

In short, we are going discuss all of the relevant terms and technologies above, and provide examples that detail how to deliver this functionality using IBM Information Server.

Software versions

All of these solutions were *developed and tested* on IBM Information Server version 8.01, on the Microsoft Windows XP/SP2 platform. IBM Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM Information Server product contains the following major components;

WebSphere Business Glossary™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, and Rational Data Architect™.

Additionally, on September 4, 2007, IBM completed acquisition of DataMirror Corporation™, considered by many to be the leader in real time change data capture (data replication). DataMirror becomes a component to IBM Information Server and the entire Information on Demand platform.

Obviously, IBM Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities. This month we give focus to the Data Mirror Transformation Server component of IBM Information Server.

10.1 Terms and core concepts

As an information technology term, *data replication* simply means to provide a redundant data set for some purpose; a copy of a single or set of data tables that reside in more than one location. The most common purposes for data replication include:

Ê Providing for *high availability*.

  A high availability data replication scenario might allow an on line transaction processing database to operate in a primary geographic location, and a mirror copy of that data set to operate in stand by mode in a another city far away. If the primary copy of the database fails for some reason, the mirror copy of the database can become active. A large geographic separation allows for availability across natural or man made disasters; earthquake, flood, fire, etcetera.

Ê Providing for *data co-location*, (expectedly to allow for higher business application performance).

  Given a set of numerous and remote point of sale locations, replicate the customer sale transaction data to the central office, allowing for more performant analysis of sales activity. (Rather than running numerous queries against tables at each of 300 remote sites, replicate the data to a central site, and run the collection of queries locally.)

Ê Providing for *audit trail capability*.

  On line transaction database systems excel at presenting data in its current state. One thing on line transaction processing systems do not do well is to present a temporal representation of data; meaning, display each state of a given record across the (n) times that record was changed today, this week, this month, and so on.

Ê Providing for *data aggregation and summation*.

  Given a data set of daily data, and the need to regularly report on weekly or monthly (totals), queries of this sort would be more performant when run against the smaller and even partially pre-calculated aggregate data set.

Ê Others.

Data replication schemes, trigger based replication

Most relational database server systems offer some degree and sophistication of built in data replication capability, and the technology used to deliver this database server capability can generally be categorized as *trigger based* or *transaction log based*. A SQL trigger offers the ability to perform some other, possibly unrelated, function as the result of any normal SQL INSERT, UPDATE, DELETE, SELECT, etcetera. Any end user application system is left unaware of the presence of triggers, which is one of the benefits in deploying triggers.

An example of the SQL data definition language (DDL) statements needed to put a manual trigger based audit trail data replication capability in place is listed in Example 10-1.

Example 10-1 SQL DDL Syntax to create trigger based audit trail capability.

```
-- Our actual single data table we wish to audit.

CREATE TABLE t1
  (
  col1 INTEGER      NOT NULL PRIMARY KEY,
  col2 INTEGER
  ) LOCK MODE ROW;


-- A table just to record INSERT, UPDATE, DELETE activity from table t1.

CREATE TABLE t1_audit
  (
  actionType        CHAR(03),
  actionTimestamp   DATETIME YEAR TO SECOND
    DEFAULT CURRENT YEAR TO SECOND,
  actionWho         CHAR(18) DEFAULT USER,
    --
  col1              INTEGER,
  col2              INTEGER
  ) LOCK MODE ROW;

-- SQL TRIGGER to record INSERTs

CREATE TRIGGER t1_insert
  INSERT ON t1
  REFERENCING new AS new
  FOR EACH ROW
    (
    INSERT INTO t1_audit (actionType, col1, col2)
      VALUES ("INS", new.col1, new.col2)
    );

-- SQL TRIGGER to record DELETEs

CREATE TRIGGER t1_delete
  DELETE ON t1
  REFERENCING old AS old
  FOR EACH ROW
    (
    INSERT INTO t1_audit (actionType, col1, col2)
      VALUES ("DEL", old.col1, old.col2)
    );

--  For SQL UPDATES, we want to record before-image of
--  record [and] post-image. We use a stored procedure
--  to allow us to, in effect, execute multiple INSERT
--  statements from one triggering action.

CREATE PROCEDURE p_t1_supportUpdate(old_col1 INTEGER,
    old_col2 INTEGER, new_col1 INTEGER, new_col2 INTEGER)
      --
  INSERT INTO t1_audit (actionType, col1, col2)
      VALUES ("OLD", old_col1, old_col2);
  INSERT INTO t1_audit (actionType, col1, col2)
      VALUES ("NEW", new_col1, new_col2);
END PROCEDURE;

CREATE TRIGGER t1_update
  UPDATE ON t1
  REFERENCING new AS new
          old AS old
  FOR EACH ROW
    (
    EXECUTE PROCEDURE p_t1_supportUpdate
      (old.col1, old.col2, new.col1, new.col2)
    );
```

As a result of executing the SQL DDL code displayed in Example 10-1, every SQL INSERT, DELETE, or UPDATE run against table t1 causes an audit trail record to be created in table t1_audit. Example as shown in Example 10-2.

Example 10-2 Recorded audit trail data from example above.

```
--  Assorted SQL DML statements to audit and observe.

insert into t1 values (1,1);
insert into t1 values (2,2);


--  This update affects zero rows.

update t1 set col2 = 4 where col1 = 6;


-- This update affects two rows.

update t1 set col2 = 8 where col1 > 0;
```

```
--  This delete affects 1 row.

delete from t1 where col1 = 1;


-- Request contents of audit trail table for example.

select * from t1_audit;


-- Sample output from t1_audit.

-- actionType    actionTimestamp     actionWho  col1  col2
--
-- INS           2007-10-29 09:10:42 EdSavage   1     1
-- INS           2007-10-29 09:10:42 EdSavage   2     2
-- OLD           2007-10-29 09:10:42 EdSavage   1     1
-- NEW           2007-10-29 09:10:42 EdSavage   1     8
-- OLD           2007-10-29 09:10:42 EdSavage   2     2
-- NEW           2007-10-29 09:10:42 EdSavage   2     8
-- DEL           2007-10-29 09:10:42 EdSavage   1     8
```

While the SQL DDL program code displayed in Example 10-1 could be viewed as some what complex, often the relational database server tooling offers a graphical interface to automate creation of this code. Other comments related to a trigger based data replication scheme are offered below:

Ê The biggest negative comment related to trigger based data replication is poor performance. In effect, every single row insert or delete causes a synchronous (blocking) foreground write of a second and redundant data row in the replicated table. In the case of an update, you have to record two inserts, one each for the before and then after image of the record.

  Using triggers, there is no way to make this write asynchronously (no way to make it faster).

  The example detailed above used an audit trail table that was adjacent to (local to) the source table. Performance gets measurably worse if one chooses to replicate to a remote table, as is the case with a high availability data replication.

Ê Other negative issues related to a trigger based data replication solution include:

  – One can try to record only a primary key value along with a code for SQL INSERT, SQL UPDATE, etcetera; this as opposed to recording the full width data record (don’t record all data columns, record only the primary key, and some sort of change code). This scenario can work to a point; consider for example the case of a new record insert that then gets updated. In this case, you would lose an earlier version of the inserted record, and receive only the post image, post update version of the record. What if you needed to have an application trigger present on both versions of the record-

If you choose to write only primary key values to the replicated or audit trail table, you then need to create and manage the execution of some sort of batch program to propagate the full width data record changes to the actual target table.

– The SQL program code above does not handle the case wherein primary keys are allowed to be updated. If it did, the SQL program code would be even larger and more complex.

– Triggers can be disabled manually by the administrator, or perhaps without notice when using advanced database server features like high performance bulk loaders and related.

– The database server vendor may not allow for trigger execution against remote or heterogeneous data sources.

– If you need triggers for some purpose other than data replication, you are then also left with the question and complexity of cascading or nested triggers, etcetera.

– Creating or modifying triggers often require that the SQL table be locked in exclusive or administrative mode for a period of time; exclusive access to the SQL table in order to create the new trigger, even if only for a few milliseconds.

– Trigger based data replication has no support for table (structure) modifications; meaning, if you add or drop columns, or change other aspects of the table definitions, you are left to manually modify any trigger based data replication code you have created.

– Others.

About the only positive features related to trigger based data replication are purchase cost and availability. Vendors often charge extra for data replication subsystems, whereas triggers are a no cost feature. It is often the case for heterogeneous data replication (replicate from Oracle to Microsoft SQL/Server, for example), that there is no vendor supported data replication capability other than to use triggers. If the vendor sells trigger based heterogeneous data replication, you are in effect paying for a graphical management console.

Data replication schemes, log based replication

Log based data replication has almost no negatives compared to trigger based data replication. Log based data replication is performed asynchronously to any normal SQL table activity (log based replication is non-blocking, and is highly performant). Because log based data replication capability is very advanced to program within the database server product itself (this is a vendor responsibility, not a customer or end user responsibility), many database server vendors do not offer this capability; their product simply has not advanced to this point.

Using IBM Information Server (IIS), more specifically the DataMirror Transformation Server component to IIS, you not only get log based data replication capability, you get it for heterogeneous database access (Oracle to Microsoft SQL/Server, and numerous other combinations of cross vendor products).

Note: *As an adjective, [Transformation] may be a misleading portion of the name for DataMirror Transformation Server. Historically, IBM DataStage, a component to IBM Information Server, was categorized as an ETL tool; ETL, Extract, Transform and Load. The transform capability to DataStage is huge. While DataMirror Transformation Server does provide basic transforms, perhaps a better name for DataMirror Transformation Server would be DataMirror Replication Server.*

Log based data replication is generally configured via a graphical user interface. In order to better understand the details of log based data replication, Figure 10-1 begins a discussion of general database server operations, as well as an overview of the role and responsibilities of the database server transaction log.

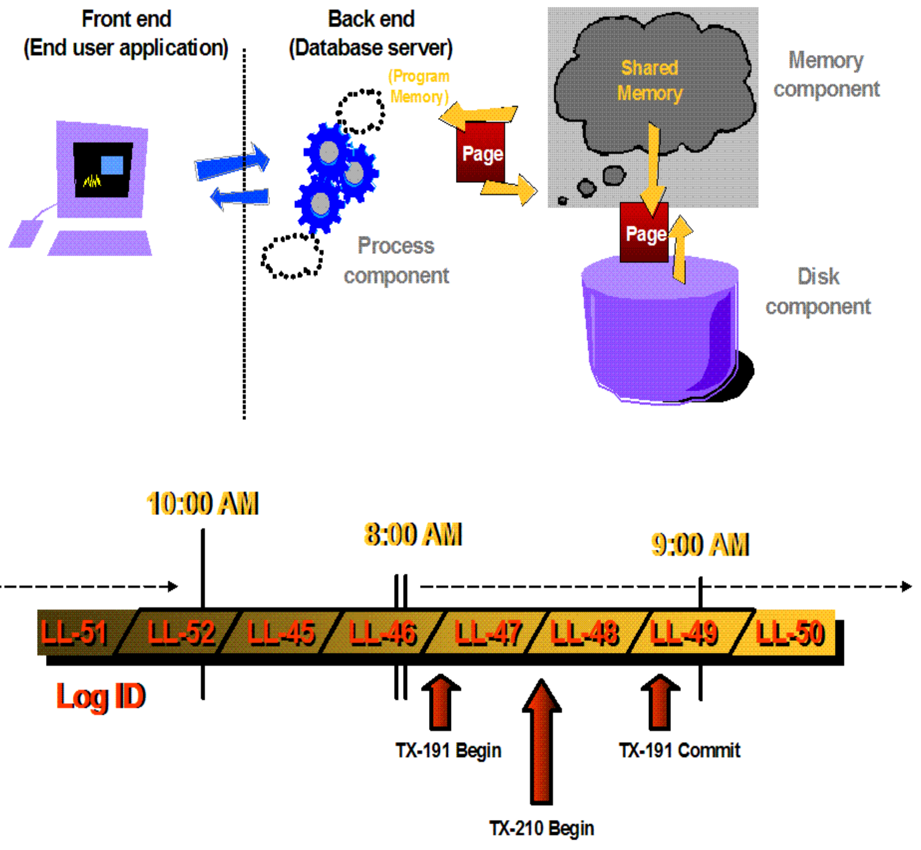


Figure 10-1   Database server system overview, including transaction log.

Concerning Figure 10-1, the following is offered:

Ê **The upper half of Figure 10-1 displays a basic database server design.** Every database server has a process component architecture, a memory component architecture, and a disk component architecture. The following is offered;

Process component to the database server-

– The process component of the database server is that collection of binary programs (processes) that perform the work of the database server.

  This work includes all of the work requested by the end user via the SQL command interface; reading records, updating records, creating indexes, and so on. This work also includes managing disk cache, writing and managing transaction log records, and related.

- A multi-threaded process architecture would have a finite number of processes that perform work on behalf of the end user applications. Database server systems of this type can perform their own advanced (process) scheduling, and generally offer the most performant systems. IBM/Informix Dynamic Server (IDS) offers this type of process architecture.

  A two process architecture database server has a finite number of system processes, and then adds an additional process for each end user connection. Oracle, and lower end relational database servers like MySQL offers this type of process architecture.
- A key point related to the process component of a database server is that this is the only part of the server which is running; memory and disk only exist as allocated resource. The database server process component manages (executes) the movement of data from the end user into memory, and then as needed, from memory to and from the disk.

Memory component to the database server-

- There are many adjectives applied to memory on a computer system.
  - *Shared memory* is a special type of memory that is managed by the operating system or its supplied libraries. Shared memory is accessible to many concurrent processes. A locking mechanism insures, for example, that two users can not write to the same portion of shared memory at the same time, thus preserving data integrity.

    The advantage of shared memory is that many processes can gain performance on tasks that execute in parallel; for example, two or more concurrent processes can build an index cooperatively, viewing and changing a shared new index structure. In this case, shared memory is like shared program memory, as opposed to a standard file system disk cache, which is also shared.

    In general, there will be several named and configurable pools within shared memory. One or more of these pools are dedicated to that information which is written to the transaction log file.
  - *Program memory* is not shared between processes, and is more like the memory one thinks of when discussing computer memory. Program memory is allocated on demand, to a single computer program (process).
  - Others.
- Generally it is stated that the process component to the database server moves data to and from the end user application into shared memory, where the data remains. Other process components then move the data from shared memory to and from the disk component.

This asynchronous processing of the movement of data allows for higher performance, as requests for disk i/o can be cached until the latest possible moment in time.

When the end user requests that a unit of work be committed, the database server will perform a hard i/o request, confirming that only the instructions the end user has requested are flushed to the hard disk, while the data proper remains cached. Program control is returned to the end user when the database server can ensure that the requested transaction is recoverable from any means of computer failure.

Generally the instructions that modified a given data set are measurably smaller than the data proper; for example, one SQL UPDATE statement can effectively change thousands of records.

Disk component to the database server-

– In addition to the obvious structures on the disk like SQL tables, SQL indexes, the SQL data dictionary, and so on, the database server also maintains several internal or control structures. One of these structures is the transaction log file. Generally it is stated;

  - The database server writes the data pages which are created or modified by the end user's request for SQL service. Additionally, the database server records the instructions that created or modified these data pages in a transaction log file, an activity which may initially be viewed as some what redundant.
  - While the transaction log file seems redundant, it really offers a great savings of resource within the database server environment. The instructions to create or modify data are measurably smaller than the data itself. As mentioned above, one 80 byte SQL UPDATE statement can effectively change thousands of records (millions of bytes).

    By recording the instructions that modified the data proper, the database server can indiscriminately overwrite a given data page (buffer) numerous times; it can do this because the instructions to recover any version of a given data page exist in the transaction log file.
  - Generally, each structure on the disk is fronted by a buffer pool of some sort. In the case of the transaction log file, the database server only needs to flush this buffer when the end user requests a commit or roll back of a unit of work. (The database server does not need to flush the data pages themselves.)

    The buffer to the transaction log file may be shared by all users, and may be written in a pre-sorted big buffer write, causing a single or small number of physical disk i/o calls.

- The transaction log file may be viewed as one logical structure, or as a number of smaller disk space allocations operating as an organized unit. The transaction log may be viewed as, for example, ten (count) one megabyte disk space allocations, or one (count) logical ten megabyte disk space allocation.

  The database server writes to one transaction log file, fills its, and then switches to the next transaction log file in the rotation.

  Some database servers do not offer an actual rotation of transaction log files, and choose instead to create new transaction log files in a given directory; the result is the same.

  In any case, the database server labels each new use of a transaction log file with a unique identifier, some sort of transaction log number, or log id. For the remainder of this document, we will imagine the transaction log file to be composed of n (count) individual transaction log files, and we will use the terms file and files indiscriminately.

Ê **The bottom half of Figure 10-1 displays a transaction log file and its sample use.**

- It was stated above that transaction log files are used in a rotation (used cyclically) and are numbered sequentially with a unique identifier; the example from Figure 10-1 being, LL-45, LL-46, and so on.

  Figure 10-1 displays transaction log files number 45 through 52. The example in Figure 10-1 supposes that unique log identifiers 44 and earlier were used and overwritten before this graphic image, and are no longer contained within the database server system.

  In Figure 10-1, transaction log file 45 is the oldest log file on the database server system. When transaction log file 45 became full, the database server began writing to transaction log file 46, and so on.

  As would always be the case, the database server is currently writing to the transaction log file with the largest, most recent numbered identifier. In Figure 10-1, this is transaction log file number 52. When transaction log file 52 becomes full, the database serve will over-write transaction log file number 45 and label this as transaction log file 53 (not pictured).

- When a transaction log file becomes full, the database server switches to the next transaction log file. There are adjectives used to describe a given transaction log file and its given status, to include;

  - The transaction log file that is having transaction data written to it is called the *current log*. There can be only one current log at one time.

    Minus the exception case of a newly created transaction log file (an administrative special event), transaction log files are labeled as either being the current log, or a *used log*. In Figure 10-1, transaction log files 45 through 51 are used.

Note: *A transaction log file is labelled as either current or used, one or the other not both, and only one transaction log file may be current.*

- After a transaction log file has been written to (used), and the database server system has switched to the next transaction log file in the rotation, the used log may be backed up, presumably to a magnetic tape device of some kind.

  Even though the used transaction log file may contain the beginning or continuation of a specific transaction whose final status is not known (the given transaction has not yet committed or rolled back), that transaction log file is full, and its contents will not change. This is what allows the given transaction log file to be written elsewhere.

Note: *A transaction log file is labelled as either backed up, or not backed up. All used logs may be backed up or not backed up. The current log could never be backed up; it is still changing because it is currently being written to.*

- As is implied in this discussion, transaction log files contain end user transaction information, received from the end users in the form of a SQL BEGIN WORK command, various SQL commands to modify data, and then an expected SQL COMMIT WORK command or SQL ROLLBACK WORK command. Each of these transactions have a beginning and an end, and can span one or more transaction log files.

  If a given transaction log file contains only completed transactions (transactions that have either committed or rolled back), and that transaction log file has been backed up, then that transaction log file can be overwritten, and essentially re-used; (the disk space allocation for the given transaction log file will receive a new identifier, and will be used to record newly received end user transaction data).

  In Figure 10-1, two end user initiated transactions are displayed, TX-191, and TX-210. TX-191 began in transaction log file 47 and completed in transaction log file 49. TX-210 begins in transaction log file TX-48 and remains open (no completion to this transaction is displayed). As a result, transaction log files 45 through 47 contain only completed transactions, and if these transaction log files are backed up, then they are available for re-use. Transaction log files 48 and beyond contain work for transaction TX-210 which is not yet completed; transaction log files 48 and beyond are said to be *active*.

Note: *A transaction log file is said to be active if it contains any portion of a transaction that has not yet completed. An active transaction log can not be overwritten, as it is needed for transaction recovery in the case of database server system failure or related function.*

Log based data replication works by having a process agent scan (read) the database server transaction log file for new entries, and then processing (interpreting and applying) those entries against the target database server system or tables. The nature of a task like reading the transaction log files naturally allows for the process agent reading the transaction files to be part of the database server system itself, or provided by another (external) piece of software like IBM Information Server (IIS), more specifically, the DataMirror Transformation Server (DM/TS) component to IIS.

While the DataMirror Transformation Server (DM/TS) component to IBM Information Server itself offers a discussion of many advanced and highly performant architectural design optimizations and related, at this point, this document is going to proceed with a brief exercise to deliver (design and make active) data replication using DM/TS. Part of the current IBM plan, subject to change, is to make DM/TS available as a data source on the parallel canvas to DataStage, example as shown in Figure 10-2.

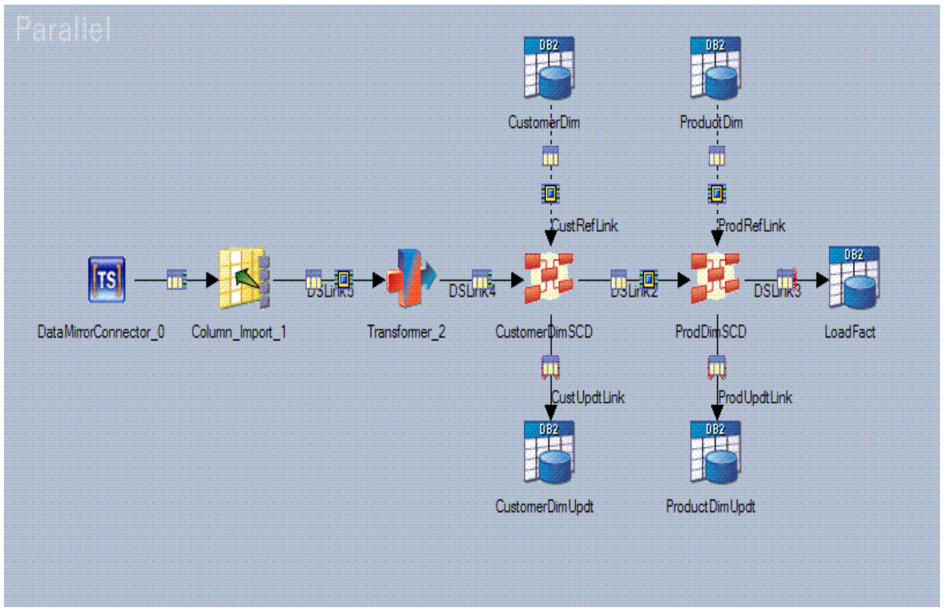


Figure 10-2   Prototype of DataMirror connector on DataStage parallel canvas.

10.2 Replication with DataMirror Transformation Server

At the 2007 IBM Information on Demand conference and in a public forum, IBM demonstrated the DataMirror Transformation Server (DM/TS) integrated within the DataStage Designer program, with DM/TS acting as a data source provider to DataStage Jobs. (See Figure 10-2, above, captured from that demonstration.) Subject to change, this program capability is expected to be delivered in fourth quarter of 2007, possibly first quarter of 2008. For this section of this document, we are going to use all currently available software, and use the existing DM/TS graphical programs to deliver a high availability data replication scenario. The following is offered:

- DataMirror Transformation Server (DM/TS), now a component to IBM Information Server, allows for many advanced data replication scenarios, including those outlined above; audit trail, high availability, data co-location, and so on.

  DM/TS also allows for heterogeneous data replication, for example, Oracle to/from Microsoft SQL/Server, SQL/Server to/from DB2 UDB, and so on.
- A DataMirror Transformation Server demonstration CD is available from an IBM Information Management sales representative which would allow one to complete the example below.
- In its history, DataMirror acquired PointBase, a small foot print and easily embedded relational database similar to Cloudscape/Derby, and also an open source database server. PointBase ships with (is included in) the DataMirror Transformation Server demonstration CD. To keep the examples that follow as simple as possible, we are going to demonstrate PointBase to PointBase high availability data replication.
- Also to keep these examples as simple as possible, we are not going to detail or otherwise cover the topic of installing DataMirror Transformation Server; as a typical graphical install, this shouldn’t cause any grief.

Software install, overview of instructions.

The following needs to be completed from the DataMirror Transformation Server demonstration CD, and is presented as an overview:

- Per this example, all of the software discussed in this document is installed on your workstation. Since all of the installed software is operating locally to one workstation, all of the IP addresses below are the same, whatever that IP address is.
- Each of the following binary installer programs need to be run, in the order listed below, and choosing only default values:
  - **dmaccesscontrol (version 6) setup.exe**

This program installs the DataMirror Access Control Server, used to validate user log on, roles, and permissions within the DataMirror component suite. A separate install, this server could be installed anywhere on the corporate network.

For this example, you would install this server on your workstation. The default port in use by the DataMirror Access Control Server is 10101.

- On the Microsoft Windows platform, the binary program representing the DataMirror Access Control Server is entitled, dmaccessserver.exe, and appears in the Windows Services administration program (services.mcs) as, **DataMirror Transformation Server Access Server**.

  This program/service should be running.

- Another port is in use by the **DataMirror Integration Server**, port 10102. This component to DataMirror Transformation Server is not discussed here.

  This program/service should be running.

– **dmclient (version 6) setup.exe**

This program installs the DataMirror two-tier, graphical designer/ management client. This install is performed on every designer or administrator workstation.

For this example, you again install this piece only on your workstation.

– **demotool (version 6) setup.exe**

This program installs the DataMirror Transformation Server demo tool.

The demo tool offers a very complete and useful training example towards learning and/or evaluating DataMirror Transformation Server. This demo tool can be used to work through an example with 60 or so pages of instruction. For brevity, we are detailing a smaller example in this document.

We install this portion of the demonstration CD for two reasons;

- It installs the PointBase relational database server for us. This gives us a consistent (shared) database platform so that persons reading this document can benefit from a common reference.

- It also makes two databases that we can set up high availability data replication between. While you can replicate between two tables inside the same database, we will use two databases which is a more common use case.

This step provides the database server software and is installed on your workstation.

With the install of this demo tool, two PointBase databases are made with the names, pointbase1 and pointbase2.

- PointBase1 uses port 9092 to listen for new database connections.

  On the Microsoft Windows platform, this binary program is entitled, PointBase1.exe, and appears in the Windows Services administration program as, DataMirror TS Demo PointBase1.

  This program/service should be running.

- PointBase2 uses port 9096.

  This is binary program, PointBase2.exe, and service entry, DataMirror TS Demo PointBase2.

– **ts-pointbase (version 6) setup.exe**

  This program installs the DataMirror Replication Agent that is specific to the PointBase database.

  This step is actually performed twice, once for the source database (pointbase1) and once for the target database (pointbase2). Remember that we are replicating between two databases, and that these databases are normally on separate hosts, and could even be database servers from different vendors.

  This binary install program name differs based on the type of database being used; ts-pointbase is in the name of this binary program which supports PointBase databases, for Oracle the name is different, Sybase, MS/SQLServer, and so on.

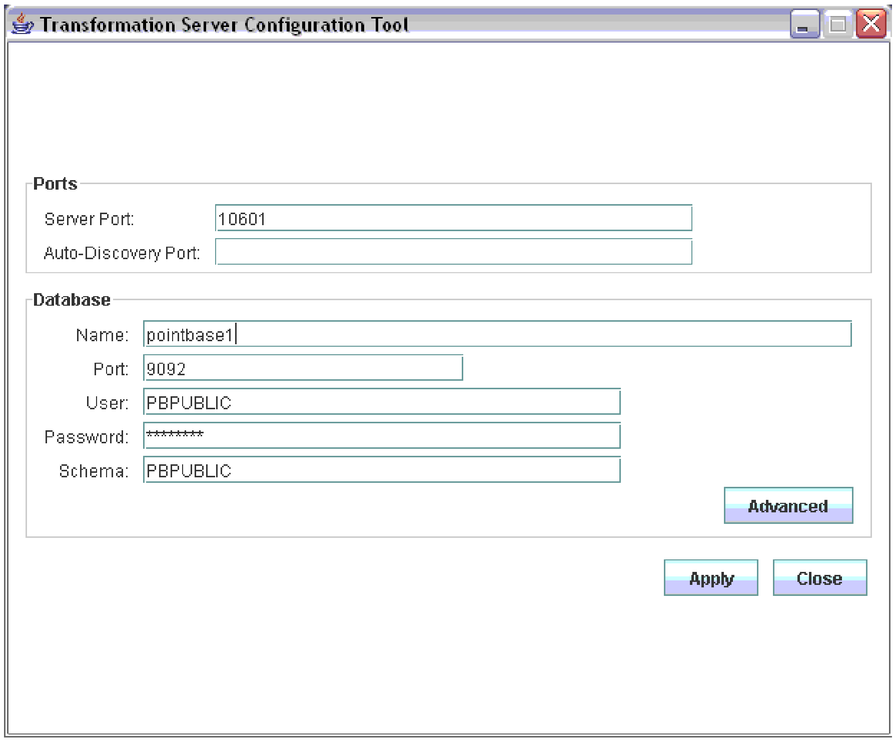
  - Again, this step is performed twice. When prompted for the name of the DataMirror Transformation server (a unique identifier of our choosing), we supplied the values, DMTS_PB1, and then DMTS_PB2.

  - After this install completes, you are prompted to configure the given instance of the DataMirror Transformation Server; in general, you supply the port number of this server (you already specified its DM/TS name above), and you supply the connection information for the related database server.

    An example of this dialog box is displayed in Figure 10-3.

    The user, password, and schema name values are all PBPUBLIC. These values were set by the install of the demo tool above

    Again, this step is performed twice, once for the source database with its given parameters, and then a second time for the target database. When completing this step for the target database, use database name pointbase2, and port number 10602.



Ports

Server Port: 10601

Auto-Discovery Port:

Database

Name: pointbase1

Port: 9092

User: PBPUBLIC

Password: ********

Schema: PBPUBLIC

Figure 10-3 DataMirror Transformation Server, Install, Configuration dialog box.

- After you enter the values displayed in Figure 10-3, click Apply.

  If you prompted to start any daemon processes, select Yes.

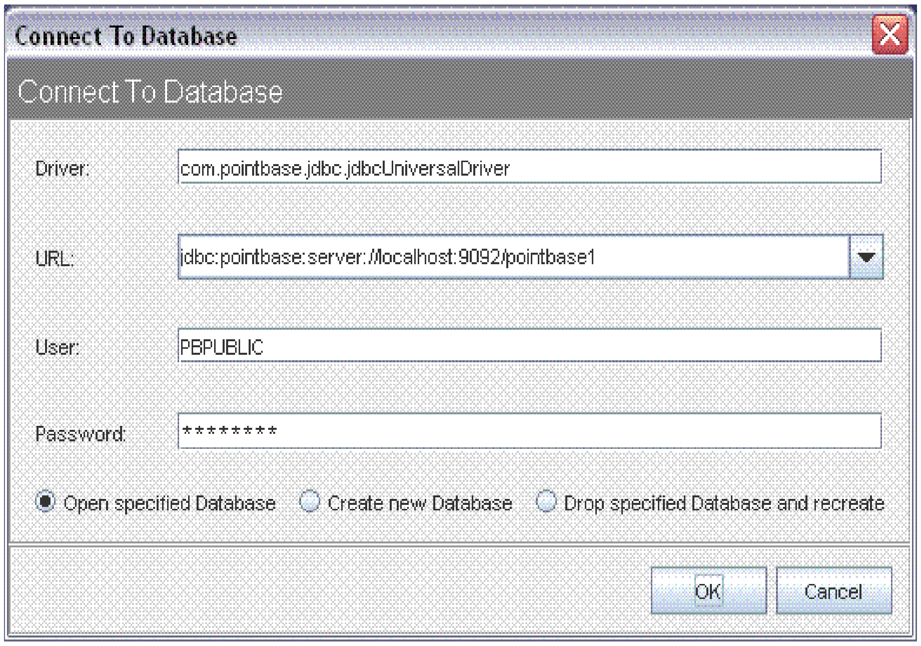
  Click Close to complete this dialog box and this step.

Preparing the databases for this example.

At this point, the sample databases created with the install of the demo tool above are both empty; they each contain no tables and no data. In this section of this document, we use the PointBase graphical administration client to create a single table in both databases (one source and one target), and seed the source table with some initial data.

Ê Run the PointBaseConsole.exe program. This is the graphical administration client to the PointBase database.

This action will produce the PointBase, Connect to dialog, example as shown in Figure 10-4.



Driver: com.pointbase.jdbc.jdbcUniversalDriver
URL: jdbc:pointbase:server://localhost:9092/pointbase1
User: PBPUBLIC
Password: *********

(●) Open specified Database ( ) Create new Database ( ) Drop specified Database and recreate

Figure 10-4 PointBase graphical administration program, connect dialog.

- Manage the display in Figure 10-4, set accurately set the port number (9092) and the database name, (pointbase1).
- The user name and password are both PBPUBLIC, in uppercase.
- Click OK to connect to the pointbase1 database.
- In the PointBase graphical administration screen for the pointbase1 database, complete the following;
  - Manage the graphical display to create a SQL table equal to;

    CREATE TABLE t1 (col1 INTEGER NOT NULL PRIMARY KEY, col2 INTEGER);
  - Insert several rows of sample data equal to;

    INSERT INTO t1 VALUES(1, 1);

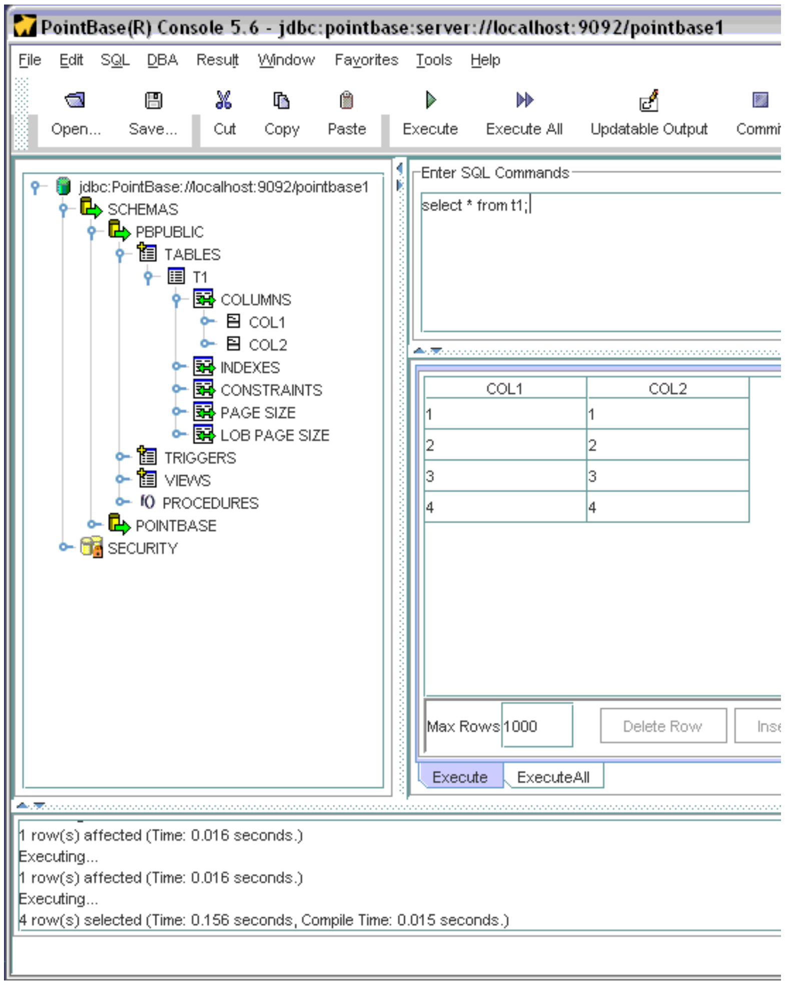
    INSERT INTO t1 VALUES(2, 2);

    INSERT INTO t1 VALUES(3, 3);

    INSERT INTO t1 VALUES(4, 4);
  - To confirm your work, run a SQL SELECT equal to;

    SELECT * FROM T1;

You should receive a response equal to what is displayed in Figure 10-5.


PointBase(R) Console 5.6 - jdbc:pointbase:server://localhost:9092/pointbase1
File Edit SQL DBA Result Window Favorites Tools Help
Open... Save... Cut Copy Paste Execute Execute All Updatable Output Commi
jdbc:PointBase://localhost:9092/pointbase1
SCHEMAS
PBPUBLIC
TABLES
T1
COLUMNS
COL1
COL2
INDEXES
CONSTRAINTS
PAGE SIZE
LOB PAGE SIZE
TRIGGERS
VIEWS
PROCEDURES
POINTBASE
SECURITY
Enter SQL Commands
Max Rows 1000
Delete Row
Inse
Execute
ExecuteAll

```
select * from t1;
```

COL1	COL2
1	1
2	2
3	3
4	4

```
1 row(s) affected (Time: 0.016 seconds.)
Executing...
1 row(s) affected (Time: 0.016 seconds.)
Executing...
4 row(s) selected (Time: 0.156 seconds, Compile Time: 0.015 seconds.)
```

Figure 10-5 PointBase graphical administration screen, with SQL SELECT results.

Ê Run a second invocation of PointBaseConsole.exe, and connect to the second database; port (9096), and database name, (pointbase2).

In the PointBase graphical administration screen for the pointbase2 database, complete the following;

– Manage the graphical display to create a SQL table equal to;

CREATE TABLE t1 (col1 INTEGER NOT NULL PRIMARY KEY, col2 INTEGER);

– Do not insert any sample data into this table.

  To confirm your work, run a SQL SELECT equal to;

  SELECT * FROM T1;

  You should receive a response with no data (zero rows) returned.

Note: *Leave these two PointBase graphical administration screens open. These screens will come in handy later for testing.*

Configure DataMirror Access Control Server.

Thus far we have performed the following;

Ê Installed the software for the DataMirror Transformation Server (DM/TS) Access Control Server.

Ê Installed the software for the DM/TS graphical designer/management client.

Ê Installed the DM/TS demo tool.

  This step also installed the PointBase database server, and created two databases.

Ê Installed the DM/TS server (replication agent) for the PointBase database.

  We did this step twice, once for the source database and once for the target. This step also prompted us to configure this server; its name, its port, and connection information for the given source/target database.

Ê We created a table in both the source and target databases, and inserted sample data into the source.

All of the previous steps were preparatory in nature; installing software and related. We have one remaining step that is also preparatory, configuring the DataMirror Transformation Server (DM/TS) Access Control server. In effect, the DM/TS Access Control server will record metadata from our previous steps, and allow named users (administrators) to manage these resources.

Complete the following;

1. Run the DM/TS Access Control server console.

   The icon for this program is entitled, Access Manager, and its binary program name is, dmaccessmanager.exe.

2. Inside the DM/TS Access Control server console, perform the following;

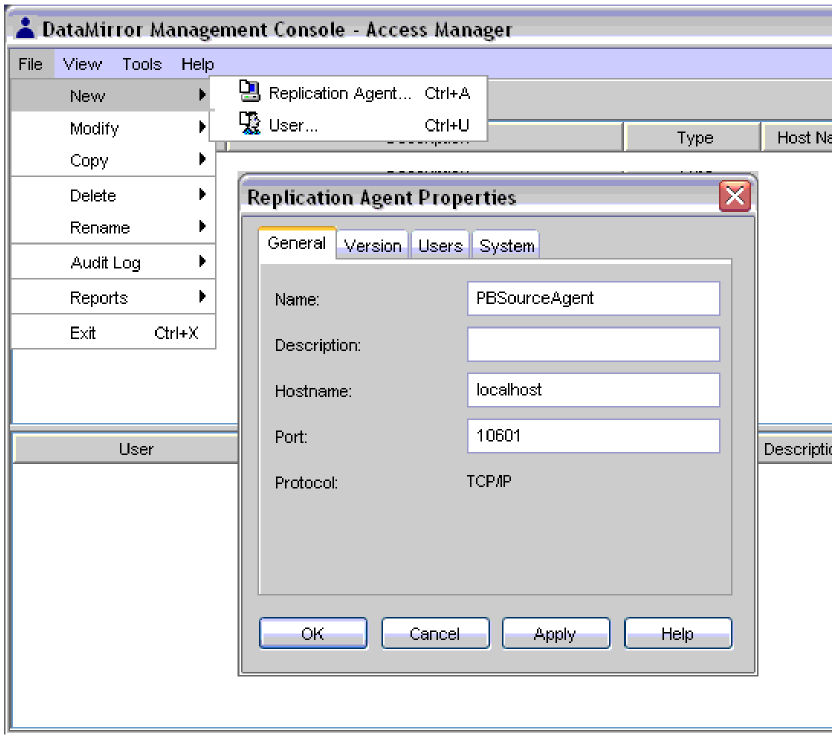
   a. From the menu bar, select, File -> New -> Replication Agent.

      This action produces the dialog box as displayed in Figure 10-6.

   b. Under the General TAB, name this Replication Agent, PBSourceAgent.

c. Enter a Hostname of, localhost, and Port number of, 10601.

This is the port number we entered when we installed the software for this replication agent earlier.



Name: PBSourceAgent
Description:
Hostname: localhost
Port: 10601
Protocol: TCP/IP

Figure 10-6   Adding metadata for replication agent, step 1 of 2.

d. Ensure that your dialog box equals what is displayed in Figure 10-6, then move to the Version TAB. Example as shown in Figure 10-7.

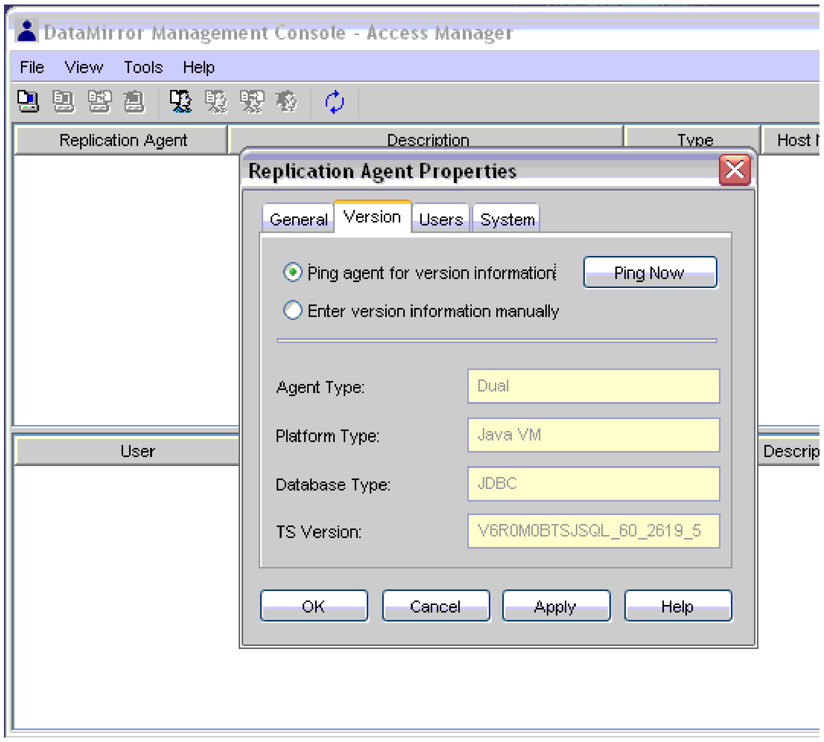
e. Select the radio button entitled, Ping agent for version information, then click the button entitled, Ping Now.

f. Click OK to save changes and close this dialog box.

3. Repeat Step 2above-

a. Name this replication agent, PBTargetAgent.

b. This agent uses port number, 10602.



(●) Ping agent for version information
( ) Enter version information manually

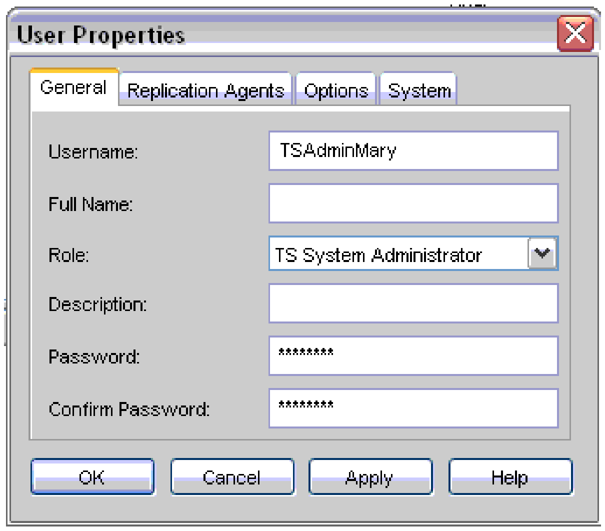
Agent Type: Dual
Platform Type: Java VM
Database Type: JDBC
TS Version: V6R0M0BTSJSQL_60_2619_5

Figure 10-7 Adding metadata for replication agent, step 2 of 2.

At this point we have added metadata for our two replication agents. Now we need to create a user (administrator) that can manage these resources.

4. From the menu bar, select, File -> New -> User.
   a. This action produces the User Properties dialog box, displayed in Figure 10-8.
   b. In the General TAB, enter a Username and password.

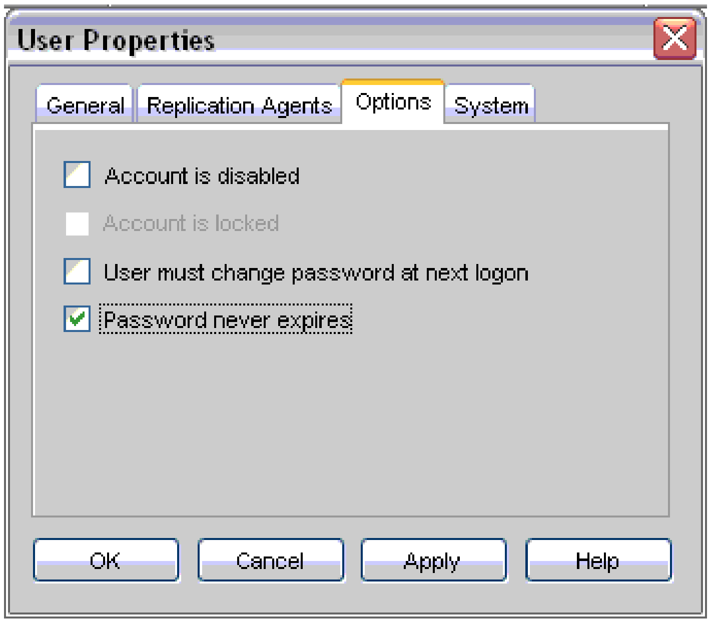
Note: *In the context of the DataMirror Access Control server, we are currently recording metadata for two Replication Agents. Later, in the context of the DataMirror Management Console, these Replication Agents take on additional properties, and are then referred to as Datastores.*



Username: TSAdminMary
Full Name:
Role: TS System Administrator
Description:
Password: ********
Confirm Password: ********

Figure 10-8 Adding a new user (administrator), General TAB.

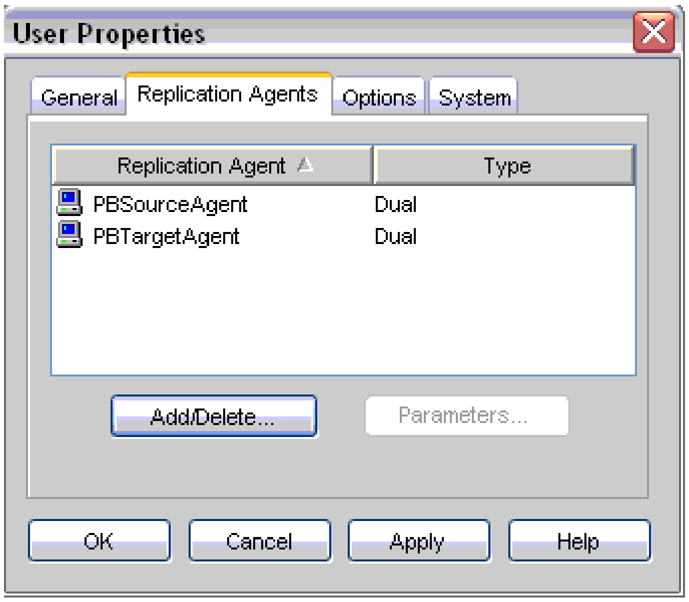
c. In the Options TAB, check the visual control so that the, Password never expires. Example as shown in Figure 10-9.



- [ ] Account is disabled
- [ ] Account is locked
- [ ] User must change password at next logon
- [✓] Password never expires

Figure 10-9 Adding a new user (administrator), Options TAB.

d. In the Replication Agents TAB, click the Add/Delete button and add both of the replication agents we defined in steps 2 and 3 above. Example as shown in Figure 10-10.



Replication Agent	Type
PBSourceAgent	Dual
PBTargetAgent	Dual

Figure 10-10 Adding a new user (administrator), Replication Agents TAB.

We're nearly done, now each of these two replication agents must have their database connection properties set.

e. In Figure 10-10, highlight first the entry entitled, PBSourceAgent.

This action will enable/un-grey the Parameters button.

Click the Parameters button.

This action produces the display in Figure 10-11.

i. The hardest visual control to set in Figure 10-11 is the Database/URL value.

For the source database, this value is-

jdbc:pointbase:server://localhost:9092/pointbase1

For the target database, this value is-

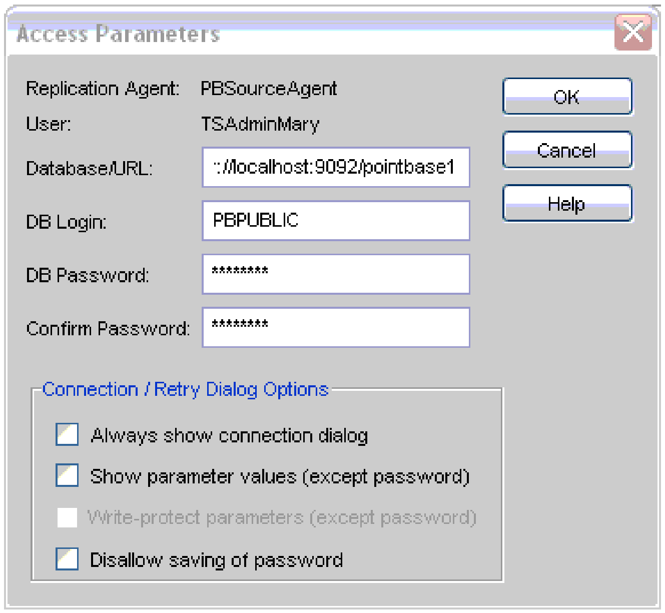
jdbc:pointbase:server://localhost:9096/pointbase2

You can reverse engineer this value from the Connect dialog box of the PointBase graphical administration client, displayed in Figure 10-4.

PointBase is a Java database, hence, this value is a connection URL. For Oracle, MS/SQLServer, etcetera., this dialog box will differ and request parameters that are specific to those vendors.

ii. The DBLogin and DB Passwords are both, PBPUBLIC.

iii. Click OK when done.


Access Parameters

Replication Agent: PBSourceAgent
User: TSAdminMary
Database/URL: ://localhost:9092/pointbase1
DB Login: PBPUBLIC
DB Password: ********
Confirm Password: ********

Connection / Retry Dialog Options
- Always show connection dialog
- Show parameter values (except password)
- Write-protect parameters (except password)
- Disallow saving of password

Figure 10-11 Setting database connection properties for replication agent.

f. Repeat "step e" above for the PBTargetAgent.

If you get warnings about violating password policies, ignore them.

g. At this point we are done with the DataMirror Transformation Server Access Server console. Exit this program.

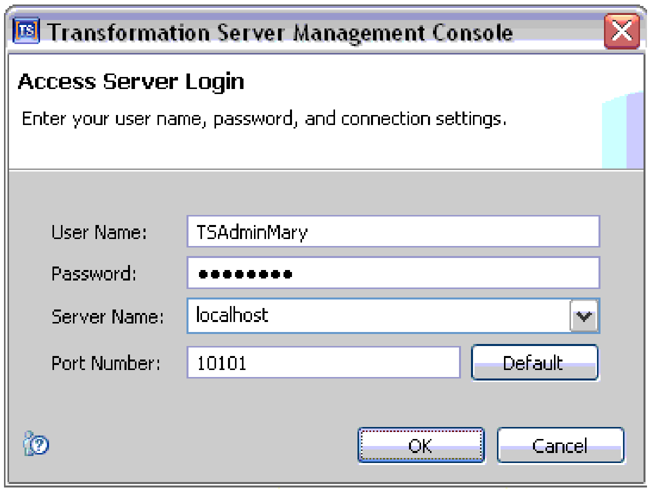
Configure replication activities proper.

All preparatory work is now complete. At this point, all remaining tasks are performed within the DataMirror Transformation Server (DM/TS) graphical designer/management client.

5. Run the DM/TS Management console.

The icon for this program is entitled, Management Console, and its binary program name is, DmClient.exe.

This action will produce the DataMirror Management Console Access Server Login dialog box displayed in Figure 10-12.


Transformation Server Management Console

Access Server Login

Enter your user name, password, and connection settings.

User Name: TSAdminMary

Password: ●●●●●●●●●

Server Name: localhost

Port Number: 10101

Figure 10-12   DM/TS Management Console, Server Login dialog box.

a. In Figure 10-12, enter the user credentials we created in step 4 above.

   The Server Name and Port Number fields refer to the DataMirror Access Control server; this is the server that authenticates our user credentials, which is the activity we are performing currently.

b. Click OK to gain access to the DT/MS Management console.

6. Add tables to a Datastore (Datastore PBSourceAgent).

   A Datastore is a logical term, and represents a grouping of physical entities, namely a set of tables.

   When we launch the DM/TS Management Console, we see two Datastores; PBSourceAgent, and PBTargetAgent. These Datastores were created for us by steps 2 and 3 above, when we used the DM/TS Access control server.

   It is at this time we are going to modify these Datastores, adding tables and related information to these entities.

   a. In the DT/MS Management Console, select, Configuration TAB -> Datastores TAB.

   b. In the upper left portion of this display (Datastores view), highlight the DataStores entry entitled, PBSourceAgent with a single click.

      This action makes this entry current.

   c. In the Source Tables view, click the button entitled, Add or Remove Tables.

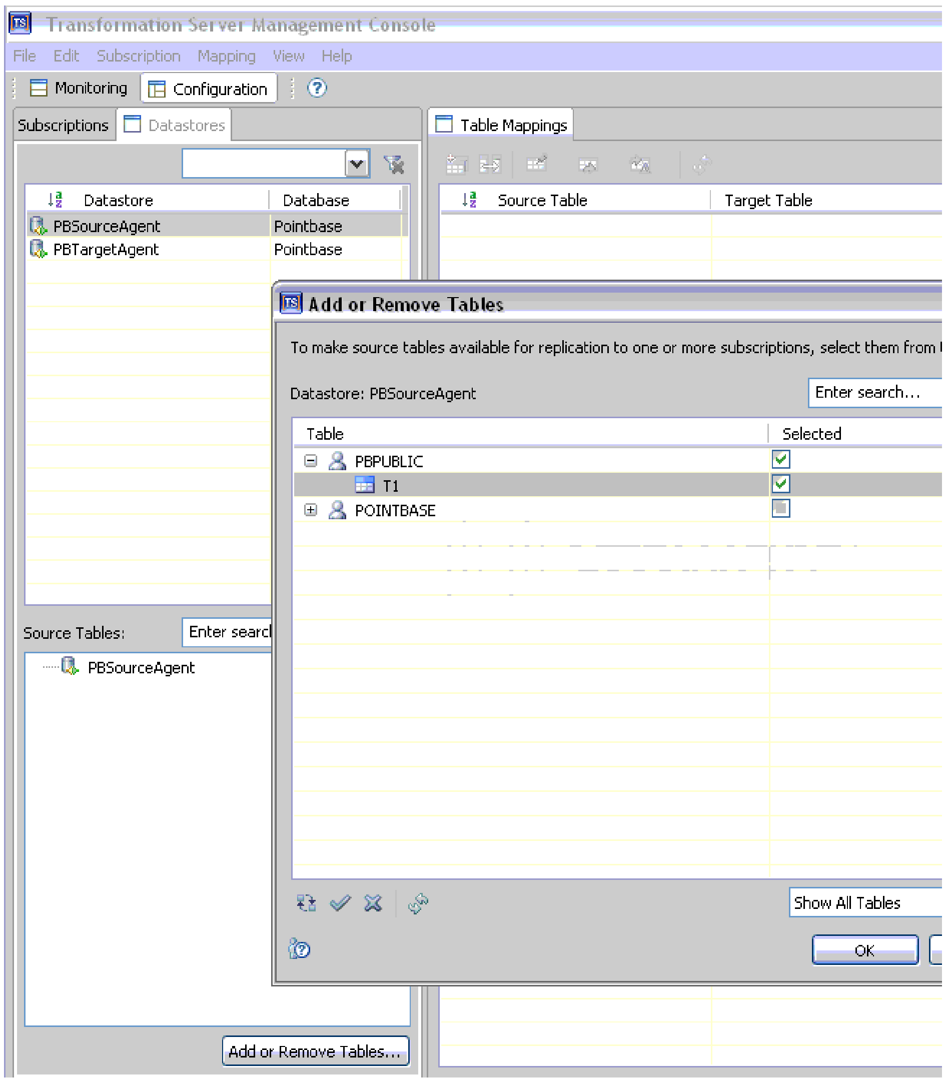
This action produces the Add or Remove Tables dialog box.

d. Manage the display so that you can check (select) the table PBPUBLIC.t1.

PBPUBLIC is the name of the SQL Schema that contains our SQL table. As a result of selecting this table, the schema is also selected.

e. Click OK when you are done.

Example as shown in Figure 10-13.



To make source tables available for replication to one or more subscriptions, select them from

Datastore: PBSourceAgent

Enter search...

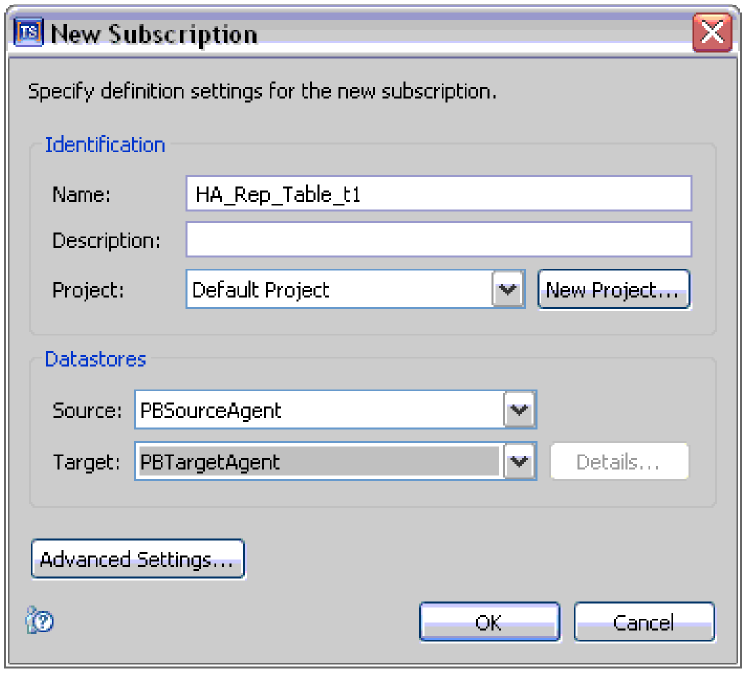
Figure 10-13   *Adding tables to a Datastore.*

7. Repeat step 6 above for PBTargetAgent; again we are adding table PBPUBLIC.t1.
8. Define a Subscription.

   A Subscription is

   a. In the DT/MS Management Console, select, Subscriptions TAB.
   b. In the Subscriptions view, right-click to produce a context sensitive menu, and select, New Subscription.

      This action produces the New Subscription dialog box. Example as shown in Figure 10-14.


New Subscription

Specify definition settings for the new subscription.

Identification

Name: HA_Rep_Table_t1

Description:

Project: Default Project

Datastores

Source: PBSourceAgent

Target: PBTargetAgent

Figure 10-14 DM/TS Management Console, New Subscription dialog box.

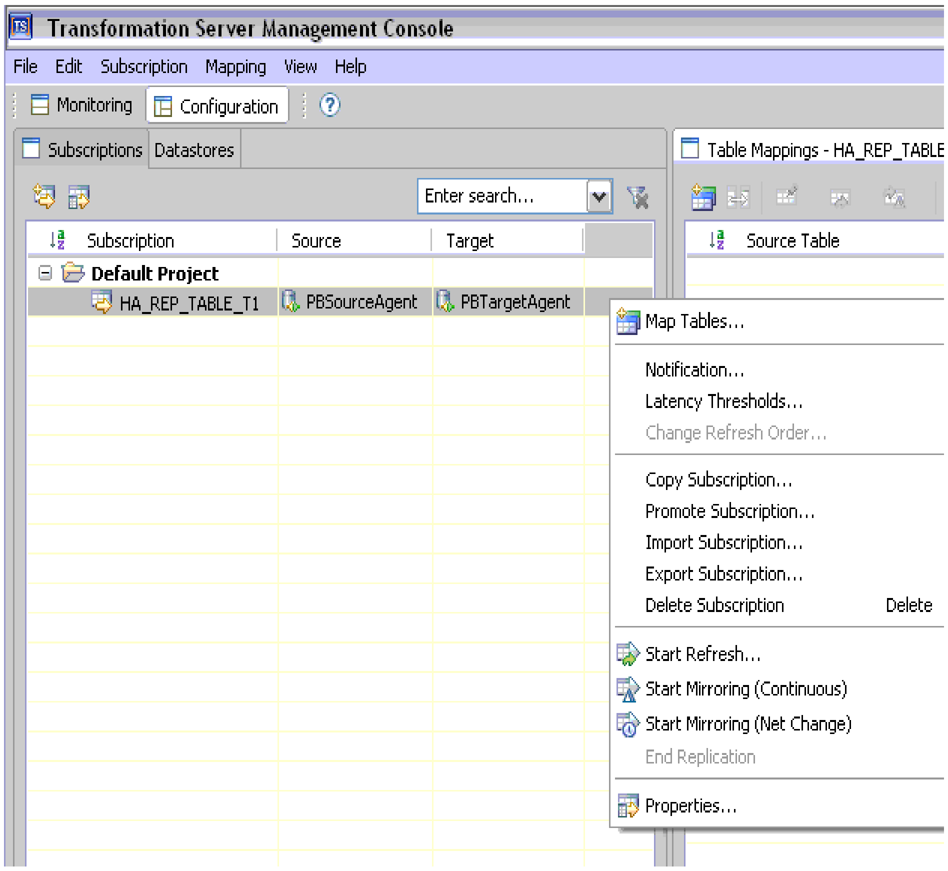
   c. Manage the display to equal what is displayed in Figure 10-14.

      You need to enter a name for this Subscription; this name is displayed as the identifier in diagnostic screens and reports for what we are about to further define.

      You need also to set the source and target Datastores.
   d. Click OK when you are done.
9. Mapping tables to a Subscription.

A right-click in the Subscriptions view of our newly created Subscription (HA_REP_TABLE_T1) produces the context sensitive menu displayed in Figure 10-15.

First we will use the Map Tables wizard to specify the relationship between our source and target tables. Then we will call to Start Mirroring (Continuous), to put the data replication process in effect.



Subscription	Source	Target
Default Project		
HA_REP_TABLE_T1	PBSourceAgent	PBTargetAgent

Figure 10-15 Mapping tables to a Subscription.

a. Right-click the entry entitled, HA_REP_TABLE_T1, in the Subscriptions view, and select, Map Tables.

   This action produces the Map Tables wizard.

b. Press the radio button for a One to One Mapping, and click Next.

Note: *The other choices here included the ability to create an audit trail mapping and a replication other than one source table to one target table.*

An audit trail mapping allows for the ability to include numerous additional columns of metadata in the replication; columns like the timestamp of the individual row activity, source (triggering) username, and so on.

After you complete the activity we are on presently, you may return and experiment with audit trail replication and the like. Later, if you create an audit trail replication, it may initially be easier to instruct DataMirror to create the target (destination) table. In that manner, you are not responsible for defining these new metadata columns.

c. Navigate to the T1 table in the PBPUBLIC schema, and click Next.

   You are ready to click Next when T1 is checked.

   (DataMirror knows that this table is in the source Datastore.)

d. Press the radio button entitled, Map to existing target tables, and click Next.

Note: *This is where you can branch, and have DataMirror create the target tables, or use your own. IF you use (create) your own tables, you can use whatever advanced DBA skills you have to optimize the table, its storage, and so on.*

e. Again, navigate to the T1 table, and click Next.

   T1 is under the PBPUBLIC schema.

   (DataMirror knows this copy of T1 is in the target Datastore.)

f. This screen offers a review of choices made thus far, click Next.

g. DataMirror can perform live or batch data replication. We wish to perform live (continuous) replication.

   Press the radio button entitled, Mirror (Change Data Capture), and click Next.

h. Click Finish.

10. Activate and test replication.

At this point we have performed every step necessary to establish high availability data replication between two tables on two different database servers. All that remains is to activate replication, and perform tests to verify the results of our work.

a. Right-click the entry entitled, HA_REP_TABLE_T1, in the Subscriptions view, and select, Start Mirroring (Continuous).

   Complete any dialog boxes (Are you sure?, etcetera.), that you are presented with.

b. Return to the two PointBase graphical administration clients and verify your work within DataMirror (ensure that data replication is actually happening as you would expect).

   The following example SQL DML statements may be of use;

   - SELECT * FROM t1.
   - DELETE FROM t1 WHERE col1 = 4;
   - UPDATE t1 SET col1 = 8 WHERE col1 = 2;
   - INSERT INTO t1 VALUES (10,10);
   - And so on.

10.3 Summation

In this document, we reviewed or created:

All of the activity performed in this document was done using DataMirror Transformation Server (DM/TS), a new component to IBM Information Server.

We used the DM/TS Access Control server console, the DM/TS Management console, and the graphical administration client to the PointBase database server. While there are numerous additional activities we can perform with these tools, the example we performed in this document was to create a high availability data replication between two tables. After you get the core concepts, the rest is easy.

Additional resources:

None.

Thank you to the persons who helped this month:

James Spyker, Glenn Steffler, Chris Tulk, and Dave Randall.

Legal statements:

Special attributions: