Distributed Database Guide

Version: 00.01

Document No: 42/DBM42-T04192006-01-DDB
Author: DBMaker Support Team Apr 19, 2006
LingAn Computer Engineering CO.

Print Date: Apr 29, 2006

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction 1-1
1.1 Additional Resources 1-1
1.2 Document Conventions 1-2

Part I Distributed Database System 1-1

2. Understanding Distributed Database 2-1
2.1 Introduction to Distributed Databases 2-1
2.2 DBMaker Distributed Database Structure 2-2

3. Remote Database Objects 3-1
3.1 Distributed Database Environment 3-1
3.2 Remote Database Connections - By Database Names 3-3
3.3 Remote Database Connections – By Database Links 3-4
3.3.1 CREATE DATABASE LINKS 3-4
3.3.2 USE DATABASE LINKS 3-4
3.3.3 CLOSE DATABASE LINKS 3-6
3.3.4 DROP DATABASE LINKS 3-7
3.4 Remote Object Mapping 3-7
3.4.1 USE DATABASE NAME OR LINK NAME 3-7
3.4.2 SYNONYMS 3-8
3.4.3 VIEWS 3-8

4. Distributed Query 4-1
4.1 Insert, Delete, Update 4-1
4.2 Select 4-1
4.2.1 SIMPLE OBJECT 4-1
4.2.2 JOIN 4-2

4.2.3 SUBQUERY 4-2
4.2.4 STORE COMMAND 4-2
4.2.5 TRIGGER 4-2
4.3 Limitations 4-2

5. Distributed Transaction Control 5-1

5.1 Two--Phase Commit 5-1
5.2 Distributed Transaction Recovery 5-1
5.3 Heuristic End Global Transaction 5-2

6. DDB Error Handling 6-1

Part II Replication 6-1

7. Replication Concept 7-1

7.1 Database Replication Concept 7-1
7.2 Table Replication Concept 7-1
7.3 Database Replication v.s. Table Replication 7-2

8. Database Replication 8-1

8.1 WHAT IS DATABASE REPLICATION 8-1
8.2 DATABASE REPLICATION LIMITATIONS 8-1
8.3 DATABASE CONFIGURATION FILE 8-2
8.4 DATABASE REPLICATION SETUP 8-4
8.5 USING JSERVER MANAGER SETUP ENVIRONMENT 8-5
8.6 EXAMPLES 8-6

9. Table replication 9-1

9.1 What is Table Replication 9-1
9.2 Table Replication Rules 9-1
9.3 Table Replication Type 9-2
9.4 Synchronous Table Replication 9-2
9.4.1 WHAT IS SYNCHRONOUS TABLE REPLICATION 9-2

9.4.2 SYNCHRONOUS TABLE REPLICATION SETUP 9-2
9.4.3 SYNCHRONOUS TABLE REPLICATION SYNTAX 9-2
9.4.3.1 CREATING TABLE REPLICATION 9-2
9.4.3.2 ALTER REPLICATION 9-4
9.4.3.3 DROP REPLICATION 9-4
9.4.4 EXAMPLES 9-5
9.5 Asynchronous Table Replication 9-6
9.5.1 WHAT IS ASYNCHRONOUS TABLE REPLICATION 9-6
9.5.2 ASYNCHRONOUS TABLE REPLICATION SETUP 9-6
9.5.3 SCHEDULE 9-7
9.5.3.1 CREATE SCHEDULE 9-8
9.5.3.2 DROP SCHEDULE 9-9
9.5.3.3 ALTER SCHEDULE 9-9
9.5.3.4 SUSPENDING AND RESUMING SCHEDULE 9-10
9.5.4 ASYNCHRONOUS TABLE REPLICATION SYNTAX 9-10
9.5.5 SYNCHRONIZING A REPLICATION 9-11
9.5.6 EXAMPLES 9-12
9.6 Heterogeneous Asynchronous Table Replication 9-14
9.6.1 WHAT IS HETEROGENEOUS ASYNCHRONOUS TABLE REPLICATION 9-14
9.6.2 HETEROGENEOUS ASYNCHRONOUS TABLE REPLICATION SETUP 9-14
9.6.3 HETEROGENEOUS ASYNCHRONOUS TABLE REPLICATION SYNTAX 9-14
9.6.4 EXAMPLES 9-15
9.7 Express Asynchronous Table Replication 9-16
9.7.1 WHAT IS EXPRESS ASYNCHRONOUS TABLE REPLICATION 9-16
9.7.2 EXPRESS ASYNCHRONOUS TABLE REPLICATION SETUP ... 9-16
9.7.3 EXAMPLES 9-17
9.8 Table Replication System Catalog 9-18
SYSDBLINK 9-18
SYSOPENLINK 9-18
SYSPUBLISH 9-19
SYSSUBSCRIBE 9-19
SYSTRPDEST 9-19
SYSTRPJOB 9-20
SYSTRPPOS 9-20
SYSUSER 9-20
9.9 Table Replication Error Handling 9-21

Appendix Tables used in the Examples i

1. Introduction

Welcome to DBMaker Distributed Database Guide. This Guide will provide some instruction and samples for users to use the DBMaker Distributed Database, you should be cognizant of the simplicity and power of the DBMaker distributed architecture. In addition, this Guide also introduces the applications with data replication. Because there are close relations between distributed database and replication among the database objects.

This Guide divides into two parts. Part I introduces the distributed database management function, including distributed databases, the distributed architecture, distributed data access, distributed database object management, and distributed transaction management. Part II describes the concept and particular applications with data replication. Of course, we will provide lots of examples on these two parts, so that help users with understanding and using DBMaker Distributed Database.

1.1 Additional Resources

DBMaker provides a complete set of DBMS manuals in addition to this one. For more detailed information on a particular subject, consult one of the books listed below:

- For an introduction to DBMaker's capabilities and functions, refer to the *DBMaker Tutorial*.
- For more information on designing, administering, and maintaining a DBMaker database, refers to the *Database Administrator's Guide*.
- For more information on database management, refer to the *JDBA Tool User's Guide*.
- For more information on database server management, refer to the *JServer Manager User's Guide*.
- For more information on configuring DBMaker, refer to the *JConfiguration Tool Reference*.
- For more information on the native ODBC API, refer to the *ODBC Programmer's Guide*.
- For more information on the dmSQL interface tool, refer to the *dmSQL User's Guide*.
- For more information on the SQL language used in dmSQL, refer to the *SQL Command and Function Reference*.
- For more information on error and warning messages, refer to the *Error and Message Reference*.

- For more information on the DBMaker COBOL Interface, refer to the *DCI User's Guide*.

1.2 Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and Command Line conventions also have a second setting used with indentation.

CONVENTION	DESCRIPTION
Italics	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but is replaced by the actual name. Italics also introduce new words, and are occasionally used for emphasis in text.
Boldface	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
SMALL CAPS	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.
Procedure	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow
Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
`CommandLine`	Indicates text, as it should appear on a text-delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file

Table 1-1Document Conventions

Part I

DISTRIBUTED DATABASE SYSTEM

2. Understanding Distributed Database

A distributed database is a database in which portions of the database are stored on multiple computers within a network. Users have access to the portion of the database at their location so that they can access the data relevant to their tasks without interfering with the work of others. A centralized distributed database management system (DDBMS) manages the database as if it were all stored on the same computer. The DDBMS synchronizes all the data periodically and, in cases where multiple users must access the same data, ensures that updates and deletes performed on the data at one location will be automatically reflected in the data stored else where.

2.1 Introduction to Distributed Databases

Traditional client-server DBMS, as shown in Figure 2-1, locate the database on a specific network computer, and the computer is responsible for handling all client requests.

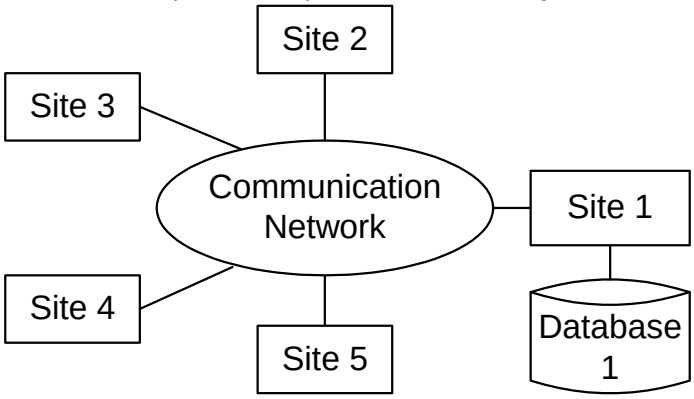

Figure 2-1 Traditional client/server database management system

Distributed databases, as shown in Figure 2-2, locate a copy of the database on several network computers, and each can independently support clients. The distributed database management system manages the databases on these computers, so users can access the data transparently.

DBMaker supports a true distributed architecture to provide a complete and robust distributed database management system (Distributed DBMS). It provides remote database connections, distributed queries, and distributed transaction management. DBMaker also provides table and database replication to keep data automatically up-to-date.

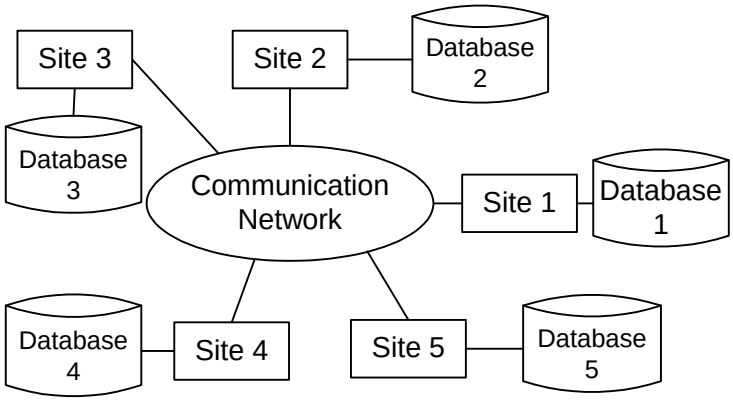

Figure 2-2 Distributed database in client-server

In the DBMaker distributed database environment, you can write application programs using the DBMaker ODBC 3.0 compatible API or perform ad-hoc SQL queries that access data from different parts of the distributed database. DBMaker will transparently integrate the data and return the results, just as if they all came from a local database.

2.2 DBMaker Distributed Database Structure

The DBMaker distributed database environment builds on the traditional client/server architecture, effectively linking multiple client applications and multiple database servers. Client applications process user requests and display the results, and the database servers handle data management. Each client has a direct connection to a single database server, which is known as the Coordinator Database to that client. Through the Coordinator Database, the client can connect to other remote databases, which are known as Participant Databases.

DBMaker uses a hierarchical distribution structure to connect to remote databases. This allows DBMaker to access data from a remote database with no direct connection to the Coordinator Database by routing through one of the Participant Databases. When this happens, the Participant Database becomes a Local Coordinator Database, and acts as coordinator for any child databases accessed through it.

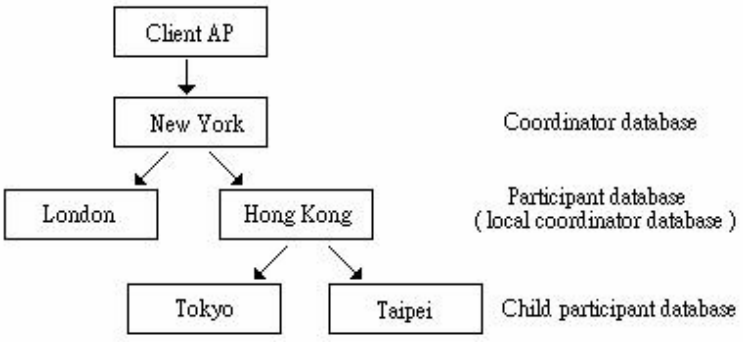

Figure 2-3 DBMaker distribution structure

In Figure 2-3, the client application program connects to the database server in New York, which makes the database in New York the Coordinator Database. If you use the database in New York

to access data from London and Hong Kong, then both the London and Hong Kong databases are Participant Databases.

Some of the data you are looking for in Hong Kong might actually be in the databases in Tokyo or Taipei, so the databases in Tokyo and Taipei are Child Participant Databases. This makes the database in Hong Kong a Coordinator Database for the databases in Tokyo and Taipei, so the database in Hong Kong is not only a Participant Database, but also acts as a Local Coordinator Database.

3. Remote Database Objects

DBMaker provides several different methods to access a Participant Database:

- Specify the Participant Database name directly.
- Using database links defined in the Coordinator Database.
- Through remote object mapping such as views or synonyms.

The difference between the first two approaches is that database links contain security information in addition to the remote database name. This allows you to specify the user name and password that you want to use in the database link when you access the remote database.

For more information about methods to access a Remote Database Objects and its example, please refer to the in later chapters.

3.1 Distributed Database Environment

Setting up a distributed database environment using DBMaker is very simple. All you need to do is add some keywords to the **dmconfig.ini** file to set the distributed database configuration options. Optionally, these parameters may be set using the JConfiguration Tool. For more information, refer to the *JConfiguration Tool User's Guide*.

To better understand how DBMaker manages distributed databases, refer to the following example.

Example 1

The local setting of dmconfig.ini file:

```
[test1]
DB_DBDIR = c:\database
DB_SVADR = 192.168.0.122
DB_PTNUM = 2100
DD_DDBMD = 1

[test2]
DB_SVADR = 192.168.0.121
DB_PTNUM = 2100
```

The remote setting of dmconfig.ini file:

```
[test2]
DB_DBDIR = c:\database
DB_SVADR = 192.168.0.121
```

```
DB_PTNUM = 2100
DD_DDBMD = 1

[test1]
DB_SVADR = 192.168.0.122
DB_PTNUM = 2100
```

In the files shown above, set **DD_DDBMd** =1 in the configuration section for the local database and remote database. You need to set DDB parameter on sever side, and the keywords setting of [remote_db_name], DB_USRID, DB_PASWD, DB_SVADR, DB_PTNUM must consistent with its setting on client side. Then you can use distributed database after start database again.

These keywords (include the local and the remote side) of the above example are the minimum settings for the distributed database environment

Example 2

The local computer (IP:192.168.0.122)is Linux OS. The remote computer(IP:192.168.0.121) is Win2000 OS.

The local setting(IP:192.168.0.122, host name: A) of dmconfig.ini file:

```
[testddb]
DB_DBDIR = /home/dbmaker/4.1/testDDB
DB_SVADR = 192.168.0.122
DB_PTNUM = 1111
DB_USRID = SYSADM
DD_DDBMD = 1
[testddb@121]
DB_SVADR = 192.168.0.121
DB_PTNUM = 2222
DB_USRID = SYSADM
DD_CTIMO = 20
DD_LTIMO = 10
```

The remote setting(IP:192.168.0.121, host name: B) of dmconfig.ini file:

```
[testddb]
DB_DBDIR = c:\testddb
DB_SVADR = 192.168.0.121
DB_PTNUM = 2222
DB_USRID = SYSADM
DD_DDBMD = 1
[testddb@122]
DB_SVADR = 192.168.0.122
DB_PTNUM = 1111
DB_USRID = SYSADM
DD_CTIMO = 20
DD_LTIMO = 10
```

In the files shown above, set **DD_DDBMd** =1 in the configuration section for the local database to enable distributed database support.

In addition, include a database configuration section for the Participant Database in the Coordinator Database configuration file, and for the Coordinator Database in the Participant Database configuration file. In this case, both the local database and the remote database use the same database name. If you use the remote database name for the name of the database configuration section, it will cause a conflict with the local database name in the **dmconfig.ini** file.

To avoid this type of problem when using distributed databases, DBMaker can distinguish the remote database name from the local database name by appending a server host description to the remote database name in the local **dmconfig.ini** file.

The remote database name would look like:

```
database_name@server_host_description
```

The server host description can be any identifying name, such as the IP address or host name of the database server, the domain name, or almost any other descriptive text. In this example, the local side client application would use **testddb@121** when it wants to access data in the remote database whose computer IP is 192.168.1.121, and the remote side client application would use **testddb@122** when it wants to access data in the machine IP is 192.168.1.122.

Also, set up the server address and port name for both the local database and the remote database in their respective configuration sections in the configuration files at both the local and remote computer.

In this example, the local database configuration file would contain the local server address in the **testddb** configuration section, and would contain the remote database server address in the **testddb@121** configuration section. Similarly, the remote database configuration file would contain itself server address in the **testddb** configuration section, and would contain the machine of local server address in the **testddb@122** configuration section.

You should also set the **DD_CTimO** and **DD_LTimO** remote connection parameters. These parameters go in the configuration section for the Participant Database in the Coordinator Database configuration file, and for the Coordinator Database in the Participant Database configuration file. Please reference the *Database Administrator's Guide* to get detailed.

3.2 Remote Database Connections - By Database Names

Users can connect to remote databases with the database name of the Coordinator Database Server. Users must know the remote database name, which is defined in the **dmconfig.ini** file in the Coordinator Database Server.

Example 1

A client application in the host A (IP:192.168.0.122) accesses the database testddb located in the host B(192.168.0.121) .It appears that the user is connecting to the host B with the user name SYSADM and the password aa. In reality, the user is connecting to the Coordinator Database, which is the host database. The Coordinator Database then connects to the remote database with the account and password used.

```
dmSQL> CONNECT TO testddb SYSADM;       //connect to the local database which is in
                                          the host A
dmSQL> SELECT * FROM testddb@121:tuser;   //access remote database which is in the host
B
```

Note: This example distinguish the remote database name from the local database name by using database section name(testddb@121)

3.3 Remote Database Connections – By Database Links

The distributed databases need to have same users and passwords on both server and client side. It is not very convenient to manage global databases. We can solve it with the method of DB- Link.

A database link creates a connection to a remote database, and contains the login information and password necessary for connecting. The link permits users to connect to a remote database with a different user name than in the Coordinator Database, or to connect to a remote database with no account. It also makes data in a distributed database environment location transparent. The link definition, which also contains the login information and password, is stored in the Coordinator Database.

3.3.1 CREATE DATABASE LINKS

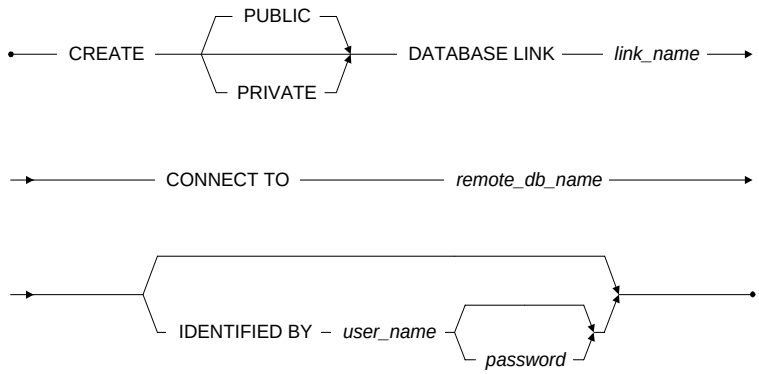

Figure 3-1 Syntax for creating a database link

Only database administrators can create public links to be used by all users in a database. Any user can create private links for themselves. Users may create private links using the same name. A private link will override a public link with an identical name.

DBMaker will create a private link by default if the user does not specify the type of link desired. If the user does not specify the login account and password in the IDENTIFY BY clause, the user's current login name and password will be used by default.

3.3.2 USE DATABASE LINKS

The following shows how to access remote database objects using database links.

Example 1

In this example, the SYSADM connects to the database and creates a public link named **lkip_121** that connects to the host B database using the SYSADM account. The SYSADM updates some values and disconnects. Then **user1** connects and performs a query on the **tuser** table.

```
dmSQL> CONNECT TO TESTDDB SYSADM;
dmSQL> CREATE PUBLIC DATABASE LINK lkip_121 CONNECT TO testddb@121
    2> IDENTIFIED BY SYSADM;
dmSQL> UPDATE lkip_121:tuser SET name = 'dbmaker'
    2> WHERE id = 1;
dmSQL> DISCONNECT;
dmSQL> CONNECT TO testddb user1 pwd1;
dmSQL> SELECT * FROM lkip_121:tuser;
dmSQL> SELECT * FROM testddb@121:tuser;
ERROR(6803):[DBMaker] invalid authorization specification (invalid username when
connecting)
```

Note: When **user1** accesses host B database testddb by using database name, an error 6803 occurred showed as above. Because of accessing remote database by using database name, the local side and the remote side need have same user name and password. But if you using DB-Link, you don't need to care about the existence of DDB. It seemed as if you are using a specific account connect to the remote database That is to say, if the user A connect to the database test1 by account A, you can using DB-Link method connect to the database test2 by a specific account, for example the user B account.

Example 2

The following examples show different ways to access a remote database, one through a database link and the other by specifying the remote database name in the form of dbname@"".

```
dmSQL> CONNECT TO BankTranx SYSADM;
dmSQL> CREATE PUBLIC DATABASE LINK BankMIS CONNECT TO BankMIS
    2> IDENTIFIED BY SYSADM;
dmSQL> DISCONNECT;
dmSQL> CONNECT TO BankTranx user1 pwd1;
dmSQL> SELECT * FROM BankMIS:employee;              //using database link
dmSQL> SELECT * FROM BankMIS@"":SYSADM.employee;    //using remote db name
```

Note: If a database link name is the same as the remote database name, DBMaker will use the database link name in preference to the remote database name. If you want to access the remote database directly, you must specify the remote database name in the form of dbname@ to force DBMaker to access the remote database directly instead of through the database link.

Example 3

In this example, the user name of connecting to the remote database is user1, and user1 only have resource authority. Therefore, user1 can not insert data to other user's table if user1 without specific authority.

```
dmSQL> CONNECT TO testddb sysadm;
dmSQL> CREATE DATABASE LINK testlk CONNECT TO testddb@121 IDENTIFIED BY user1;
//the user1 account only has resource authority
```

```
dmSQL> INSERT INTO testlk:sysadm.tuser values(1,'dbmaker');
ERROR(6830):[DBMaker] access violation: no INSERT privilege
```

Note: The IDENTIFIED BY keywords specify the user name and password to use when connecting to the remote database. The user name provided must be an existing user in the remote database with the CONNECT or higher security privileges. When the link is used to connect to the remote database, the operations a user can perform depend on the security and object privileges granted to. Therefore, the user1 account must have the specific authority to ensure that the user1 can access database objects. Otherwise, an error will occur.

3.3.3 CLOSE DATABASE LINKS

Once a user accesses a remote database with an SQL command, the Coordinator Database will build a remote connection to the Participant Database. The remote connection will remain open until all users disconnect from the Coordinate Database or until the link is closed with the CLOSE DATABASE LINK command. DBMaker provides up to 256 remote connections for each database; it is a good idea to close remote connections that are no longer in use to free the connections for other users.

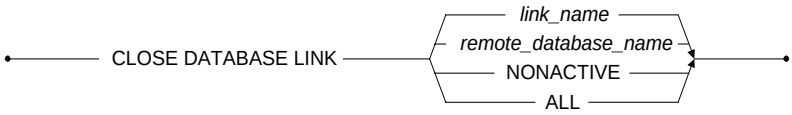

Figure 3-2 Syntax for closing a database link

Example 1

To close a database link using the remote database name **testddb**:

```
dmSQL> close database link testddb;
```

Example 2

To close a database link using the remote database link **lkip_121**:

```
dmSQL> close database link lkip_121;
```

When a user issues a CLOSE DATABASE LINK command, DBMaker will decrease the remote connection counter by one. When the counter reaches zero, the connection is fully closed and the occupied resources freed. Otherwise, the connection remains open.

Example 3

To close all database links and free the connections and resources:

```
CLOSE DATABASE LINK ALL
```

Example 4

To close all NONACTIVE remote connections no longer being used:

```
CLOSE DATABASE LINK NONACTIVE
```

3.3.4 DROP DATABASE LINKS

Figure 3-3 Syntax for deleting a database link

Only database administrators can delete public links, and only the owner of a private link can delete it. Ensure to specify the public link to be deleted when it has the same name as a private link or DBMaker will delete the private link by default.

Example

To delete a public database link named **lkip_121**:

```
dmSQL> DROP PUBLIC DATABASE LINK lkip_121;
```

3.4 Remote Object Mapping

Database Object Mapping provides better location transparency in a distributed database environment. There is no difference between the way a user accesses remote database objects with Database Object Mapping and local database objects.

3.4.1 USE DATABASE NAME OR LINK NAME

Previous section has details on the usage of this method; this section will offer some examples for reference.

Example 1

The following example accesses the remote database testddb located in the host B database testddb by using database name. It appears that the user is connecting to the BeiJing branch with the user name SYSADM and the password aa. In reality, the user is connecting to the Coordinator Database.

```
dmSQL> CONNECT TO testddb SYSADM aa;
dmSQL> SELECT * FROM testddb@121:SYSADM.employee;
```

Note: This example distinguish the remote database name from the local database name by using database section name(testddb@121)

Example 2

The following example creates a public database link named **lkip_121** to the remote **testddb** database. The user creating the link must have DBA or SYSADM security privileges in the local database. Using this link automatically connects the user to the remote database with the user name **LinkUser** and password **dil3ryx9**. It provides the security and object privileges granted to this user.

```
dmSQL> CREATE PUBLIC Database LINK lkip_121 CONNECT TO testddb@121
       IDENTIFIED BY LinkUser dil3ryx9;
```

Example 3

The following creates a private database link named **lkip_123** to the remote **testddb** database. Using this link automatically connects a user to the remote database with the user name **Vivian** and password **a23456**. It provides the security and object privileges granted to this user. This is useful if you have a different user name in the local and remote databases. If there is a public link with the same name, the private link is used instead.

```
dmSQL> CREATE PRIVATE Database LINK lkip_123 CONNECT TO testddb@123
       IDENTIFIED BY Vivian a23456
```

3.4.2 SYNONYMS

Using a synonym to define a remote database object is done by assigning the remote database object an alias name. The privileges you have in the remote database when using a synonym are the same as in a local database.

Example

To access a remote database object using a synonym:

```
dmSQL> CONNECT TO BankTranx user1;
dmSQL> CREATE DATABASE LINK MISLK CONNECT TO BankMIS IDENTIFIED BY user2;
dmSQL> CREATE SYNONYM s1 FOR BankTranx:customer;
dmSQL> CREATE SYNONYM s2 FOR MISLK:user2.employee;
dmSQL> SELECT * FROM s1;
       // SELECT * FROM BankTranx:user1.customer; (BankTranx, user1)
dmSQL> SELECT * FROM s2;
       // SELECT * FROM MISLK:user2.employee; (BankMIS, user2)
dmSQL> DISCONNECT;
dmSQL> CONNECT TO BankTranx user3;
dmSQL> CREATE DATABASE LINK MISLK CONNECT TO BankMIS IDENTIFIED BY user4;
dmSQL> SELECT * FROM s1;
       // SELECT * FROM BankTranx:user1.customer; (BankTranx, user3)
dmSQL> SELECT * FROM s2;
       // SELECT * FROM MISLK:user2.employee; (BankMIS, user4)
```

The comments indicate the equivalent SQL expression, the database being connected to, and the account used to connect.

3.4.3 VIEWS

Using a view to define a remote database object is a bit different than using a synonym. The view is not just an alias, but includes the database name, user account, password, object owner, and object name as part of the definition. The privileges a user has in the remote database depend on the privileges of the user that created it.

Example

To access a remote database object using a view:

```
dmSQL> CONNECT TO BankTranx user1;
dmSQL> CREATE DATABASE LINK MISLK CONNECT TO BankMIS IDENTIFIED BY user2;
```

```
dmSQL> CREATE VIEW v1 AS SELECT * FROM BankTranx:customer;
dmSQL> CREATE VIEW v2 AS SELECT * FROM MISLK:user3.employee;
dmSQL> SELECT * FROM v1;
       // SELECT * FROM BankTranx:user1.customer; (BankTranx, user1)
dmSQL> SELECT * FROM v2;
       // SELECT * FROM BankMIS:user3.employee; (BankMIS, user2)
dmSQL> DISCONNECT;
dmSQL> CONNECT TO BankTranx user3;
dmSQL> CREATE DATABASE LINK MISLK CONNECT TO BankMIS IDENTIFIED BY user4;
dmSQL> SELECT * FROM v1;
       // SELECT * FROM BankTranx:user1.customer; (BankTranx, user3)
dmSQL> SELECT * FROM v2;
       // SELECT * FROM MISLK:user3.employee; (BankMIS, user4)
```

The comments indicate the equivalent SQL expression, the database being connected to, and the account used to connect.

4. Distributed Query

There is no obvious difference between the statements of a distributed query and a normal query, except in the way database objects are specified. However, when using a remote database, the only remote database objects that can be accessed are tables, views, or synonyms. To access a remote database object, provide the remote database name or database link when specifying the name of the database object. This provides two ways to identify a remote database object:

- remote_database_name:object_owner.object_name
- database_link:object_owner.object_name

4.1 Insert, Delete, Update

To specify a remote database object in a query:

```
dmSQL> CONNECT TO TESTDDB SYSADM;
dmSQL> CREATE PUBLIC DATABASE LINK lkip_121 CONNECT TO testddb@121
    2> IDENTIFIED BY SYSADM;
dmSQL> INSERT INTO lkip_121:tuser values(1,'dbmaker');              //by database link
dmSQL> INSERT INTO testddb@121:tuser values(2,'casemaker');         //by database name
dmSQL> UPDATE lkip_121:tuser SET name = 'dbmaser' WHERE id = 1;
dmSQL> DELETE FROM lkip_121:tuser WHERE id = 1;
```

select into between different database

```
dmSQL> select * from lkip_121:tuser into tuser;
       // select remote database data into local database
dmSQL> SELECT * from tuser into lkip_121:tuser;
       // select local database data into remote database
dmSQL> select * from lkip_121:tuser into lkip_123:tuser;
       // select remote db lkip_121 into lkip_123
```

4.2 Select

4.2.1 SIMPLE OBJECT

The following examples only using simple object to access remote database objects:

```
dmSQL> CONNECT TO testddb SYSADM aa;
dmSQL> SELECT * FROM testddb@121:SYSADM.tuser ORDER BY id;
dmSQL> SELECT * FROM testddb@121:SYSADM.tuser where usetime = CURRENT_TIME;
dmSQL> SELECT dept_ID as ID1, AVG(salary) FROM testddb@121:employee group by ID1;
```

4.2.2 JOIN

Using joins to access separate remote database objects:

```
dmSQL> CONNECT TO testddb SYSADM
dmSQL> CREATE DATABASE LINK LKIP_121 CONNECT TO testddb@121 IDENTIFIED BY SYSADM;
dmSQL> CREATE DATABASE LINK LKIP_123 CONNECT TO testddb@123 IDENTIFIED BY SYSADM;
dmSQL> SELECT * FROM lkip_121:tuser cross join linker;
       // cross join remote db lkip_121 and local database
dmSQL> SELECT * FROM lkip_121:tuser inner join lkip_123:tuser on
       lkip_121:tuser.id=lkip_123:user.id;
      // inner join between remote db lkip_121 and remote db lkip_123
dmSQL> SELECT * from tuser a,lkip_121:tuser b, lkip_123:tuser c where a.id=b.id and
       a.id=c.id;      // join between local db, remote db lkip_121 and remote db
       lkip_123
```

4.2.3 SUBQUERY

Using subquery to access remote database objects:

```
dmSQL> SELECT * FROM tuser where tuser.id in (SELECT id FROM testddb@121:linker)
dmSQL> SELECT * FROM testddb@121:tuser where id > (SELECT AVG(id) FROM
       testddb@123:linker);
```

4.2.4 STORE COMMAND

Using store command to access remote database objects:

```
dmSQL> create command sc1 as insert into testddb@121:tuser values(1,?,?);
dmSQL> execute command sc1;
```

4.2.5 TRIGGER

Using trigger to access remote database objects(create trigger on remote database):

```
dmSQL> create table tuser(id int not null, name char(10), usetime time);
dmSQL> create table linker(id int not null, sex char(4), name char(10));
dmSQL> insert into tuser values(1,'dbmaker');
dmSQL> create trigger trg1 before update on tuser for each row (insert into linker
       values(old.id,old.name));
//these sql commands above were created on the client side of the remote database
dmSQL> update testddb@121:tuser set name='dbmaker' where id =1;     //in local
database
dmSQL> select * from testddb@121:tuser;
dmSQL> select * from testddb@121:tuser;
```

4.3 Limitations

There have some limitations when you perform a distributed query. If you start the database in DDB mode, update and delete can not support subquery, but a sub-query can appear in the insert statement.

For example:

```
dmSQL> insert into testddb@123:tuser (select id,name from testddb@123:linker);
5 rows insert
dmSQL> update testddb@123:linker set sex='f' where id in (select id from
       testddb@123:tuser);
ERROR (6105): [DBMaker] limitation or not supported yet
dmSQL> delete from testddb@123:tuser where id in (select id from testddb@123:linker);
ERROR (6105): [DBMaker] limitation or not supported yet
```

5. Distributed Transaction Control

DBMaker supports a distributed transaction mechanism that is transparent to users. Participant Databases do not commit only a part of a distributed transaction; DBMaker handles it.

Example

The following shows how distributed transaction control works.

```
dmSQL> CONNECT TO BankTranx user1;                    // ABC Bank in Taipei
dmSQL> SET AUTOCOMMIT OFF;
dmSQL> UPDATE BankTranx:Customer SET money=money-1000 where id=123;
dmSQL> UPDATE BankTranx@"Bank_in_Seattle":Customer SET money=money+1000
    2> WHERE id=123;
dmSQL> COMMIT;
```

Since the Coordinator Database handles all database operations from the client application, the Coordinator Database knows the scope of the instructions via the Distributed System Catalog Manager. The Coordinator Database will handle transactions belonging to the local site in the same manner as a regular client/server transaction. Transactions belonging to remote sites need to reference the appropriate remote database. The Coordinator Database will exchange information with every Participant Database and coordinate the whole transaction until it is either rolled back or committed.

5.1 Two--Phase Commit

Database management systems need to maintain data integrity, and this requires that all transactions be *atomic*. All operations in the transaction must commit or roll back together. In the traditional client/server architecture, a journal is used to make sure that the changes are either rolled back or committed.

In the distributed database architecture, a two-phase commit protocol with presumed abort is used as the mechanism for controlling distributed transactions that span multiple database servers. A transaction that modifies data on two or more databases must complete the two-phase commit protocol before it is committed. The two-phase commit mechanism guarantees that all sites commit or roll back globally. It also protects data manipulation operations performed by remote synonyms, integrity constraints, and triggers. To commit a transaction, a user has to ensure every sub-transaction has finished; otherwise, the transaction will be aborted. For the same reason, if any sub-transaction cannot commit, the other sub-transactions must be aborted as well.

5.2 Distributed Transaction Recovery

DBMaker uses the two-phase commit protocol to inform all Participant Databases to commit a global transaction. Before entering the commit phase, the Coordinator Database will check the status of the Participant Databases to ensure there are no server or network problems. If the Coordinator Database finds a problem with any of the Participant Databases, it informs the other

Participant Databases to roll back their part of the transaction, and returns an error indicating the global transaction has failed. If the two-phase commit protocol has finished the preparation phase, but there is a server or network problem with a Participant Database, the global transaction is regarded as a success. DBMaker records which Participant Database server cannot commit its part of the transaction in the SYSGLBTRANX system catalog table, and records which database contains a pending transaction in the SYSPENDTRANX system catalog table for the crashed database.

DBMaker also provides an automatic recovery mechanism to handle network or site failure during the execution of a distributed transaction. In the Coordinator Database, you start the global transaction recovery daemon (GTRECO). This daemon scans the SYSGLBTRANX system catalog table and periodically recovers any pending global transactions. Then it tries to connect to the crashed Participant Database and informs it to commit or roll back its part of the global transaction.

5.3 Heuristic End Global Transaction

After a network or site failure occurs during the two-phase commit, pending transactions continue to hold some resources such as locks or journal blocks. The pending transaction will occupy these resources until the problem is solved by the global recovery daemon (GTRECO). If the network or site failure cannot be solved immediately, then the held resources may block some of the users in the Participant site. To solve this problem, DBMaker supports heuristic end global transaction. A heuristic end transaction is an independent action taken by the database administrator to force a pending transaction in a Participant Database to commit or roll back. The database administrator can use JDBATool to solve this problem. Refer to the *JDBA Tool User's Guide* for more information.

➲ **To resolve a pending transaction manually, perform the following:**

In the Participant Database, browse the SYSPENDTRANX table to find out whether there are transactions that have been pending for a long time.

1. In the Coordinator Database, determine the commit status of the pending transaction from SYSGLBTRANX. Also determine the commit status of the pending transaction from the Participant Database. For example, if there are two pending transactions "DB_1-3376aafd" and "DB_2-3376aafd:DB_3-3376ab0f#1", the administrator of the coordinator database should ask the administrator of DB_1 to determine the status of "DB_1-3376aafd" and ask the administrator of DB_3 to determine the status of "DB_2-3376aafd:DB_3-3376ab0f#1".
2. In the Coordinator Database, when the administrator receives the transaction status query he can examine SYSGLBTRANX to determine the transaction status. If the STATE is 2 (COMMIT) or 3 (PENDCOM), reply "commit". If the STATE is 4 (PENDABO), reply 'abort'. But if the STATE is 1 (PREPARE), this transaction branch is pending in this site too, and ask the administrator of the parent site to determine its status.
3. In the Participant Database, the administrator uses JServer Manager to perform a heuristic commit or abort the pending transaction based on the reply.

If the administrator initiates a heuristic end transaction on a pending transaction that is different from action taken in the Coordinator Database, the distributed data will be inconsistent.

6. DDB Error Handling

For distributed database, the local side may face kinds of errors as follows:

- Warning

For example, if a CHAR(10)data type is insert to a CHAR(5) column type of remote table, there will be a data truncation warning. These kinds of errors will be ignored.

- Connection Errors

If the local server fails to connect to the remote database server, it will abort distributed transaction.

Example:

```
dmSQL> CONNECT TO testddb sysadm;
dmSQL> CREATE DATABASE LINK LKIP_121 CONNECT TO testddb@121 IDENTIFIED BY user2;
//there have no account user2 in the remote database
dmSQL> INSERT INTO lkip_121:user2.tuser values(1,'dbmaker');
ERROR(16209):[DBMaker] abort distributed transaction by remote connection error
```

In this example, connect to the remote database failed is due to there have not exist user2 account in the remote database. The user name provided must be an existing user in the remote database with the CONNECT or higher security privileges. When the link is used to connect to the remote database, the operations a user can perform depend on the security and object privileges granted to. If a user name is not specified when connecting to the remote database, DBMaker uses the current user name in the local database.

- Transaction Errors

DBMaker uses the two-phase commit protocol to inform all Participant Databases to commit a global transaction. For example, if the Coordinator Database finds a problem with any of the Participant Databases, it informs the other Participant Databases to roll back their part of the transaction, and returns an error indicating the global transaction has failed. DBMaker also provides an automatic recovery mechanism to handle network or site failure during the execution of a distributed transaction.

Please refer to the section Distributed Transaction Control (chapter 5) for more information about distributed transaction and how to handling.

Part II

REPLICATION

7. Replication Concept

Data Replication, in the broadest sense of the term, refers to the process of representing objects in more than one database.

Only a few years ago, corporate data resided in a central location. Remote departments accessed the information they needed by establishing direct connections to the central sites, or by requesting printed reports from central MIS. The connections however were expensive, unreliable, and limited in number, while the reports were inflexible and untimely.

Open systems brought inexpensive and powerful computing resources to all corners of an enterprise. The ability to share corporate information effectively using these new resources became an important competitive advantage for organizations. The question enterprises face today is not “Why distribute and share corporate data?” but rather “How can one distribute information effectively?” Replication is quickly becoming the architecture of choice for a majority of distributed corporate applications.

7.1 Database Replication Concept

Most enterprises or companies had to put all data in one file server or database in their headquarters, and all terminals needed to connect to the server directly. Under this architecture, the application system might work smoothly if all terminals are in the same building or area. However, the performance was much slower if the remote branch wanted to access the database, because of transmitting speed and network bandwidth.

To share data and increase access speed, DBMaker provides database replication. The database replication will replicate the primary database to the slave database during a set time frame. In other words, when there are changes happening to the primary database like newly added, modified, or deleted data, DBMaker will replicate the changed data into the slave database automatically. There are three advantages to using database replication. First, the performance of application systems will increase since it can access data from the local side directly. Second, the destination database will have the same data modification as the local one; thus, the goal of data sharing can be achieved efficiently. Third, if it fails to connect to the local database for an application, it still could connect to the remote database and continue to work. Although there are advantages to using data replication, more storage for data and more processes to handle the replication are required.

7.2 Table Replication Concept

A table replication creates a full or partial copy of a table in a remote location. This allows users in remote locations to work with a local copy of data. The local copy remains synchronized with the databases in other locations. This way each database can service data requests immediately and efficiently, without having to go to another machine over a slower network connection. This is not the same as backing up the database to a remote location, since the synchronization is done on a transaction-by-transaction basis by the DBMS itself, without any intervention from users.

7.3 Database Replication v.s. Table Replication

The major difference between database and table replication is that the replicated data object is different: one is the whole database; the other one is a table. Users can choose one of them depending on their demands. If you choose database replication, since the unit to replicate is the whole database, the target (or slave) database is read-only.

8. Database Replication

8.1 WHAT IS DATABASE REPLICATION

In this section, we will use *Figure 8-1* to explain the flow of database replication. Database replication will work with the journal backup server, if you are not familiar with database backup and restoration, please read Chapter 14, “Database Recovery, Backup, and Restoration”.

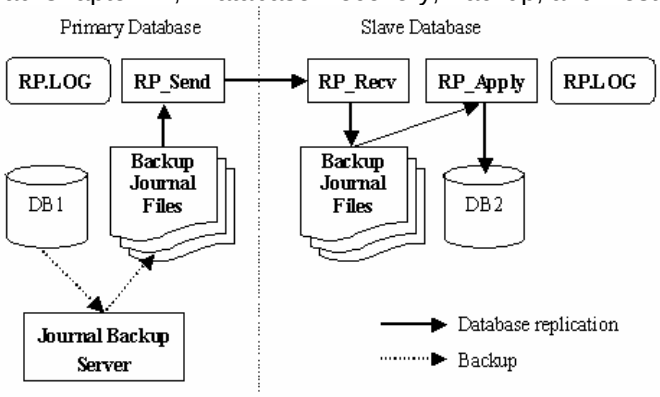

Figure 8-2: Flow of database replication

Database replication is accomplished with 4 servers: journal backup server, RP_Send server, RP_Recv server, and RP_Apply server. The journal backup server and the RP_Send server are started from the primary database. The RP_Recv server and the RP_Apply server are started from the slave database.

The initial step of database replication is to make the slave database the same as the primary. All subsequent changes, data insertion, deletion, creating schema, etc., have to be made in the primary database first. Then the journal backup server running on the primary database regularly writes changes to the backup journal.

The RP_Send server periodically sends the backup journal of the primary database to the slave-RP_Recv server. The RP_Recv gives notice to the RP_Apply to apply the received backup journal to the slave database.

The slave database is read-only for all users, only RP_Apply can update the slave database.

8.2 DATABASE REPLICATION LIMITATIONS

Summary of database replication limitations:

- Byte ordering must be the same on the primary and slave machines.
- Slave database is read-only.
- One primary database supports up to 256 slave databases.

- Databases for database replication cannot perform online full backups.
- Currently replication of FILE data type is not supported.
- To replicate BLOB data, i.e. columns of LONG VARCHAR or LONG VARBINARY data type, set DB_BMode to 2 (BACKUP-DATA-AND-BLOB) and remember to set the BACKUP BLOB ON option while creating tablespaces.

8.3 DATABASE CONFIGURATION FILE

This section is a summary of database replication related keywords used in the **dmconfig.ini** file.

Parameters in primary side

1. Back up server

- **DB_Bmode:** Backup mode（1— data 、2— date and blob)
- **DB_BkSvr ＝1:** Start backup server
- **DB_BkDir:** Directory of backup journal files
- **DB_BkTim:** Start time of backup
- **DB_BkItv:** Backup time interval

2. Replication

- **DB_SMODE＝4** : Startup as a primary database
- **RP_Btime** : Starting time of replication
- **RP_Iterv** : Schedule for the database replication
- **RP_Sladr** : IP and port number of Slave port
- **RP_ReTry** : The times of DBMaker try to connect to remote databases after a network failure
- **RP_Clear** : Whether backup files should be cleared after transmit

Note： RP_ReTry sets how many times to retry if network connection fails. The value of RP_Clear determines whether the backup journal files should be cleared after being sent to the slave. Set RP_Clear = 1 to clear up the backup journal files. The default value is 0. If backup journal files are used for database replication only, clearing those files can reclaim some storage space. However, if a hardware crash occurs, the data from backup journal files is irretrievable and will not be able to be restored. In this case, a full backup against the slave database must be made to restore the primary database. Therefore, if backup journal files will be used for primary database backup as well, RP_Clear should be set to 0.

Parameters in slave side

- **DB_SMODE＝5** : startup as a slave database
- **RP_PRIMY** ; IP of primary side
- **RP_PTNUM** ; Port number of primary side
- **DB_BKDIR** ; Specifies the directories where the backup server puts the database backup files

The details of these keywords refer to the *Database Administrator's Guide*.

Example:

In the following **dmconfig.ini** file, the primary machine IP is 192.168.0.57, and the slave machine IP is 192.168.0.8.

The following is the minimum settings for the primary e database:

```
[RPDB] ;; Primary Config, CPU：x86, OS：WinNT, IP：192.168.0.57
;;------ Database related settings ---------
DB_DBDIR = F:\SHORY\RPDB
DB_FODIR = F:\SHORY\RPDB\FO
DB_DBFIL = RPDB.SDB
DB_BBFIL = RPDB.SBB
DB_USRBB = F:\SHORY\RPDB\RPDB.BB 2
DB_USRDB = F:\SHORY\RPDB\RPDB.DB 200
DB_JNFIL = RPDB.JNL
DB_SVADR = 192.168.0.57                ; IP address of primary machine
DB_PTNUM = 10057                       ; Port number of primary database
DB_USRID = SYSADM
;; ------ journal backup server related settings --------
DB_BKDIR = F:\SHORY\RPDB\BACKUP       ; Directory of backup journal files
DB_BMODE = 2                           ; Database start up in BACKUP-DATA-AND-BLOB
mode
DB_BKSVR = 1                           ; Start up journal backup server
DB_BKTIM = 2006/04/12 9:26:00          ; The initial backup time
DB_BKITV = 0-01:00:00                  ; Perform journal backup every 1 hour
;; ------ RP_SEND server related settings ---------------
DB_SMODE = 4 ;                         ; Start as primary database
                                       ; and also start up RP_SEND server
RP_BTIME = 2006/04/12 9:28:00          ; Initial replication time
RP_ITERV = 0-01:00:00                  ; Replicate every 1 hour
RP_SLADR = 192.168.0.8:8008            ; Replicate to slave machine with port no. 8008
RP_RETRY = 3                           ; retry 3 times if network connection fails
RP_CLEAR = 1                           ; Clear backup journal files after sending
```

The following is the minimum settings for the slave database:

```
[RPDB]  ;; Slave Config, CPU：x86, OS：Red Hat Linux T, IP：192.168.0.8
;;------ Database related settings ---------
DB_DBDIR = /HOME/DBMAKER/RPDB
DB_DBFIL = RPDB.SDB
DB_BBFIL = RPDB.DBB
DB_USRBB = /HOME/DBMAKER/RPDB/RPDB.BB
DB_USRDB = /HOME/DBMAKER/RPDB/RPDB.DB
DB_SVADR = 192.168.0.8
DB_PTNUM = 10008
DB_USRID = SYSADM
;;------ RP_RECV and RP_APPLY servers related settings --------
DB_BKDIR = /HOME/DBMAKER/RPDB/BACKUP   ; Directory of temporary backup journal files
DB_SMODE = 5             ; Start as primary database,
                         ; also start up RP_RECV and RP_APPLY servers
RP_PRIMY = 192.168.0.57  ; Receive replication data only from this machine
```

```
RP_PTNUM = 8008          ; Port number for RP_SEND and RP_RECV connection
```

8.4 DATABASE REPLICATION SETUP

To manually set up database replication:

1. **Off-line full backup in primary side.**

You should Shut down the primary database at first, then make a copy of the primary database's data files, BLOB files, and journal files.

Please refer to the *Database Administrator's Guide* for more detailed information of Database Backup—Offline full backups.

2. **Duplicate the primary database to the slave database, but can not use UNLOAD command.**

You should move the copy of files to the machine that will run the slave database, then you need have to modify the slave database's **dmconfig.ini** configuration file to reflect the corresponding file directory changes.

Please note the byte ordering of primary and slave database machines must be the same. For example, if the primary database is running on an x86 machine, the slave database machine must be byte ordering-compatible with x86; Sun Sparc's byte ordering is different from x86's, so the slave database cannot run on a Sparc machine and vice versa.

3. **Set up the journal backup server and RP_Send server on the primary database.**

All changes to a database are logged to the journal files. DBMaker therefore uses backup journal files as the source data for replication. After the journal backup server performs an incremental backup, the RP_Send server sends backup journal files to the slave database. All the transactions logged on the journal files are then committed to ensure that all the changes to the primary database will also take place on the slave side.

The RP_SEND is located on the primary database and is responsible for sending backup journal files to the slave side. RP_SEND is started and shut down automatically whenever the primary database is started or shut down.

The RP_SEND must know the IP address and port number of the machine where RP_RECV is located. Note that the port number is specifically for communication between RP_SEND and RP_RECV for data replication, and must be different from the one used by database server. RP_SEND uses the keyword **RP_SlAdr** in **dmconfig.ini** on the primary database to determine where to send replication data.

Syntax for **RP_SlAdr**:

```
RP_SLADR = {address[:port number]}
```

The default port number is 23001

4. **Set up the RP_Recv server and RP_Apply server on the slave database.**

The RP_RECV and RP_APPLY are on the slave database side. RP_RECV receives backup journal files from the primary database, and RP_APPLY executes the changes according to the journal files. The RP_RECV and RP_APPLY also start and end simultaneously with the slave database.

In order to start up RP_RECV and RP_APPLY servers, the start-up mode **DB_SMode** of the slave database should be set to 5.

Note: After the journal backup server completes an incremental backup, RP_SEND will send all the backup files that have not been sent to the slave database yet from the backup directory (**DB_BkDir**) to the slave database. Meanwhile, RP_RECV on the slave database will receive all the backup files transferred from the primary database, and put these files under the backup directory (**DB_BkDir**) on the slave database. After the files

have been received, RP_APPLY will execute the changes recorded in the backup journal files to the slave database, and the flow of database replication will be complete.

5. Start the primary and slave databases.

The start mode of the primary database is different from the slave. If you want to set a database to be a primary, set the start mode to be in primary database mode. On the other hand, if you want to set a database to be a slave, set the start mode to be in slave database mode. Using the **DB_SMode** located in the **dmconfig.ini** file can specify all of the setup options. The start mode **DB_SMode** for the primary database mode is 4. The **DB_SMode** for a slave database mode is 5.

You can start the primary and slave databases separately, in no particular order.

6. Verify the replication log (RP.LOG) and error log (ERROR.LOG).

In the process of database replication, if there are any network failures or any error messages, the system will generate a log file, ERROR.LOG, in the current database directory. Error log entries follow the following syntax:

```
yy/mm/dd hh:mm:ss Daemon name:Error number:Error message
```

Note :

1. Data in the slave database must be the same as the primary. It cannot accept data definitions (DDL, such as Create Table and Alter Table) and update data (such as INSERT, UPDATE, and DELETE). Thus, we can say that the slave database is read-only.

During the process of database replication, the slave database uses the backup journal files received from the primary database to restore data. The system will not lock data on the primary database during the process of restoring data. Therefore, queries to the slave database are a kind of *dirty read*. In other words, at any given point in time, the same query on both the primary and slave databases may return different values because RP_APPLY is restoring data.

2. We can also execute database replication immediately. We mentioned that the resident server for database replication will detect whether there are changes to data and will replicate the data automatically.

Use the dmSQL tool to enter a SQL command to have DBMaker execute the database replication immediately.

```
dmSQL> Set Flush;
```

Use this command when you need to synchronize the data between a primary and slave database.

When you execute this command, the journal backup server will immediately execute the incremental backup and backup the current journal files. Then three resident servers (RP_SEND, RP_RECV, and RP_APPLY) will follow the procedure and replicate data.

8.5 USING JSERVER MANAGER SETUP ENVIRONMENT

1. Produce a Full Backup

   First, produce an offline full backup of the primary database, and then copy the backup to the slave database. For more information on offline full backup, refer to the *Database Administrator's Guide*.

2. Primary Database Setup

To allow the database replication server to function, first set up the database so that incremental backups are being made. Refer to chapter Database Backup—Incremental backups, for more information. Next, set up the environment for the primary database.

- To setup for the primary database environment:
  - Start the **JServer Manager** application on the primary database server.
  - Click on the Setup button in the Start Database window.
  - Click the Replication tab in the Start Database Advanced Settings window.
  - To enable database replication:
    - **a)** Enter slave database IP and port numbers into the IP and Port Number of Target Database.
    - **b)** Enter a date and time into the Begin Time of Database Replication time fields.
    - **c)** Enter a value into the Times to Retry upon Failure field
    - **d)** If desired, enable Remove Backup Journal Files after Replication
    - **e)** Enter the number of days, hours, minutes, and seconds between each successive database replication in the Time Interval to Start Database Replication time fields
  - Click the Save button.
  - Click the Cancel button to return to the Start Database window.

3. Slave Database Setup

After copying a full backup of the primary database to the slave database, use JServer Manager to setup the configuration of the slave database.

- Start the JServer Manager application on the slave database server.
- Click on the Setup button in the Start Database window.
- Click the Replication tab in the Start Database Advanced Settings window.
- To enable database replication:
  - **a)** Enter the IP Address of Source Database.
  - **b)** Enter a port number for RP_RECV in the Port Number of Receive Daemon on Target DB.
- Click the Save button.
- Click the Cancel button to return to the Start Database window

8.6 EXAMPLES

The following simple example demonstrates the steps necessary to build a Database Replication.

The Primary Database is created in 192.168.0.57(x86/WinNT/DBMaker4.1.0).

The Slave Database is created in 192.168.0.8(x86/Red Hat Linux/DBMaker4.1.6).

Step 1 : Create the Primary Database

a, Add the following section into the dmconfig.ini at 192.168.0.57, and save it .

```
[RPDB]
;------basic database configuration---------
DB_DBDIR = F:\SHORY\RPDB
DB_FODIR = F:\SHORY\RPDB\FO
```

```
DB_DBFIL = RPDB.SDB
DB_BBFIL = RPDB.SBB
DB_USRBB = F:\SHORY\RPDB\RPDB.BB 2
DB_USRDB = F:\SHORY\RPDB\RPDB.DB 200
DB_JNFIL = RPDB.JNL
DB_SVADR = 192.168.0.57
DB_PTNUM = 10057
DB_USRID = SYSADM
;----backup configuration--------------------
DB_BKDIR = F:\SHORY\RPDB\BACKUP
DB_BMODE = 2
DB_BKSVR = 1
DB_BKTIM = 2006/04/12 9:26:00
DB_BKITV = 0-01:00:00
;----replication configuration---------------
DB_SMODE = 4      ; Start as primary database, and also start up RP_SEND server
RP_BTIME = 2006/04/12 9:28:00    ; Initial replication time
RP_ITERV = 1-00:00:00     ; Replicate everyday
RP_SLADR = 192.168.0.8:8008      ; Replicate to 192.168.0.8 with port no. 8008
RP_RETRY = 3      ; retry 3 times if network connection fails
RP_CLEAR = 1      ; Clear backup journal files after sending
```

b, Create the folder F:\SHORY\RPDB at 192.168.0.57, and Create the folder FO in F:\SHORY\RPDB.

c, Start the dmSQL client tool at 192.168.0.57, Perform the command “CREATE DB RPDB;”, The database RPDB is built, and it started automatically as Single-User Mode. Perform the command “TERMINATE DB;” to close the database RPDB.

Now, the files RPDB.SDB, RPDB.SBB, RPDB.DB, RPDB.BB, RPDB.JNL appeared in the folder F:\SHORY\RPDB.

Step 2 : Create the Slave Database

a, Add the following section into the dmconfig.ini at 192.168.0.8 , and save it.

```
[RPDB]
;--------------basic database configuration-----------------------
DB_DBDIR = /HOME/DBMAKER/RPDB
DB_DBFIL = RPDB.SDB
DB_BBFIL = RPDB.DBB
DB_USRBB = /HOME/DBMAKER/RPDB/RPDB.BB
DB_USRDB = /HOME/DBMAKER/RPDB/RPDB.DB
DB_SVADR = 192.168.0.8
DB_PTNUM = 10008
DB_USRID = SYSADM
;----------backup and replication configuration --------
DB_BKDIR = /HOME/DBMAKER/RPDB/BACKUP
DB_SMODE = 5      ; Start as Slave Database,
                          ; also start up RP_RECV and RP_APPLY servers
RP_PRIMY = 192.168.0.57   ; Receive replication data from this machine
RP_PTNUM = 8008    ; Port number for RP_SEND and RP_RECV connection
```

b, Create the folder /HOME/DBMAKER/RPDB at 192.168.0.8.

c, Copy the Primary Database Files(RPDB.SBB, RPDB.SDB, RPDB.DB, RPDB.BB, RPDB.JNL in the folder F:\SHORY\RPDB) to the machine to run the Slave Database (192.168.0.8, /HOME/DBMAKER/RPDB).

Step 3 : Start the Slave Database and the Primary Database

Start up the Primary Database and the Slave Database separately.

Then all changes in the Primary Database will be replicated to the Slave Database.

Note:

a, The name of the Slave Database must be same as the Primary Database.

b, Configuring the Slave Database, do not need define the size of the files. The files of the Slave Database must be copied from the Primary Database.

c, If you want to create the tablespace or add the files into the tablespace in the Primary Database. Corresponding in the dmconfig.ini of the Slave Database, defining the same logical filename and for the physical file, the filename and location is random, and do not need define the size of user file because of the size of user file lies on the Primary Database.

d, The Primary Database can not replicate the FO(file object) into the Slave Database. If both the Primary Database and the Slave Database access a shared network file, you can insert this file into the Primary Database as user FO, then you can access this USER FO in both the Primary Database and the Slave Database (only need configure DB_USRFO=1 at the Primary Database. In addition, network file must use UNC filename, e.g.: \\192.168.0.170\doc\myfile.txt) .

e, Both the Primary Database and the Slave Database can open Distributed Database Mode(DD_DDBMD=1).

f, The STORED COMMAND created in the Primary Database can be replicated into the Slave Database, but it not be used in the Slave Database, because the Slave Database not be updated.

g, Don’t replicate the STORED PROCEDURE and UDF(user defined function) into the Slave Database.

h, you can also use command set flush to execute the database replication immediately.

9. Table replication

9.1 What is Table Replication

Table replication creates a full or partial copy of a table to a destination location. This allows users in remote locations to work with a local copy of data. The local copy remains synchronized with the databases in other locations. This way each database can service data requests immediately and more efficiently, without having to go to another machine over a slower wide area network connection. This kind of request happens frequently between the headquarters and area companies or branch offices.

For example, after creating a replication from table A on Taipei's database server to table B on Tokyo's database server, modifications made to table A will replicated to table B. Clients in Tokyo can then access Tokyo's database rather than connecting to Taipei's database to acquire the same data.

Destination tables may also reside on databases on the same server. This situation may occur if data must be shared between two databases designed for different functions, or in the case of sharing data between databases of different types (for example, Oracle, Sybase, etc.). Tables that are receiving data from another database through replication will be referred to as destination tables rather than remote tables, even though the destination tables may reside on a remote database.

9.2 Table Replication Rules

- Schemas of source and destination tables must exist. This means DBMaker does not create tables when performing table replication.
- Replication names for a table must be unique.
- The subscriber name for a replication must be unique using the <*link_name*|*database_session_name*>+ <*table_owner_name*> + <*table_name*> syntax.
- Projected columns for every table must contain primary key columns.
- Primary key columns must be included with fragment columns.
- Only the table owner of the source table or a user with DBA or higher authority has the privilege to create, drop, or alter replications.
- If no column-identifier exists with the destination table, the destination column names must be the same as the base table names.
- Number of primary key columns must be equal in the base table and destination tables.

9.3 Table Replication Type

There are two types of table replication. One is synchronous. ‘Synchronous’ means the modification to the destination site shows up immediately; the destination table is modified at the same time as the local table. DBMaker uses a ‘two phase commit’ and triggers to perform synchronous table replication. Thus, after establishing a replication, any update on the source table will become a DDB (distributed database) action. This will affect the local database’s behavior. If the destination database server is unreachable, updates on the local database will fail.

The other table replication type is asynchronous. Modifications to the destination site are delayed. The delay between source and destination database depends on a user-defined schedule. Asynchronous table replication stores changes to the local table and modifies the destination table based on a schedule. In this type of replication, two databases of a replication pair are independent and can work as normal even if the network is not available.

You will find more detailed descriptions of the uses of these two types of table replication in later chapters.

9.4 Synchronous Table Replication

9.4.1 What Is Synchronous Table Replication

Synchronous table replication modifies the remote table at the same time it modifies the local table. Synchronous table replication in DBMaker uses a global transaction model, in which the replication of data to the remote table is treated as an integral part of the local transaction. This means that if the replication of data to the remote database fails, the transaction on the local table will also fail

Two-phase commit allows the synchronization of distributed data. A transaction will only be accepted if all interconnected distributed sites agree. A ‘handshake’ mechanism across the network allows distributed sites to coordinate their acceptance for each transaction. Therefore, using synchronous table replication guarantees that data will be synchronous whenever an update occurs.

9.4.2 Synchronous Table Replication Setup

Synchronous Table Replication Setup

DBMaker uses two-phase commits to perform synchronous table replication. Source and destination databases must be in distributed database (DDB) mode. Therefore, the first step is to start the databases in DDB mode (**DD_DDBMd** =1) and add database sessions to the **dmconfig.ini** file. Refer to Chapter Distributed Database-- Distributed Database Environment for more information.

After creating a table replication, any modifications (insertion, deletion, updates) to source a table will affect the destination tables.

9.4.3 Synchronous Table Replication Syntax

9.4.3.1 Creating Table Replication

CREATE REPLICATION replication-name WITH PRIMARY AS local-table-name [(column-identifier [, column-identifier] ...)]

[WHERE search-condition] REPLICATE TO

{ remote-table-name [(column-identifier [, column-identifier] ...)] [CLEAR DATA | FLUSH DATA | CLEAR AND FLUSH DATA]}

- **Note:**
  - In fact, Synchronous Table Replication is a global transaction. That is to say, if there are some failures with update in remote (for example network halt suddenly), then the update operator will fail in local.
  - Synchronous Table Replication must be set into DDB mode in dmconfig.ini file.
  - If you drop a table or a column that is referenced by an asynchronous table replication, alter a table and modify the column definition, or alter a table and add a column using the BEFORE and AFTER keywords, the synchronous replication becomes invalid and cannot be used again.
  - Drop an invalid replication to remove it from the database. Any replications created on a table are dropped automatically when dropping a table.

The CREATE REPLICATION command generates a new table replication for a table. Replications, synonyms, or views may not be created on a temporary table. Only the table owner, a DBA or SYSADM may execute the CREATE REPLICATION command.

DBMaker will replicate the entire table unless using a column list. When replicating an entire table without a column list, the columns in the local table and corresponding columns in the remote table must have the same names and data types. Columns in the local table (from left to right) will replicate to the corresponding columns named in the column list for the remote table. Specify which columns in the local table correspond to columns in the remote table by providing a column list for both the local and remote tables. In all cases, include the primary key columns in the replication and the number and data types of primary key columns in both tables must match.

The CLEAR DATA/FLUSH DATA/CLEAR AND FLUSH DATA keywords are optional. These keywords specify the operations that take place when creating a replication. The CLEAR DATA keywords delete all data from the remote table when generating the replication. The FLUSH DATA keywords copy all data that matches a search condition into the remote table. The CLEAR AND FLUSH DATA keywords clear all data from the remote table, and then copy all data that matches a search condition into the remote table.

Example 1

Suppose there is a table owner in database testddb@122, a table tuser in testddb@121. We want to replicate owner's data to tuser's and do not want to modify tuser's current data.

```
dmSQL> CREATE REPLICATION ownrep WITH PRIMARY AS owner REPLICATE TO testddb@121:tuser;
```

In the above example, we use a database session name to specify the destination database. Alternatively, a database link name could be used

Example 2

The following creates a replication named **EmpRep** for the local table named **Employee.** The remote database is identified in the database configuration section named **FieldOffice** in the local **dmconfig.ini** file. The remote table is also named **Employee** and all column names and data types in both tables are the same. And all data in the remote table is deleted and any data in the local table is replicated to the remote table.

```
dmSQL> CREATE REPLICATION EmpRep WITH PRIMARY AS Employee REPLICATE TO
       FieldOffice:Employee CLEAR AND FLUSH DATA
```

More examples of asynchronous and synchronous table replication in later chapters.

9.4.3.2 ALTER REPLICATION

Users may add destination tables to or drop destination tables from an existing table replication. The following syntax diagrams and examples demonstrate how.

Ⅰ

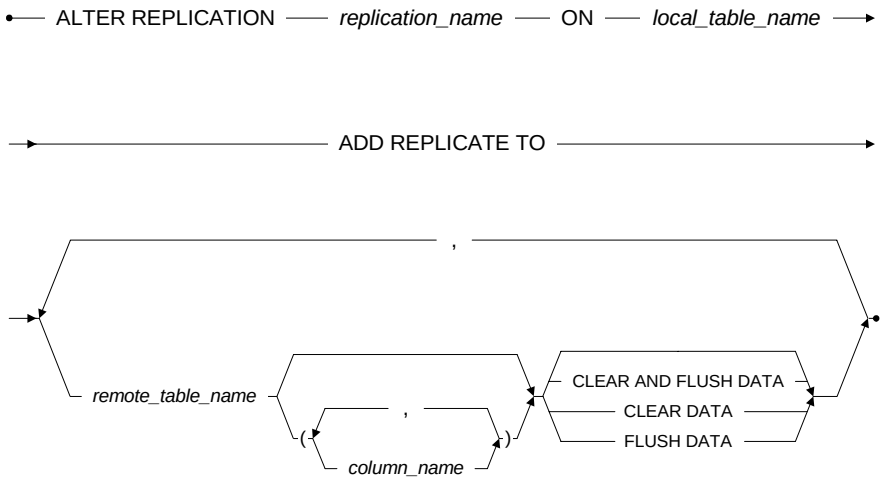

Figure 9-1 Syntax for the ALTER REPLICATION … ADD REPLICATE TO Statement

Ⅱ

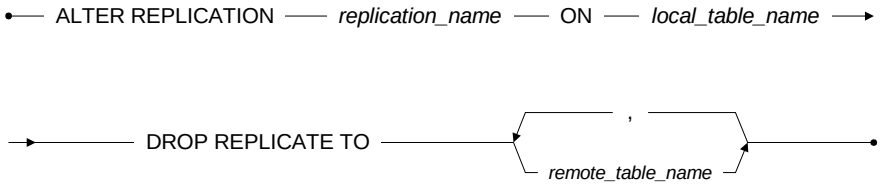

Figure 9-2 Syntax for the ALTER REPLICATION … DROP REPLICATE TO Statement

Example 1

The first command creates a replication to replicate table **creator** on the local database to table **creator** on database **sentDB**. The second command adds 2 subscribers, **copyist** on **receiveDB** and **owner** on **testddb@123**, to the replication of **multi_rep** in table **creator.**.

```
dmSQL> CREATE REPLICATION multi_rep WITH PRIMARY AS creator
       REPLICATE TO sentDB:creator;
dmSQL> ALTER REPLICATION multi_rep ON creator
       ADD REPLICATE TO receiveDB:copyist,
                   testddb@123:owner (id, name) clear data;
```

Example 2

To drop subscriber **copyist** on **dbY** from the replication of **multi_rep** for table **creator**:

```
dmSQL> ALTER REPLICATION multi_rep ON creator
       DROP REPLICATE TO dbY:copyist;
```

9.4.3.3 DROP REPLICATION

This command will drop a replication from a source table.

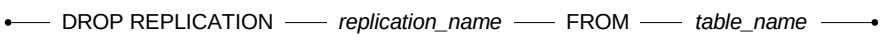

Figure 9-3 Syntax for the DROP REPLICATION Statement

Example

To drop the replication **Ownrep** from the **owner** table:

```
dmSQL> DROP REPLICATION Ownrep FROM owner;
```

9.4.4 EXAMPLES

The following simple example demonstrates the steps necessary to build a Synchronous Table Replication.

Step 1: Setup dmcongif.ini file

The first step is to setup dmconfig.ini file of the source and destination databases. You must be add keywords DD_DDBMd =1 to the dmconfig.ini file of two sides databases so that start the databases in DDB mode to accomplish the table replication.

Step 2 : Design the Environment

This example demonstrates how to replicate the table creator at the local database hstDB using Synchronous Table Replication to the remote database slvDB. The remote table is also named creator and all column names and data types in both tables are the same.

create table in local and remote database :

```
dmSQL> CREATE table SYSADM.creator(num INTEGER default null,name CHAR(10) not null)
       in DEFTABLESPACE lock mode page fillfactor 100;
dmSQL> alter table SYSADM.creator primary key ( name) in DEFTABLESPACE;
```

Note: the limiting conditions of table replication, namely projected columns for every table must contain primary key columns. The primary key of table creator is the name column.

Step 3 : Create Replication

```
dmSQL> CREATE replication RP_Sync with primary as creator ( num, name ) replicate to
       slvDB:creator ( num, name ) CLEAR AND FLUSH DATA;
```

Note: all data in the remote table is deleted and any data in the local table is replicated to the remote table

Step 4 : Check the data on both sides

After creating a table replication, any modifications (insertion, deletion, updates) to source a table will affect the destination tables.

```
INSERT :
dmSQL> insert into creator values (1,'a');           ; insert data to source table
dmSQL> select * from creator;                        ; query source table
dmSQL> select * from slvDB:creator;                 ; query destination table, check
                                                      whether replication is
                                                      successfully
UPDATE :
dmSQL> update creator set num=10 where name='a';
dmSQL> select * from creator;
dmSQL> select * from slvDB:creator;
DELETE :
dmSQL> delete from creator where name='a';
dmSQL> select * from creator;
dmSQL> select * from slvDB:creator;
```

9.5 Asynchronous Table Replication

9.5.1 What is Asynchronous Table Replication

Synchronous table replication modifies the destination table at the same time it modifies the local table, while asynchronous table replication stores changes to the local tables and modifies the destination table based on a schedule.

DBMaker uses a file known as a replication log to store changes to local tables. Modifications to local tables are stored in replication logs, and are replicated to the destination table according to a predefined schedule. Using replication logs enables DBMaker to treat the local transaction and the destination transaction independently, allowing you to update local tables normally even if the remote connection is not available. This allows asynchronous table replications to tolerate network and destination database failures, since DBMaker will keep trying until failures are corrected.

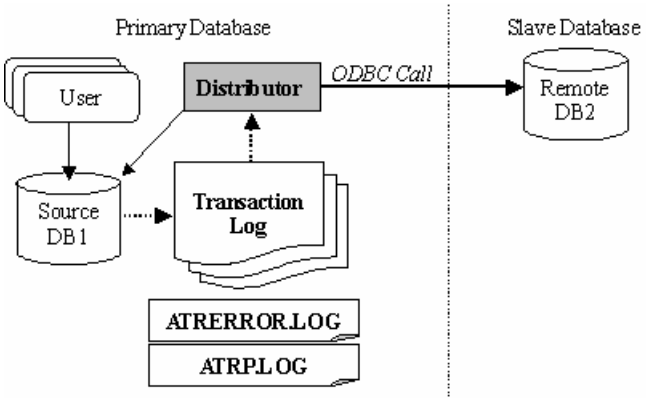

Figure 9-4: Architecture of asynchronous table replication

Asynchronous table replication uses a replication log system and the distributor server to handle data replication. Replication logs are not DBMaker journal files. The level of the replication log is higher than the journal, and it is only for replicating tables. The content of a journal is physical data modification, but the replication log consists of commands that are applied to the destination tables.

When the source database is running, DBMaker logs the modification of source tables into the replication log files. When the distributor server activates, it will redo all changes made to the source tables to the destination tables according to the replication log.

Normally the distributor server uses ODBC function calls to communicate with the destination database servers, so it is possible to replicate tables to heterogeneous database servers such as Oracle, SQL Server, Informix, etc. Heterogeneous table replication will be covered later in this chapter. Express asynchronous table replication is another type of asynchronous table replication that does not use ODBC function calls. It will also be covered in later chapter.

9.5.2 Asynchronous Table Replication Setup

The distributor server resides in the source database. The distributor connects to the destination databases periodically and performs the table replication.

The **DB_AtrMd** keyword in the ***dmconfig.ini*** file in the source database specifies whether to start the distributor server for a database. If you do not start the distributor server, the database cannot be the source for asynchronous table replications.

The **RP_LgDir** keyword in the ***dmconfig.ini*** of the source database specifies the directory where DBMaker will place replication log files for asynchronous table replication. The replication log files are binary and users should not manually remove them. The default directory of **RP_LgDir** is the subdirectory named /**TRPLOG** in the database home directory.

When creating table replications with schema checking, the distributed database mode in the source and the destination database must be enabled (**DD_DDBMd** = 1). If the schedule is set to NO CHECK, DBMaker will not check the schema, and the distributed database mode can be set to OFF (**DD_DDBMd** = 0). See the following section for more information about creating replication schedules.

Example 1

To replicate a table from the **SRCDB** database to the destination **DESTDB** database, add the following lines to the ***dmconfig.ini*** file on the source database server:

```
[SRCDB]
DB_DBDIR = /disk1/DBMaker/src
DB_USRBB = /disk1/DBMaker/src/SRCDB.BB 2
DB_USRDB = /disk1/DBMaker/src/SRCDB.DB 150
RP_LGDIR = /disk1/DBMaker/src/trplog
DB_ATRMD = 1
DD_DDBMD = 1
DB_SVADR = srcpc
DB_PTNUM = 22222

[DESTDB]
DB_SVADR = destpc
DB_PTNUM = 33333
```

*The **dmconfig.ini** file* on the destination database:

```
[SRCDB]
DB_SVADR = srcpc
DB_PTNUM = 22222

[DESTDB]
DB_DBDIR = /disk3/DBMaker/dest
DB_USRBB = /disk3/DBMaker/dest/DESTDB.BB 2
DB_USRDB = /disk3/DBMaker/dest/DESTDB.DB 150
DD_DDBMD = 1
DB_SVADR = destpc
DB_PTNUM = 33333
```

Due to the way distributor server makes use of the ODBC driver manager to perform asynchronous table replication, the ODBC data source name (DSN) of the destination database must be set if the source database is running under the Microsoft Windows environment.

9.5.3 SCHEDULE

Before creating asynchronous replication to one or more destination tables, a user with DBA privilege or higher needs to define a *schedule*. A schedule defines the starting time, the period, and the account and password used to connect to the destination database. The user can create several schedules for different destination databases on the same source database, but cannot build more than one schedule to the same destination database.

9.5.3.1 CREATE SCHEDULE

The CREATE SCHEDULE command creates a replication schedule for asynchronous table replications.

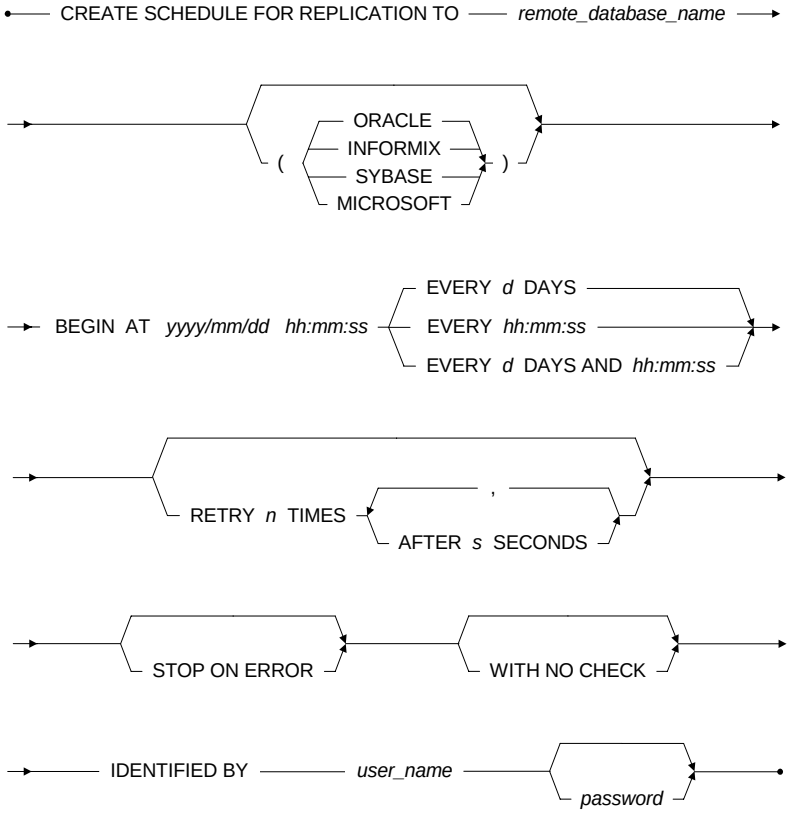

Figure 9-5 Syntax for the CREATE SCHEDULE Statement

See the description for each keyword, please refer to the SQL Command and Function Reference.

Example 1

To create a schedule for the **DESTDB** database:

```
dmSQL > CREATE SCHEDULE FOR REPLICATION TO destdb
        BEGIN AT 2000/1/1 00:00:00
        EVERY 12:00:00
        IDENTIFIED BY User Password;
```

From January 1, 2000, the distributor server will activate every 12 hours to perform an asynchronous replication. The ATRP.LOG distributor message log in the database home directory will record the starting time and status of the distributor server.

The IDENTIFIED BY keyword specifies the destination account used by the distributor server to connect to the destination database and execute the table replication. The account must have privilege to insert, delete, and update on the destination tables.

Example 2

The following creates the same schedule as the example above and also sets the number of times to retry after an error, a lock time-out, or a rollback to save point due to a full Journal to 3 times with an interval of 5 seconds between successive tries. In addition, sets the action DBMaker should take when data in the remote database has been updated in such a way that the replication cannot take place to STOP.

```
dmSQL > CREATE SCHEDULE FOR REPLICATION TO destdb
        BEGIN AT 2000/1/1 00:00:00
        EVERY 12:00:00
         RETRY 3 TIMES AFTER 5 SECONDS
         STOP ON ERROR
        IDENTIFIED BY User Password;
```

Example 3

This is a heterogeneous table replication; specify the **WITH NO CHECK** keywords to prevent DBMaker from performing schema checking on the remote database. Ensure that columns and data types in the remote table are compatible with the columns and data types in the local table the following creates the same schedule as the example above and uses the **ORACLE** keyword to indicate that the remote table is in an *Oracle 8.0* database.

```
dmSQL > CREATE SCHEDULE FOR REPLICATION TO EmpRep (ORACLE)
        BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
        RETRY 3 TIMES AFTER 5 SECONDS
        STOP ON ERROR
        WITH NO CHECK
        IDENTIFIED BY RepUser rdejpe88
```

The WITH NO CHECK keywords are optional. Since DBMaker cannot currently perform schema checking on a third-party database, use this keyword when creating a heterogeneous table replication. When using the WITH NO CHECK keywords, users must take responsibility for schema checking, and ensure that columns and data types in the remote table are compatible with the columns and data types in the local table. The WITH NO CHECK keywords are not necessary if performing a homogeneous table replication (from one DBMaker database to another DBMaker database).

9.5.3.2 DROP SCHEDULE

If the schedule is not necessary and there is no replication associated with it, users having the DBA privilege on the source database can drop it.

Figure 9-6 Syntax for the DROP SCHEDULE Statement

Example 1

To drop a schedule:

```
dmSQL> DROP SCHEDULE FOR REPLICATION TO destdb;
```

9.5.3.3 ALTER SCHEDULE

After creating a schedule, users having the DBA privilege can alter the attributes of a schedule, including the distributor's activation interval, the account used to connect to the destination database, the RETRY option and STOP/IGNORE ON ERROR option.

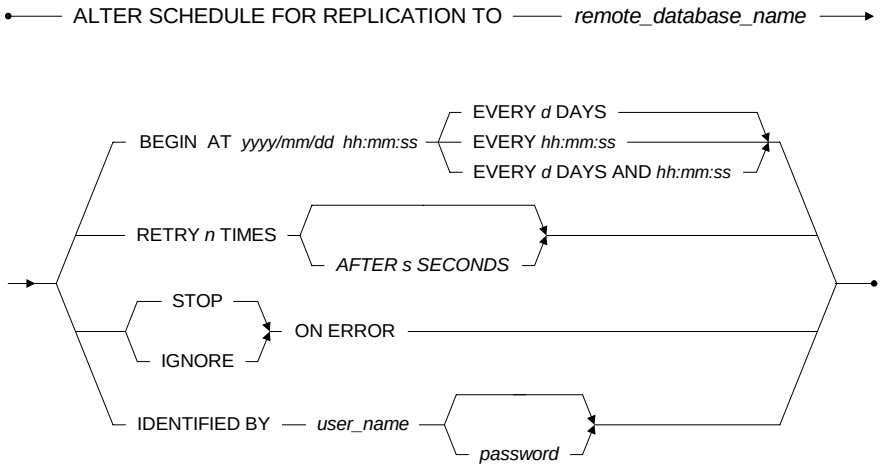

Figure 9-7 Syntax for the ALTER SCHEDULE Statement

Example 1

To alter the schedule for replications to the **DESTDB** destination database by adding the IGNORE ON ERROR option:

```
dmSQL> ALTER SCHEDULE FOR REPLICATION TO destdb IGNORE ON ERROR;
```

Example 2

```
dmSQL> ALTER SCHEDULE FOR REPLICATION TO destdb
       IDENTIFIED BY User2 Password2;
dmSQL> ALTER SCHEDULE FOR REPLICATION TO destdb STOP ON ERROR;
dmSQL> ALTER SCHEDULE FOR REPLICATION TO destdb RETRY 5 TIMES AFTER 3
       SECONDS;
```

9.5.3.4 SUSPENDING AND RESUMING SCHEDULE

If we replicate data to a database located in Tokyo, and we know that the database will be shut down on traditional holidays, we can suspend the schedule until the database is ready.

Users with the DBA privilege can suspend and resume schedules. After a schedule is suspended, the distributor server will stop trying to connect for replications.

Example 1

To suspend the schedule to the **DESTDB** destination database:

```
dmSQL> SUSPEND SCHEDULE FOR REPLICATION TO destdb;
```

Example 2

To restart the schedule for the **DESTDB** destination database:

```
dmSQL> RESUME SCHEDULE FOR REPLICATION TO destdb;
```

9.5.4 ASYNCHRONOUS TABLE REPLICATION SYNTAX

CREATE [ASYNC] REPLICATION replication-name WITH PRIMARY AS base-table-name [(column-identifier [, column-identifier] ...)]

[WHERE search-condition]

REPLICATE TO

{ remote-table-name [(column-identifier [, column-identifier] ...)] [CLEAR DATA | FLUSH DATA | CLEAR AND FLUSH DATA] [NO CASCADE]

[, remote-table-name [(column-identifier [,column-identifier,] ...)] [CLEAR DATA | FLUSH DATA | CLEAR AND FLUSH DATA] [NO CASCADE]]…}

Note:

- All asynchronous table replications depending on the same schedule will be done at the same time. The asynchronous table replication mechanism supports all data types, including LONG VARCHAR, LONG VARBINARY, and FILE types.
- Altering a table and adding a column without using the BEFORE and AFTER keywords has no impact on a synchronous replication. Asynchronous table replications are not affected when you alter a table.
- The NO CASCADE keywords are optional. It only functions when the replication type is asynchronous. The keyword specifies cascade replication. Commands flow in most organizations from the highest level to lower levels; for example, replicating data from A to B, and then B to C. This is a typical kind of cascade replication. A typical no-cascade model replicates data to B and B replicates data to A. If your data model works like this, you can turn on the NO CASCADE option. The default is setting is CASCADE. The NO CASCADE option causes table replication to occur only across one site.

Example 1

To create a replication named rpemp for the **SRCDB** database, based on the schedule to the **DESTDB** destination database:

```
dmSQL> CREATE ASYNC REPLICATION rpemp
       WITH PRIMARY AS emp
       REPLICATE TO destdb:eddit;
       CLEAR AND FLUSH DATA;
```

Users can specify CLEAR DATA, FLUSH DATA, or CLEAR AND FLUSH DATA to perform data initialization for a destination site. When creating replication, DBMaker may use a database link to connect to the destination database for checking and initialization. The action is performed through the current user account or the destination account using the DESTDB destination database or database link name.

After the asynchronous table replication has been created, the job of replicating is moved to the distributor server. The account used to connect to the destination database will be changed to the one specified by the IDENTIFIED option defined in the CREATE SCHEDULE statement.

Any transactions resulting from the asynchronous table replication will be recorded in the distributor message log ***ATRP.LOG*** under the source database home directory. The distributor message log is a pure text file, and records the start-up and actions of the distributor server.

9.5.5 SYNCHRONIZING A REPLICATION

Users will need synchronized data sometimes; to achieve this DBMaker provides synchronized schedules. The synchronized schedule forces the distributor server to perform local changes to the specified database immediately. Users do not need to wait until the schedule activates the distributor server.

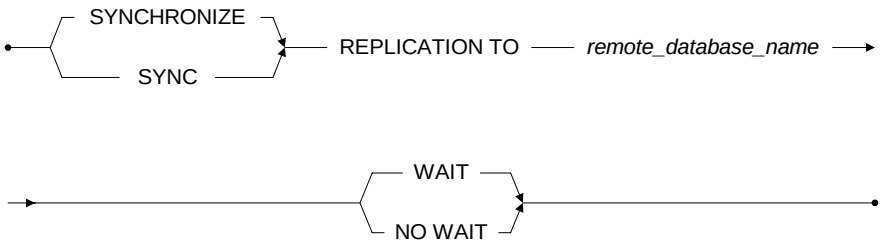

Figure 5-9 Syntax for the SYNCHRONIZE REPLICATION Statement

Example 1

To synchronize schedule to the **DESTDB** destination database:

```
dmSQL> SYNC REPLICATION TO destdb WAIT;
```

The default WAIT option causes the distributor server to wait until all changes have been made. The command returns only after the completion of replication. The NO WAIT option instructs the distributor server to perform its job immediately, and the SYNC command will return immediately.

Example 2

Using SYNC REPLICATION TO with NO WAIT:

```
dmSQL> SYNC REPLICATION TO destdb NO WAIT;
```

9.5.6 EXAMPLES

The following example demonstrates the steps necessary to build an Asynchronous Table Replication.

Step1: setup dmconfig.ini file

You may reference the section asynchronous table replication setup (chapter 9.5.2) for more information about setup dmconfig.ini file. In addition, there have a example about setup dmconfig.ini of asynchronous table replication for reference.

Since Async Table Replication is not easy to set up, the following and verifying method should be noticed on server side:

1. Add DD_DDBMD=1 to dmconfig.ini, restart up DB, submit the “select * from SYSUSER” to find out if GTRECO_D is alive
2. Add DB_ATRMD=1 to dmconfig.ini, restart up DB, submit the “select * from SYSUSER” to find out if Distributor is alive
3. Add RP_LGDIR = [some_location] to dmconfig.ini, this directory will be used to accommodate the Replication Log file.

Step2: design the environment

This example shows how to replicate the table creator at the local database hstDB using Asynchronous Table Replication to the remote database slvDB. The remote table is also named creator and all column names and data types are same as the table in local database.

Create table in local and remote database:

```
dmSQL> CREATE table SYSADM.creator(num INTEGER default null,name CHAR(10) not null)
in DEFTABLESPACE lock mode page fillfactor 100;

dmSQL> alter table SYSADM.creator primary key ( name) in DEFTABLESPACE;
```

Note:

1. The limiting conditions of table replication, namely projected columns for every table must contain primary key columns. The primary key of table creator is the name column.
2. Table name and column name can different in source table and destination table, but the column you replicated must have same data type in both sides.

Step3: create schedule

Before creating an asynchronous replication, a user with DBA privilege or higher privilege needs to define a schedule.

Create schedule in source table:

```
dmSQL> CREATE schedule FOR REPLICATION TO slvDB BEGIN AT 2006/4/11 16:00:00
       EVERY 00:01:00 IDENTIFIED BY SYSASM;
```

Setp4: create asynchronous replication

You should check if there's a file named ATRP.LOG under the DBDIR directory at first. Review its content to see that ATR starts up successfully.

Now, you can create table replication as follows:

```
dmSQL> CREATE async replication RP_Async with primary as creator ( num, name )
       replicate to slvDB:creator ( num, name ) CLEAR AND FLUSH DATA;
```

Note: use 'clear and flush data 'all data in the remote table is deleted and any data in the local table is replicated to the remote table.

Setp5: Verify the ATR Log to check if it's ok

You could manipulate the table or data, verify if there's a 1.TRP file showing up in the RP_LGDIR. In addition, you can make RP_RESET=1 to reset ATR journal file

Setp6: Check the data on both sides

You can access the table on local database.

```
INSERT:
dmSQL> insert into creator values (1,'a');            ; insert data to source table
dmSQL> select * from creator;                         ; query source table
dmSQL> select * from slvDB:creator;                  ; query destination table, check
whether replication is successfully
UPDATE:
dmSQL> update creator set num=10 where name='a';
dmSQL> select * from creator;
dmSQL> select * from slvDB:creator;
DELETE:
dmSQL> delete from creator where name='a';
dmSQL> select * from creator;
dmSQL> select * from slvDB:creator;
Synchronize the schedule to the database slvDB:
dmSQL> SYNC REPLICATION TO slvDB WAIT;          ; wait all changes have been made
dmSQL> SYNC REPLICATION TO slvDB NO WAIT;       ;Synchronize the data immediately
```

9.6 Heterogeneous Asynchronous Table Replication

9.6.1 What Is Heterogeneous Asynchronous Table Replication

DBMaker not only allows asynchronous table replication to other DBMaker databases, but also to Oracle, Informix, Sybase, and Microsoft SQL Server databases. This type of replication allows DBMaker to coexist with other databases in a heterogeneous environment, and is known as *heterogeneous table replication*.

9.6.2 Heterogeneous Asynchronous Table Replication Setup

DBMaker needs to preprocess the replicated data before sending it to a third-party destination database; users must specify the type of DBMS they are replicating to when creating a schedule in a heterogeneous environment using the ORACLE, INFORMIX, SYBASE, and MICROSOFT keywords, where ORACLE indicates a remote Oracle database, SYBASE indicated a remote SYBASE database, INFORMIX indicated a remote INFORMIX database, and MICROSOFT represents a remote Microsoft SQL Server database.

When creating a heterogeneous table replication, the CLEAR DATA, FLUSH DATA, or CLEAR AND FLUSH DATA keywords cannot be used. Manually delete or insert data in the third-party remote database to put the table in its initial state before the replication begins. In addition, performing schema checking on the third-party remote database cannot be done. Check schema to ensure that columns and data types in the remote table are compatible with the columns and data types in the local table. When creating a schedule for a heterogeneous table replication, use the WITH NO CHECK keywords to prevent DBMaker from performing schema checking. At the same time, users must take responsibility for schema checking, and ensure that columns and data types in the remote table are compatible with the columns and data types in the local table.

Due to the way DBMaker makes use of the ODBC Driver Manager to perform asynchronous table replication, the DBMaker server must be located on a computer running Windows NT or Windows 2000, and the definition of the destination database name cannot include a link name. The third-party destination databases may be located on either Windows, UNIX, or Linux platforms.

9.6.3 Heterogeneous Asynchronous Table Replication Syntax

The action to create a heterogeneous asynchronous table replication is the same as to creating a general asynchronous table replication. But users must specify the type of DBMS they are replicating to when creating a schedule in a heterogeneous environment using the ORACLE, INFORMIX, SYBASE, and MICROSOFT keywords. The Syntax for the CREATE SCHEDULE of heterogeneous asynchronous table replication Statement as follows.

CREATE SCHEDULE FOR REPLICATION TO ODBC data source name (ORACLE|INFORMIX|SYBASE|MICROSOFT) BEGIN AT yyyy/mm/dd hh:mm:ss EVERY n DAYS AND hh:mm:ss [RETRY n TIMES [AFTER n SECONDS]] [STOP ON ERROR] WITH NO CHECK IDENTIFIED BY user_name [password];

Note: ORACLE indicates a remote Oracle database, SYBASE indicated a remote SYBASE database, INFORMIX indicated a remote INFORMIX database, and MICROSOFT represents a remote Microsoft SQL Server database.

Example:

Create schedule for a heterogeneous asynchronous table replication. To connect to an Oracle database with an ODBC data source name of **EmpRep**, user **RepUser** with password **rdejpe88** enters:

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep (ORACLE) BEGIN AT
      2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
      RETRY 3 TIMES AFTER 5 SECONDS
      STOP ON ERROR
      WITH NO CHECK
      IDENTIFIED BY RepUser rdejpe88;
```

Note: When creating a schedule for heterogeneous table replication, use the WITH NO CHECK keywords to prevent DBMaker from performing schema checking.

The following shows the heterogeneous replication of table **creator**

```
dmSQL> CREATE ASYNC REPLICATION rpcopy
       WITH PRIMARY AS creator
       REPLICATE TO EmpRep: RepUser.copyist;
```

Note: The CLEAR DATA, FLUSH DATA, or CLEAR AND FLUSH DATA keywords cannot be used when creating a heterogeneous table replication. Data in the third-party destination database must be manually deleted or inserted to put the table in its initial state before the replication begins. Heterogeneous replication schedules use the same syntax as homogeneous schedules.

9.6.4 EXAMPLES

The following example demonstrates the steps necessary to build a heterogeneous asynchronous table replication.

Step1: setup dmconfig.ini file

Setup dmcongif.ini file for a heterogeneous asynchronous table replication is the same as setup dmconfig.ini file for homogeneous asynchronous table replication. Please refer to the chapter 9.5.2 to get detail.

Step2: design the environment

This example shows how to replicate the table creator at the local database hstDB using heterogeneous asynchronous Table Replication to the remote database slvDB which is created by SQL Sever. The remote table is also named creator and all column names and data types are same as the table in local database.

Create table in local database hstDB:

```
dmSQL> CREATE table SYSADM.creator(num INTEGER default null,name CHAR(10) not null)
in DEFTABLESPACE lock mode page fillfactor 100;
dmSQL> alter table SYSADM.creator primary key (name) in DEFTABLESPACE;
```

Create table in remote SQL Server database slvDB:

```
Create table creator(num int ,name char(10) not null);
```

Step2: configure the ODBC Driver Manager

DBMaker makes use of the ODBC Driver Manager to perform heterogeneous asynchronous table replication. Now, we are log on SQL Server by user shory with password shory. We need to configure the ODBC Driver Manager in local computer for remote SQL Server database. For example, we named the ODBC data source name is reptest, then specify connect server and its default database, namely is slvDB.

Step3: create schedule

Before creating an asynchronous replication, a user with DBA privilege or higher privilege needs to define a schedule.

Create schedule in source table:

```
dmSQL> create schedule for replication to reptest(microsoft) begin at 2006/1/1
       00:00:00 every 00:01:00 with no check identified by shory shory;
```

Note: you must use the WITH NO CHECK keywords to prevent DBMaker from performing schema checking when creating a schedule for heterogeneous table replication. Otherwise, you will be occur the error 7705 that is to say heterogeneous ATR cannot perform data initialization.

Setp4: create asynchronous replication

dmSQL> create async replication rpitem with primary as creator replicate to reptest:dbo.copyist;**Note:** The CLEAR DATA, FLUSH DATA, or CLEAR AND FLUSH DATA keywords cannot be used when creating a heterogeneous table replication.

In addition, you can also use the SYNC command to perform table replication immediately. Please refer to the section synchronizing A Replication for more information.

9.7 Express Asynchronous Table Replication

9.7.1 What Is Express Asynchronous Table Replication

Asynchronous table replication uses ODBC function calls to communicate with destination databases, which might cause poor performance in a WAN environment. To achieve better performance on a WAN, DBMaker provides another mechanism named express asynchronous table replication. DBMaker packs commands into a package to travel the network.

Since other database management systems do not support this protocol, express asynchronous table replications cannot work with on heterogeneous replications. It also does not support the STOP ON ERROR option when creating express schedules.

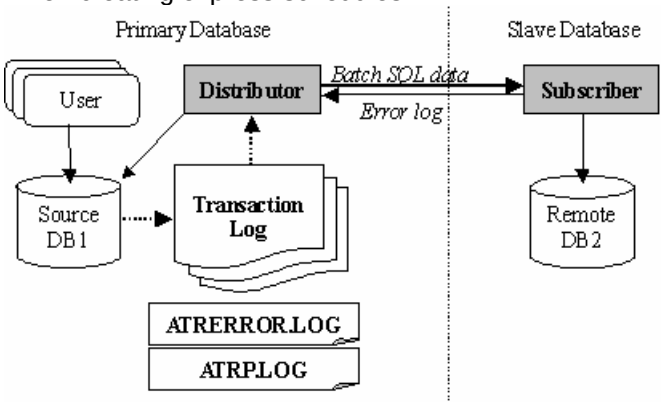

Figure 5-1: Architecture of express asynchronous table replication

There is a distributor server on the source database and a subscriber daemon on the destination database. They co-operate to do the work for express asynchronous table replication. The distributor does not directly apply the changes from the source table to the destination table via ODBC calls. Instead, it only packages the SQL commands and related data applied on the source table, and sends the packet out to the subscriber daemon on the destination database. After getting the packet on the destination database, the subscriber daemon applies the commands to the destination tables.

9.7.2 Express Asynchronous Table Replication Setup

To build an express asynchronous table replication, you should setup the distributor server on the source database and subscriber daemon on the destination database(s) at first. In order to start the subscriber daemon, the **DB_EtrPt** keyword in the ***dmconfig.ini*** file for the destination

(subscriber) database should be set. It specifies the port number of the communicating channel between the distributor server and the subscriber daemon. Then you must uses the EXPRESS option specified in the CREATE SCHEDULE command to create an express schedule for the destination database(s). Finally, create the asynchronous table replication(s) based on the schedule.

9.7.3 EXAMPLES

Step 1: Enable the distributor server on the source database and subscriber daemon on the destination database(s).

Example 1

To replicate a table from the **SRCDB** source database to the **DESTDB** destination database by express replication, the subscriber daemon should be started on the destination database.

A sample ***dmconfig.ini*** file in a destination database follows:

```
[SRCDB]
DB_SVADR = srcpc    ; tell the target database where the source
DB_PTNUM = 22222    ; database is
[DESTDB]
DB_DBDIR = /disk3/DBMaker/dest
DB_USRBB = /disk3/DBMaker/dest/DESTDB.BB 2
DB_USRDB = /disk3/DBMaker/dest/DESTDB.DB 150
DD_DDBMD = 1
DB_SVADR = destpc
DB_PTNUM = 33333
DB_ETRPT = 44444    ; port number used by Subscriber Daemon
```

In the source database, the **DB_AtrMd** keyword is used to start the distributor server. It is not necessary to let the distributor know which port number the subscriber daemon uses for the destination database.

A sample ***dmconfig.ini*** file from the source database follows:

```
[SRCDB]
DB_DBDIR = /disk1/DBMaker/src
DB_USRBB = /disk1/DBMaker/src/SRCDB.BB 2
DB_USRDB = /disk1/DBMaker/src/SRCDB.DB 150
RP_LGDIR = /disk1/DBMaker/src/trplog
DB_ATRMD = 1
DD_DDBMD = 1
DB_SVADR = srcpc
DB_PTNUM = 22222
[DESTDB]
DB_SVADR = destpc
DB_PTNUM = 33333
```

Step 2: Create an express schedule to the destination database(s).

Express asynchronous table replication uses the EXPRESS option specified in the CREATE SCHEDULE command.

Example

To build a schedule for express asynchronous table replication:

```
dmSQL> CREATE SCHEDULE FOR EXPRESS REPLICATION TO destdb
          BEGIN AT 2000/1/1 00:00:00
          EVERY 12:00:00
          IDENTIFIED BY User Password;
```

Step 3: Based on the schedule, create the asynchronous table replication(s).

The steps are the same as for creating asynchronous table replications. Refer to the section *Asynchronous Table Replication*, or the *SQL Command and Function Reference* for more information.

9.8 Table Replication System Catalog

The system catalog is a set of tables that contains information on all objects in the database. The following table lists all of the system catalog tables on table replication in DBMaker, and a brief description of what is contained in each table.

In addition, the SYSUSER table have no direct relations, but you may get information on the status of all users currently connected to a database by querying the SYSUSER table.

TABLE NAME	CONTENTS
SYSDBLINK	Database link information
SYSOPENLINK	Open link information
SYSPUBLISH	Table replication source information
SYSSUBSCRIBE	Table replication destination information
SYSTRPDEST	Table replication information
SYSTRPJOB	Information for recording all jobs to be replicated
SYSTRPPOS	Information for Distributor to prune transaction log files
SYSUSER	Information on users logged into the database

SYSDBLINK

The SYSDBLINK table contains information on remote database links.

COLUMN NAME	DESCRIPTION
OWNER	Link Owner
DB_LINK	Link Name
DBSVR	A database section, which contains the remote database information.
USER_NAME	User Name in the remote database

SYSOPENLINK

The SYSOPENLINK table contains information on open database links.

COLUMN NAME	DESCRIPTION
DB_LINK	Open link.
DBSVR	Server.
USER_NAME	User name.
TXN_STATUS	Transaction status. 'R' — Read 'W' — Write 'N' — No transaction

SYSPUBLISH

The SYSPUBLISH table contains information on table replication sources.

COLUMN NAME	DESCRIPTION
REPLICATION_NAME	Name of replication.
TYPE	S — Synchronous A — Asynchronous.
TABLE_OWNER	Owner of table being replicated.
TABLE_NAME	Name of table being replicated.
NUM_PROJECT	Number of projected columns.
FRAGMENT	Fragment string.
NUM_SUBSCRIBER	Number of subscribers.

SYSSUBSCRIBE

The SYSSUBSCRIBE table contains information on table replication targets.

COLUMN NAME	DESCRIPTION
BASE_TABLE_OWNER	Base table owner.
BASE_TABLE_NAME	Base table name.
REPLICATION_NAME	Replication name.
DB_LINK	Database link.
TABLE_OWNER	Table owner.
TABLE_NAME	Table name.

SYSTRPDEST

The SYSTRPDEST table contains information on schedules used by asynchronous table replication.

COLUMN NAME	DESCRIPTION
SVRNAME	Remote database name.
USER_NAME	User account in the remote database.
STATUS	Status of the remote database. (0: normal; 1: suspend)
BEGTIME	Beginning time of replication.
INTERVAL	Interval between replicating.

SYSTRPJOB

The SYSTRPJOB table contains information on logs used by asynchronous table replication.

COLUMN NAME	DESCRIPTION
DESTINATION	The database to which data is replicated.
FN	The file numbers of a transaction log record.
OFFSET	The Offset in the transaction log record.

SYSTRPPOS

The SYSTRPPOS table contains information used by asynchronous table replication.

COLUMN NAME	DESCRIPTION
POSARRAY	Internal usage.

SYSUSER

The SYSUSER table contains information on the status of all users currently connected to a database. Before killing a connection, you should query the SYSUSER table for the ID of the connection you want to kill. If your login host name is not registered in the network, LOGIN_HOST is **anonymous**.

COLUMN NAME	DESCRIPTION
CONNECTION_ID	Connection ID.
USER_NAME	Login user name.
LOGIN_TIME	Login time.
LOGIN_IP_ADDR	Login IP address.
LOGIN_HOST	Login host name.
NUM_SCAN	Number of **SELECT** operations.
NUM_INSERT	Number of **INSERT** operations.
NUM_UPDATE	Number of **UPDATE** operations.
NUM_DELETE	Number of **DELETE** operations.

COLUMN NAME	DESCRIPTION
NUM_TRANX	Number of processed transactions.
NUM_JBYTE_PER_TRAN	Number of journal bytes per transaction.
FIRST_W_JNR_FN	First journal file number of one active transaction.
FIRST_W_JNR_BN	First journal block number of one active transaction
NUM_BYTE_JNR_DATA	Total journal bytes used in the active transaction
NUM_J_BLOCK_DURATN	The span between first journal block number used by the active transaction and one used most recently.
SQL_CMD	The most recently executed SQL command and the command status. The command status could be the following: [PRE] - The SQL command is preparing. [EXEC] - The SQL command is executing from a SQLExecute call. [EXDIR] - The SQL command is executing from a SQLExecDirect call. [FETCH] - The operation is in a fetch data phase. [EXIT] - The SQL command has finished to prepare, execute or fetch operation.
TIME_OF_SQL_CMD	The time when the most recently used SQL command was executed.

9.9 Table Replication Error Handling

In the process of data replication, the distributor may face five kinds of errors: warning, connection, data, statement, and transaction.

- Warning

For example, if a CHAR(10)data type is replicated to a CHAR(5) column type, there will be a data truncation warning. The distributor server will ignore these kinds of errors.

- Connection Errors

If the distributor server fails to connect to the destination database server, it will abandon the schedule and wait until the next time. All jobs will be kept until then.

- Errors

Because asynchronous table replication is loosely coupled, it is possible that someone else will update the destination data first. For example, the distributor server wants to insert a record to a destination database, but it already exists. Another situation could be that the distributor server wants to delete a record, but it does not exist. Users can use the STOP ON ERROR option to

make the distributor server stop when it incurs such errors. The default behavior of the distributor is to ignore these kinds of errors.

NOTE: *The data error mentioned above includes an integrity violation error and affected row error. The integrity violation error calls function SQLError() and returns the '23000'error state. The affected row error calls function SQLRowCount(), the result is not one. For more information on ODBC functions, refer to the "ODBC Programmer's Guide".*

- Statement Errors

When the distributor server faces lock time-out errors when executing a statement, it needs to wait and retry using the RETRY <*n*> TIMES option. The AFTER <*s*> SECONDS option specifies how long to wait before the next try.

- Transaction Errors

For example, a dead lock causes transactions belonging to transaction errors to be rolled back. If the distributor server faces errors that rollback transactions, it will retry once for each whole transaction. If the outcome still fails, the distributor will leave these actions for the next schedule.

Users can check a text file named ***ATRERROR.LOG*** for error records that occurred during replication. This file is located in the home directory of the database.

Appendix Tables used in the Examples

The following table shows all the tables that used in the previous samples, including the detail information of tables. It can help user to understand the sample easily.

Table name	Table definition	Table description
TUSER	create table SYSADM.USER(ID INTEGER NOT NULL, NAME CHAR(20) DEFAULT NULL, USETIME TIME DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database testddb@121, testddb@122, testddb@123,
OWNER	create table SYSADM.OWNER(ID INTEGER NOT NULL, NAME CHAR(20) DEFAULT NULL , OWNTIME TIME DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database testddb@122, testddb@123,
LINKER	create table SYSADM.LINKER(ID INTEGER NOT NULL, SEX CHAR(4) DEFAULT NULL, NAME CHAR(20) DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database testddb@121, testddb@122, testddb@123,
CREATOR	create table SYSADM.CREATOR(NUM INTEGER DEFAULT NULL, NAME CHAR(10) NOT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database sentDB, hstDB, slvDB
COPYIST	create table SYSADM.COPYIST(NUM INTEGER DEFAULT NULL, NAME CHAR(10) NOT NULL)	The basic table create in database receiveDB, slvDB, Emprep of Oracle,

Table name	Table definition	Table description
	IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	Reptest of Microsoft SQL Server, copyist is a copy of table creator used in Replication Table.
CUSTOMER	Create SYSADM.CUSTOMER(ID INTEGER NOT NULL, MONEY INTEGER DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database BankTrank
EMPLOYEE	Create table SYSADM.EMPLOYEE(DEPT_ID INTEGER NOT NULL, SALARY INTEGER DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 100;	The basic table create in database testddb@121, Fieldoffice