Multi-Master Replication

CHAPTER 6

In this chapter we will explore the most powerful type of replication that Oracle offers, Multi-Master replication (MMR). We'll look at the features of MMR and what it can offer you before mapping the steps involved in planning MMR replication. We'll then be ready to see an example of MMR replication in action. Finally, we will look at various MMR maintenance issues that you will want to be aware of.

An Introduction to Multi-Master Replication

You have already seen how to create and use read-only and updatable materialized views. They offer the powerful ability to replicate data in tables across separate databases. With multi-master replication, you can replicate more than just database tables. You can replicate:

- Tables
- Indexes
- Procedures, functions, and triggers
- Packages
- User-defined types (Oracle9i)

As always, there are plusses and minuses to using multi-master replication. The positive benefits of MMR include the following:

Replicates more objects, including user-defined objects.
Updates or modifies the objects being replicated. Adding a column to a table at the master definition site can be replicated to other master sites.
Replicates with any number of other databases. Any master site can replicate with other master sites, updatable Mview sites, and read-only Mview sites.

However, there are some downsides such as:

- Potentially large network bandwidth requirements. Not only does multi-master push and pull changes between sites, it also sends acknowledgements and quite a bit of administrative data.

- Reduced Performance. Complexity and robustness comes at a price. MMR involves the use of triggers and procedures, and this can result in a database performance hit. Depending on how much data you are replicating, this performance hit can be substantial.
- Significant increases in administration requirements. When problems appear in the database, the DBA must insure that replication is not the cause or that the cause is not replicated to other databases. Database performance tuning and problem resolution becomes more complicated by an order of magnitude.
- Database changes require additional planning. Rolling out a new version of an application can be much more difficult. Each new version will require revisiting the design of the replication.

The considerations above should reinforce the earlier recommendation not to implement a higher level of replication than you need.

Multi-master replication is powerful, and it is complicated to create and monitor the replication environment. Because of this complexity, multi-master replication requires some additional planning.

Planning Multi-Master Replication

There are a number of considerations when planning a MMR site. First, you need to decide what you are going to replicate. Then you need to configure your tables for replication.

Begin your preparation by deciding what data needs to be replicated. Because of the bandwidth requirements to support multimaster replication, you should replicate only those objects that are required at the remote sites. Again, only use MMR on those objects that require MMR.

A master definition site can contain any number of master groups. The criterion for grouping replication objects into groups is to insure data integrity. If different objects replicate to different master sites, they should be placed in separate groups. Unlike updatable Mviews, every object in a master group will replicate to all other master sites that support that group.

Configuring Your Tables For Replication

Before you can start replicating tables with MMR, you need to make sure your table's logical design will support MMR.

Let's take a look at the following configuration issues:
Establishing primary keys
Establishing foreign keys
Establishing database parameters
Validating that the replication packages are installed
Creating replication administration accounts
Creating database links
Creating push and purge jobs

Configuring Primary Keys

To replicate tables, the database must identify rows uniquely. This is normally accomplished by using primary keys. If a table does not have a primary key defined, you can identify a column or set of columns to uniquely identify rows. You need to understand that defining a row or set of rows essentially defines a primary key on that table. We always recommend that you define primary keys on all tables used for replication to avoid confusion later on.

Using Sequences to Create Primary Keys

If a table simply has no row or group of rows that uniquely identifies it, you will need to use a sequence to establish the primary key. The sequence creates a special type of primary key known as a pseudo key (or surrogate key).

Using sequences as primary keys introduces a data integrity problem, as multiple sites add rows to the table. If every site starts the sequence at 1, the first row will contain a key of "1". When another site adds a row it will also assign a key of "1". This will create a key conflict that must be remedied.

There are two easy ways to avoid this conflict. One is to have each master site begin the sequence at a different number. For example Site 1 starts a 1, Site 2 starts at 100,000, and Site 3 starts at 200,000. For the first 99,999 rows inserted at one site this plan works. However, the 100,000th row from any site will create a key conflict.

Another method is for each site to start all the sequences at 1, but to concatenate the site name to the key. Site NAVDB would thus insert rows with the primary key being NAVDB1, NAVDB2, and so forth. The MYDB site would use MYDB1, MYDB2, and so forth. This method provides a more flexible solution, especially if additional master sites may be added at a later

date. Conflict resolution is covered in greater detail in Chapter 7, Conflict Resolution.

A table with a primary key defined will have a primary key index that the database uses to enforce the constraint. This index may be defined by the user or created by the database. When you add the table to the replication group, you do not have to include the primary key index. When the table is replicated to another master site, the remote site will automatically build the primary key index.

Foreign Keys

Foreign keys are used to enforce referential integrity. Normally, an index is created on the foreign key column of the child table to keep from having to execute a full table scan on the child table every time the parent table is updated. By adding the foreign key index to the master replication group, you do not have to replicate the child table and can still enforce the referential integrity. An update to the child table will update the foreign key index, which is replicated to the remote sites. However, if you replicate a child table, you must replicate the parent to maintain the integrity.

Database Parameters

The only additional change to the database init.ora parameters from updatable materialized view replication is to add an additional 80M to the shared pool size on all the master sites. Below is a quick synopsis of init.ora changes to support multi-master replication.

Parameter Name	Default Value	Recommended Value
COMPATIBLE	Depends on the version of Oracle that you are using.	Set compatible to the version of Oracle that you are using in order to use all replication features of that version of the database
DB_DOMAIN	.WORLD	This is the extension component of the local databases Global

		Name. If not defined, it will default to ".WORLD"
DISTRIBUTED_ TRANSACTIONS	.25 * the parameter setting for transactions	Add 5 + 2 per master to the existing value. Note this is obsolete in Oracle9i and later.
GLOBAL_ NAMES	FALSE	GLOBAL_NAMES must be set to TRUE in each database that will be involved in advanced replication.
JOB_QUEUE_ PROCESSES	0	This parameter must be set to a value of at least one. Higher values will allow more parallel replication of objects. We recommend 3 + 1 per additional master.
OPEN_LINKS	4	OPEN_LINKS defines the number of concurrent database links that are required for a given database. This parameter needs to be configured for an intial setting of 4 + 2 additional links for each master site.
PARALLEL_ AUTOMATIC_ TUNING	FALSE	Oracle9i offers this parameter to help establish the correct level of parallelism. Set to TRUE to allow Oracle to determine the best configuration for parallel operations.
PARALLEL_ MAX_ SERVERS	Derived based on the parameters: • CPU_COUNT • PARALLEL_	Only important if you need parallel propagation, which is recommended. You should configure this parameter's value high enough to allow sufficient parallel servers to

	• AUTOMATIC_ • TUNING • PARALLEL_ • ADAPTIVE_ • MULTI_USER	be started. Generally, the default is sufficient.
PARALLEL_MIN_SERVERS	0	Set this value to the number of parallel streams that you are expecting. We suggest 2.
PROCESSES	Derived from the value of the parameter parallel_max_servers	Add at least 12 to the current value.
REPLICATION_DEPENDENCY_TRACKING	TRUE	Should be set to the default value.
SHARED_POOL_SIZE	OS Dependent	Add at least 80m to the shared pool for most MMR replication instillations.

Verify the Replication Packages Are Loaded

As with all forms of advanced replication, the replication packages must be valid. Ask SYS or SYSTEM to execute the following query to identify invalid objects in the database.

```
SELECT count(*)
FROM   dba_objects
WHERE status = 'INVALID'
AND owner IN ('SYS', 'SYSTEM');
```

For additional information concerning installing and verifying the replication packages, refer to Chapter 2, Preparing to Use Replication.
Remember that the user repadmin administers each master site. Make sure that you are working on the correct database site when conducting maintenance or other administration tasks.

Creating the User Repadmin

Just as with updatable Mviews, the replication administrator is a user called repadmin. The repadmin user is created in the same way and granted the same privileges on all master sites. Passwords can be different at each site for additional security.

```
create user repadmin identified by repadmin;
grant connect, resource to repadmin;
execute dbms_repcat_admin.grant_admin_any_schema('repadmin');
grant comment any table to repadmin;
grant lock any table to repadmin;
execute dbms_defer_sys.register_propagator('repadmin');
```

Normally, repadmin is the administrator, receiver, and the propagator. These are three distinct functions:

The administrator maintains the master group, adds or removes objects, etc.
The propagator is responsible for pushing items in the deferred transaction queue to all other master sites.
The receiver takes items that have arrived in the deferred transaction queue and applies them to the local objects.

Oracle recommends (and we agree!) that you use repadmin to perform all three tasks when establishing your replication environment. For additional security, you can establish a separate user as the receiver and propagator, similar to the way repproxy was used in Chapter 4, Updatable Materialized Views. When you use separate users to perform these tasks, you will implement either the trusted or untrusted security model.

Trusted and Untrusted Security Models

If you decide to create a separate user as the propagator and/or the receiver, you can implement either the trusted or untrusted security model. The trusted

model allows the propagator/receiver to support all master groups on all systems. The untrusted model assigns specific groups to a receiver and the receiver only has access to those specific groups. Regardless of the model you choose, there are a few rules that can not be violated:

- A master site can have only one propagator.
- A propagator has the "execute any procedure" grant.
- A master site can have multiple receivers.
- A master group can have only one receiver per master site.
- A receiver is not granted "execute any procedure".

So there are three main options for establishing our replication admin users:

- The repadmin does it all approach
- The trusted model
- The untrusted model

Let's look at each of these in a bit more detail next.

Repadmin Does All - The recommended approach

Most multimaster replication uses one admin user, repadmin, who acts as administrator, propagator, and receiver. Since repadmin is a propagator, it has the "execute any procedure" grant, and therefore has access to any procedure in the database, regardless of whether it is part of replication. Repadmin is on every master site with these privileges and has access to all replication groups, objects, and procedures.

Trusted Model - Separate User for each Task

In the trusted model, there are three users that each perform a separate task. Repadmin administers the replication, but since he is not a propagator, he is not granted "execute any procedure". Another user, we'll call him repprop, is the propagator. This user has the "execute any procedure" privilege only on his master site, and he has no access to anything on other master sites. A third user we'll call reprecv, is defined as the receiver for only that master site. Reprecv is the receiver for all master groups on that master site. This model works well when all master groups are propagated to all master sites. There is no reason to define multiple receivers since they will each have access to all master groups on the site.

Untrusted Model - Separate User for each Task, Multiple Recievers

The difference between the trusted and untrusted model is that in the untrusted model all master groups are not replicated to all master sites. Since there is only one propagator per master site, and he will try to replicate all the master groups to all the master sites, it is the receiver that determines which master groups are applied at each master site.

In Figure 6.1 the master definition site has three master groups. REP_GROUP1 replicates with master sites A and C. REP_GROUP2 replicates with master sites A, C, and D. This configuration can use a single administrator (repadmin performs all the tasks) on the master definition site and on master site A. But master sites B, C, and D must implement the untrusted model to limit the master groups each site replicates.

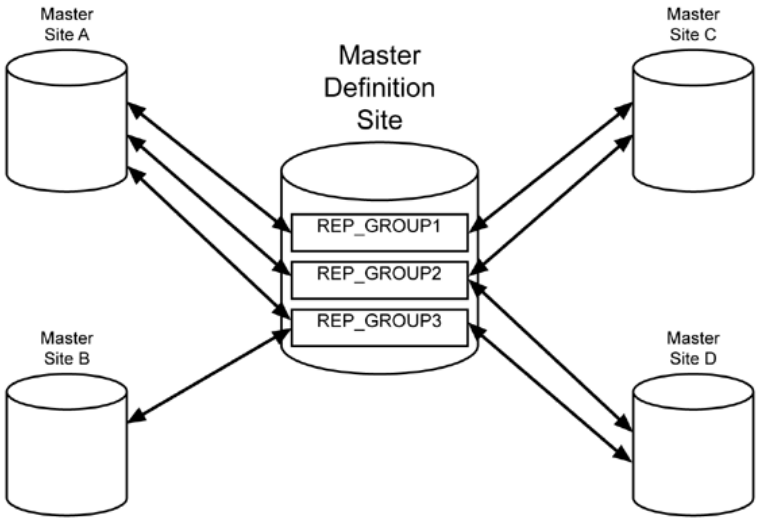

Fig 6.1: *Untrusted Security Model*

Master site B will implement a receiver that only receives REP_GROUP3. Master site C will implement a receiver that only receives REP_GROUP1 and REP_GROUP2. Master site D will implement a receiver that only receives REP_GROUP2 and REP_GROUP3.

Add multiple master definition sites and you can see that multi-master replication can become quite complicated. One significant advantage of both

the trusted and untrusted models is that no user has the "execute and procedure" grant on a remote site. All three users only have grants to their own master site.

Creating the Trusted and Untrusted Models

Let's create an example of a trusted model. Here is the SQL used to create the different user accounts. You will execute this SQL on all master sites:

```
connect system/?????

create user repadmin identified by repadmin;
grant connect, resource to repadmin;
execute dbms_repcat_admin.grant_admin_any_schema(
          'repadmin');
grant comment any table to repadmin;
grant lock any table to repadmin;

create user repprop identified by repprop;
grant connect, resource to repprop;
execute dbms_defer_sys.register_propagator(
          'repprop');

create user reprecv identified by reprecv;
grant connect, resource to reprecv;
execute dbms_repcat_admin.register_user_repgroup( username => 'reprecv',
privilege_type  => 'receiver',   list_of_gnames => NULL);
```

Here, we created the three users, each having separate tasks, and none having access to remote sites.

If you want to implement the untrusted model, create repadmin and repprop, just as we did in the previous example. Then create two master groups and the receiver. The master group must exist before making a user a receiver. In this example, we register user reprecv as a receiver for the master groups REP_GROUP2 and REP_GROUP3.

```
connect system/?????

create user repadmin identified by repadmin;
grant connect, resource to repadmin;
execute dbms_repcat_admin.grant_admin_any_schema(
          'repadmin');
grant comment any table to repadmin;
grant lock any table to repadmin;

create user repprop identified by repprop;
```

```
grant connect, resource to repprop;
execute dbms_defer_sys.register_propagator(
          'repprop');

create user reprecv identified by reprecv;
grant connect, resource to reprecv;

-- First create the groups.
execute dbms_repcat.create_master_repgroup(
          gname=> 'REP_GROUP2');

execute dbms_repcat.create_master_repgroup(
          gname=> 'REP_GROUP3');

-- Now register the reciever.
execute dbms_repcat_admin.register_user_repgroup(
  username => 'reprecv',
  privilege_type  => 'receiver',
  list_of_gnames => 'REP_GROUP2, REP_GROUP3');
```

Finally, we need to discuss the user who owns the data being replicated. In this example, we are going to replicate the PUBS schema from the NAVDB.WORLD to the PUBS schema in MYDB.WORLD. The PUBS schema on the remote sites will start out empty. No special grants are required for the schema owner.

pubs_db.sql

The PUBS schema is detailed in the appendix and available in the code depot.

Creating Database Links

The next step in preparing for multimaster replication is establishing the database links. Links must be established in both directions between master sites. Database links are covered in Chapter 2, Preparing To Use Replication. Here, we are going to create two private links between the two repadmin users.

```
connect repadmin/repadmin@navdb.world

create database link MYDB.WORLD
  connect to repadmin identified by repadmin
  using 'MYDB.WORLD';

connect repadmin/repadmin@mydb.world

create private database link NAVDB.WORLD
  connect to repadmin identified by repadmin
```

```
  using 'NAVDB.WORLD';
```

If there are other master sites in the replication setup, create links in both directions to all other master sites. Note that this is different from the previous forms of replication with materialized views. Since MMR is a two-way form of replication, you need database links going in both directions.

If you are using either the trusted or untrusted model (discussed above), you must create the database links from the local propagator to the registered receiver at the remote site. Repadmin does not need a database link if he is not acting as a propagator or receiver. Here, we are creating the database links to support the examples later in this book:

```
connect repprop/repprop@navdb.world

create database link MYDB.WORLD
  connect to reprecv identified by reprecv
  using 'MYDB.WORLD';

connect repprop/repprop@mydb.world
```

```
create database link NAVDB.WORLD
  connect to reprecv identified by reprecv
  using 'NAVDB.WORLD';
```

You only need to link each master site with one paired master site (though from a HA redundancy point of view this may be desirable). Each master site does not need to be linked to all other master sites.

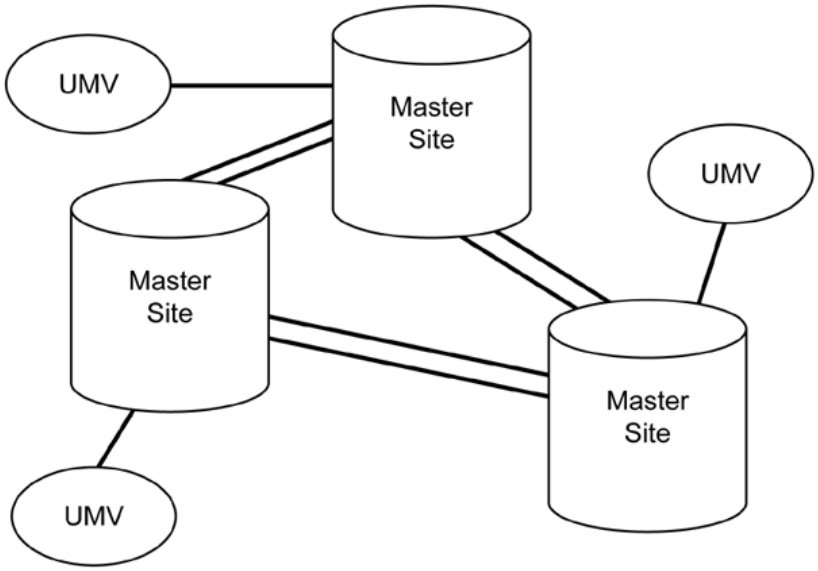

Fig 6.2 *Linking Master Sites*

Database links can also be created in Oracle Enterprise Manager. Please refer to Chapter 2, Preparing To Use Replication, for additional details.

Creating Push/Purge Jobs

All changes to replication objects are applied at the local site and placed in the deferred transaction queue. We need a job to periodically push those changes to other master sites. We also need a job to remove the changes from the transaction queue after they have been pushed.

These two tasks are split into two jobs to increase efficiency. The push needs to happen as quickly and efficiently as possible. The purge needs to happen often enough to keep the queue manageable. We create the push/purge jobs using the *schedule_push* and *schedule_purge* procedures in the Oracle supplied *dbms_defer_sys* package. We will push every minute and purge every hour. If the system has a high transaction rate, you might want to purge more often.

MM_PushPurge.sql

```
-- Add jobs to NAVDB
connect repadmin/repadmin@navdb
```

```
begin
  dbms_defer_sys.schedule_push(
    destination => 'MYDB.WORLD',
    interval => 'SYSDATE + 1/(60*24)',
    next_date => sysdate,
    stop_on_error => FALSE,
    delay_seconds => 0,
    parallelism => 1);
end;
/

begin
dbms_defer_sys.schedule_purge(
  next_date => sysdate,
  interval => 'sysdate + 1/24',
  delay_seconds => 0,
  rollback_segment => '');
end;
/



-- Add jobs to MYDB
connect repadmin/repadmin@mydb

begin
  dbms_defer_sys.schedule_push(
    destination => 'NAVDB.WORLD',
    interval => 'SYSDATE + 1/(60*24)',
    next_date => sysdate,
    stop_on_error => FALSE,
    delay_seconds => 0,
    parallelism => 1);
end;
/

begin
dbms_defer_sys.schedule_purge(
  next_date => sysdate,
  interval => 'sysdate + 1/24',
  delay_seconds => 0,
  rollback_segment => '');
end;
/
```

Although there is no way to create purge/push jobs in OEM, you can edit them. Under the Distribution->Advanced Replication -> Schedule tabs, you will find both the push and purge jobs. Modify them by selecting the edit button or changi the parameters and selecting apply. Below, we changed the push job to execute every 10 seconds.

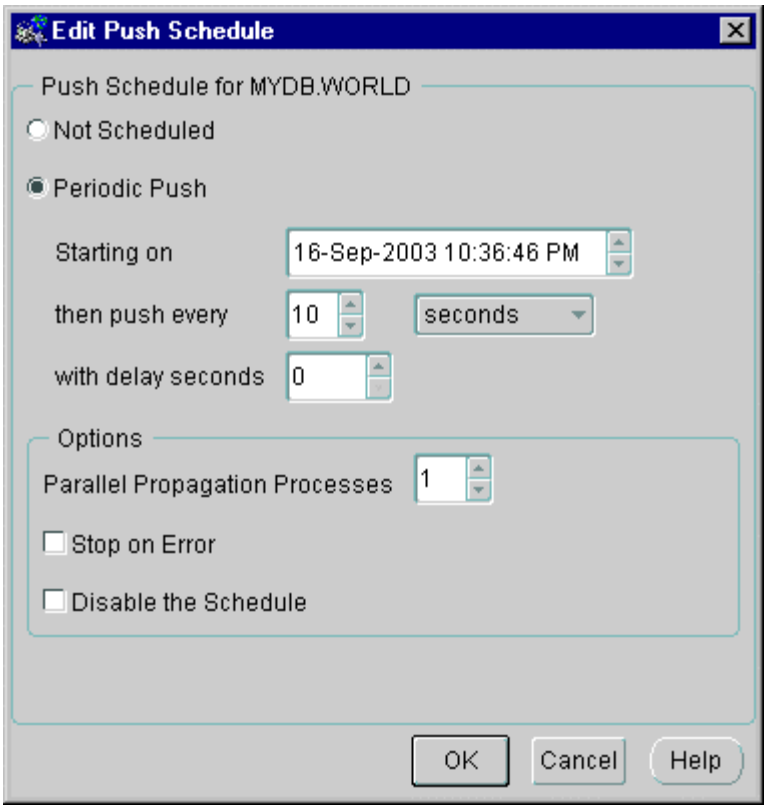

Fig 6.3 *OEM Edit Push Job*

Now that we have determined what we want to replicate, it is time to create the Master Definition Site on NAVDB.WORLD and then replicate it on MYDB.WORLD. This can be done using the command line or with Oracle Enterprise Manager.

Setting Up MMR – By Example

In this section, we'll demonstrate how to set up MMR with an example. Let's become a little more familiar with the master definition site before creating the master replication group.

The Master Definition Site

The master definition site is the "home base" for a master group. It contains the base objects that are replicated to the remote sites. For our example, the master definition site is NAVDB and the remote master site is MYDB.

Once created, the master definition site will automatically replicate itself to the new master site(s). There can be only one master definition site for a replicated object. All changes to a replicated object are performed at the master definition site and propagated to all other master sites.

Within a replication environment there can be multiple master definition sites (Figure 6.4). For example, the PUBS schema could be replicated in master group REP_GROUP1 from master definition site A while the SCOTT schema is replicated in REP_GROUP2 from master definition site B. Both sites are master definition sites but on separate objects.

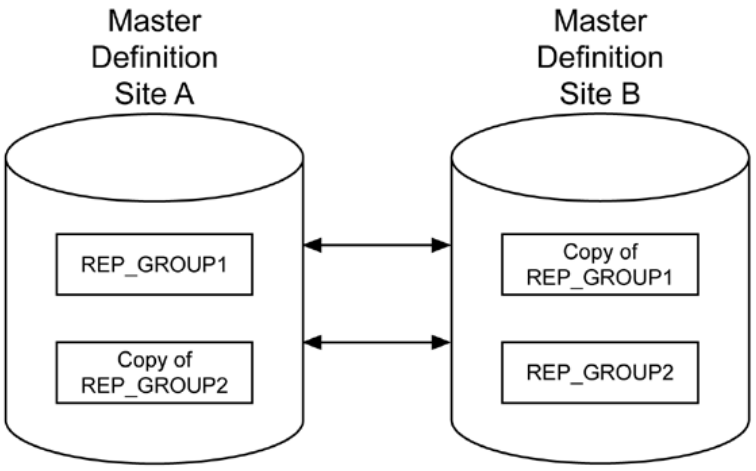

Fig 6.4 *Multiple Master Definition Sites*

Create the Master Replication Group

The master replication group contains all the objects that will be replicated. If the master group is going to also support updatable materialized views, you will need to insure that materialized view logs exist on those tables before adding them to the master group. The example will create a master replication group and add three tables, author, book_author, and book. The script MM_MasterGroup.sql in the code depot will replicate the entire schema.

MM_MasterGroup.sql

Here is an example of the creation of a master replication group called GROUP1.

```
connect repadmin/repadmin@navdb

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPGROUP(
     gname => '"GROUP1"',
     qualifier => '',
     group_comment => '');
END;
/
```

Adding Tables to the Master Replication Group

Next, we want to add our AUTHOR table from the PUBS schema. We use the *create_master_repobject* procedure of the *dbms_repact* package for this, as seen in this example:

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     gname => '"GROUP1"',
     type => 'TABLE',
     oname => '"AUTHOR"',
     sname => '"PUBS"');
END;
/
```

Now let's add the remaining two tables.

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     gname => '"GROUP1"',
     type => 'TABLE',
     oname => '"BOOK"',
     sname => '"PUBS"');
END;
/
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     gname => '"GROUP1"',
     type => 'TABLE',
     oname => '"BOOK_AUTHOR"',
     sname => '"PUBS"');
END;
/
```

****

When executing the above code, you might receive the following errors:

```
SQL> BEGIN
  2     DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  3       gname => '"GROUP1"',
  4       type => 'TABLE',
  5       oname => '"BOOK"',
  6       sname => '"PUBS"',
  7       use_existing_object=>TRUE);
  8  END;
  9  /
BEGIN
*
ERROR at line 1:
ORA-23309: object "PUBS"."BOOK" of type TABLE exists
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 105
ORA-06512: at "SYS.DBMS_REPCAT_MAS", line 2552
ORA-06512: at "SYS.DBMS_REPCAT", line 562
ORA-06512: at line 2
```

The error message indicates that you have already set up the given table for some form of replication. This is very possible if you have run previous examples from this book. We can clear this problem by using the *delete_master_repobject* procedure of the *dbms_repcat* supplied procedure, as seen in this example:

```
exec dbms_repcat.drop_master_repobject('PUBS','BOOK_AUTHOR','TABLE');
```

****

If Your Tables Don't Have a Primary Key

Suppose in the example above that the BOOK table has a defined primary key but the other two tables do not. We must tell Oracle how to uniquely identify rows in the *author* and *book_author* tables.

```
BEGIN
   DBMS_REPCAT.SET_COLUMNS(
     sname => '"PUBS"',
     oname => '"AUTHOR"',
     column_list => '"AUTHOR_KEY"');
END;
/
BEGIN
   DBMS_REPCAT.SET_COLUMNS(
     sname => '"PUBS"',
     oname => '"BOOK_AUTHOR"',
```

```
     column_list =>
      '"AUTHOR_KEY","BOOK_KEY"');
END;
/
```

Be careful with spaces inside of parameters contained within quotes. It's easy to make a mistake, for "*author*" is not the same as "*author* ".

We know that authors are uniquely identified by the *author_key* and that in *book_author,* unique rows are determined by the two columns *author_key* and *book_key*. What we have actually done is define two quasi-primary keys on those two tables. Any row that violates these keys will result in a key conflict, which must be resolved. Note this is not the same as an actual primary key. No index is created and you can insert duplicate values in the table all you want. So, it's generally much preferable to just define the primary key on the table if at all possible.

Generate Replication Support

It's time to generate replication support for the three tables with the procedure *dbms_repcat.genrate_replication_support*. This creates the objects, triggers, procedures, etc. required for Oracle to support replication. Here is an example:

```
BEGIN
   DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
     sname => '"PUBS"',
     oname => '"AUTHOR"',
     type => 'TABLE',
     min_communication => TRUE,
     generate_80_compatible => FALSE);
END;
/
BEGIN
   DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
     sname => '"PUBS"',
     oname => '"BOOK"',
     type => 'TABLE',
     min_communication => TRUE,
     generate_80_compatible => FALSE);
END;
/
BEGIN
   DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
     sname => '"PUBS"',
     oname => '"BOOK_AUTHOR"',
     type => 'TABLE',
     min_communication => TRUE,
```

```
     generate 80 compatible => FALSE);
END;
/
```

Check the view *dba_repcatlog* for errors.

```
SELECT COUNT(*)
FROM dba_repcatlog;
```

This view should be empty. If not, wait until it is empty before continuing. Once you have dealt with any errors appearing in *dba_repcatlog*, start replication support.

```
BEGIN
      DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
      gname => '"GROUP1"');
END;
/
```

At this point, the master definition site is ready to replicate GROUP1 to other master sites.

Adding Additional Master Sites

Once the master definition site has created and populated a master replication group, it is relatively easy to add a master site. Before adding a new master site you must complete the steps outlined in the planning section of this chapter. The repadmin user must be created and have the appropriate grants and be defined as the propagator. The database links must be created and functioning, along with the push/purge jobs. Once this has been accomplished, you are ready to add the new master site.

The script MM_AddMaster.sql in the code depot will add the MYDB.WORLD site to the replication scheme. It is run from the master definition site only!

Whenever you add a master site, you must suspend replication support on the master definition site.

MM_AddMaster.sql

```
connect repadmin/repadmin@navdb
```

```
BEGIN
      DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY(
      gname => '"GROUP1"');
END;
/
```

This command will stop replication support, not only on the master definition site, but also on every other master site in the replication scheme. It also places all objects in the master group (at all master sites) in read-only status to maintain data integrity.

Now add the new master site.

```
BEGIN
      DBMS_REPCAT.ADD_MASTER_DATABASE(
      gname => '"GROUP1"',
   master => 'MYDB.WORLD');
END;
/
```

This command will replicate all objects in GROUP1 to the MYDB.WORLD database.

To complete the process, restart replication support.

```
BEGIN
      DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
      gname => '"GROUP1"');
END;
/
```

Replication activity will restart on all master sites.

Note that all activity took place on the master definition site. *Dbms_recat.add_master_site* took care of creating the replicated objects in the PUBS schema on MYDB.WORLD.

Adding additional master just repeats this process:
Prep the new master site
Create repadmin user
Create database links
Create the push/purge jobs
At the master definition site

Stop replication activity
    Add the new master site
    Resume replication activity
At both sites – verify and test

Adding additional master sites is even easier in OEM.

Fig 6.5 *OEM Master Group Status Page*

Log on to the master definition site (NAVDB) as repadmin and navigate down to REP_GROUP2. Replication activity is running (normal), so you need to stop the replication activity by clicking the "Submit Stop Request" button. Select the Master Sites tab at the top of the screen and select Add. This will bring up the "Add master site to the group" window (Figure 6.6).

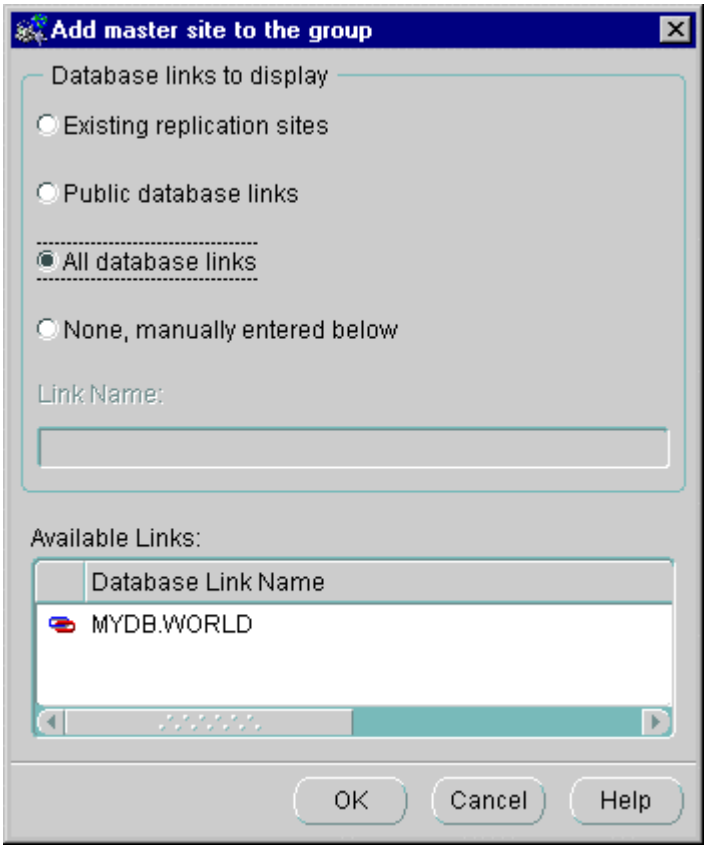

Fig 6.6 *OEM Master Group Add Master Site*

Since multi-master replication requires the use of global names, the database links to remote master sites will be names for the site. Select "All database links" to display available links. For our example, select MYDB.WORLD and click OK. The next window allows you to change the propagation methods (Figure 6.7). We will keep the defaults for our example.

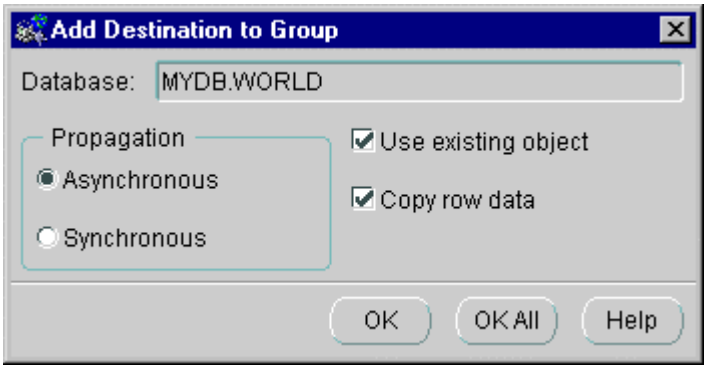

Fig 6.7 *OEM Master Group Add Destination*

If you select OK, the selection will apply only to this master site. If you select OK All, the selections will apply to all master sites.

Clicking Apply will cause the wizard to execute the propagation of the master group to all master sites. OEM displays a rather cryptic message "Applying User Changes..." as it propagates the master group objects. The last step is to restart the replication activity.

At this point, you can log onto a master site and verify that the empty PUBS schema now contains all the objects in the master group.

Now that we have multi-master replication going, we need to insure that it continues to function.

MMR and OEM

Lets quickly look at the creation of the master definition site's master replication group using OEM. We will start with the user repadmin created and the required grants executed. The push/purge jobs are in place, as are the database links.

Start Oracle Enterprise Manager (OEM), either by navigating from the Start button on Windows or by executing the Unix/Linux command:

```
oemapp console &
```

Either log on to the Management Server (if you use one) or select Stand-alone.

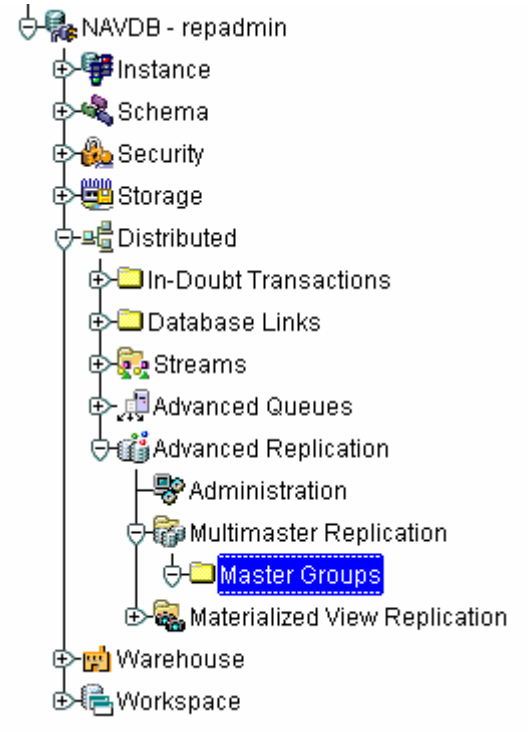

Fig 6.8 *OEM Navigation to Master Groups*

Log onto the database containing the data to be replicated as repadmin. Navigate down through the folders to Multi-Master Replication and the Master Groups (Figure 6.9). Right click on the Master Groups folder and select create.

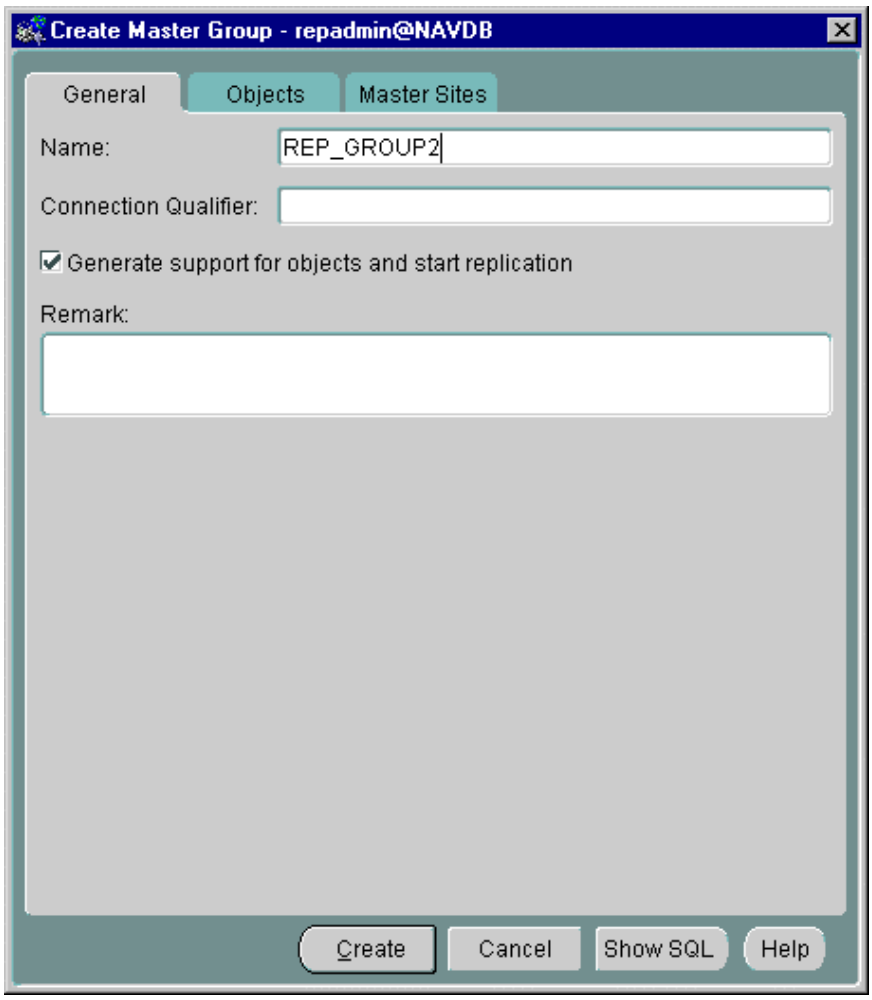

Fig 6.9 *OEM Create Master Group*

For this example, we named the master replication group REP_GROUP2. If you would like to manually generate replication support for each object, uncheck the check box. The Connection Qualifier is to identify a special type of database link, such as through a modem, which must be set up prior to creating the master group.

The next step is to add objects to the group by selecting the Objects tab at the top and then selecting the add button. This will bring up the "Add objects to group" window (Figure 6.10). You can mix object types and schema owners within a master group.

In our example, we will replicate all the objects in the PUBS schema. After selecting PUBS from the schema compo box, click on all the check boxes to

display all objects in the Available Objects box. Select each object and click the Add button.

Notice that we did not include the primary key indexes. As we discussed at the start of this chapter, each master site will automatically create primary key indexes on all replicated tables that have them defined. Clicking the OK button will cause the wizard to include the selected objects in the master group.

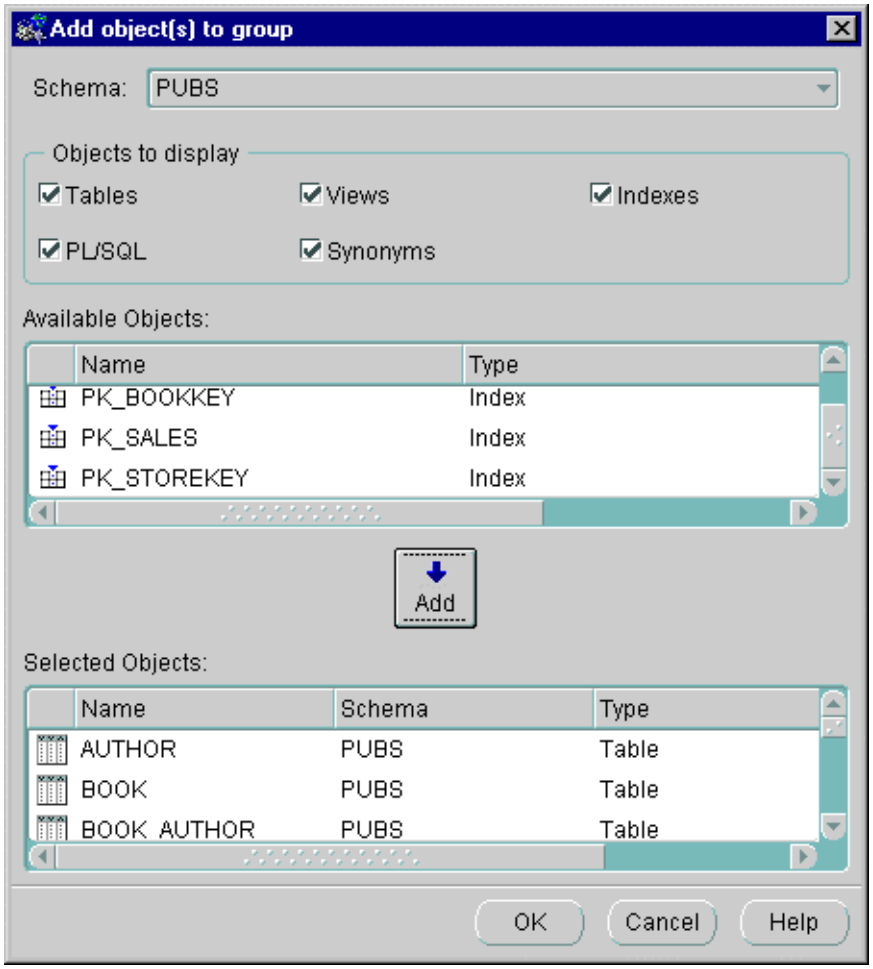

Fig 6.10 *OEM Master Group Add Objects Window*

As the wizard adds each table, it will verify that it contains a primary key. If not, the wizard will ask you to define a column or columns that will uniquely identify rows in the table (Figure 6.11). In this case, the *author* table does not have a primary key defined. The *author_key* column uniquely identifies each author (and should be a primary key) and can be used. The wizard will

require you to identify columns that uniquely identify rows for every table without a primary key.

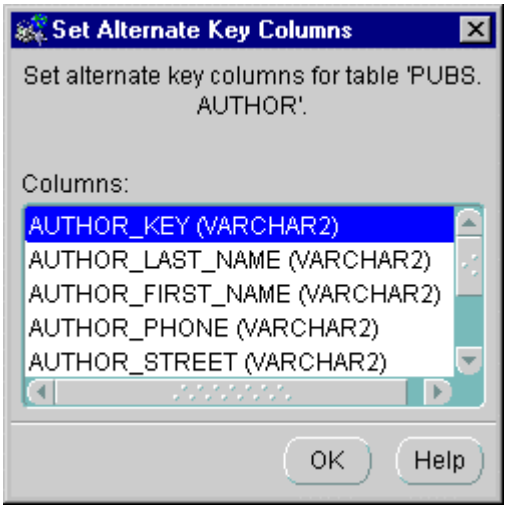

Fig 6.11 *OEM Set Alternate Key Columns*

If you have a table that cannot uniquely identify rows (duplicate rows are allowed), then you will need to use a sequence (or some other method) to create a primary key for the table. Cancel the wizard, key the table, and then restart the process. Note that as far as replication is concerned you are defining primary keys on all tables being replicated. Any violation of these keys, whether defined with primary keys or with set alternate columns, will result in a key collision that must be resolved.

Once the wizard verifies each table's key, it will list them in the Objects tab (Figure 6.12). At this point, you can add or remove additional objects as required. You can also add remote master sites to accept the replicated group.

Adding master sites is a bit too much even for us, so we don't recommend adding them at this time. Suppose master sites are added and the wizard replicates the master group to each master site. If the replication fails to function, you would have no idea what caused the problem.

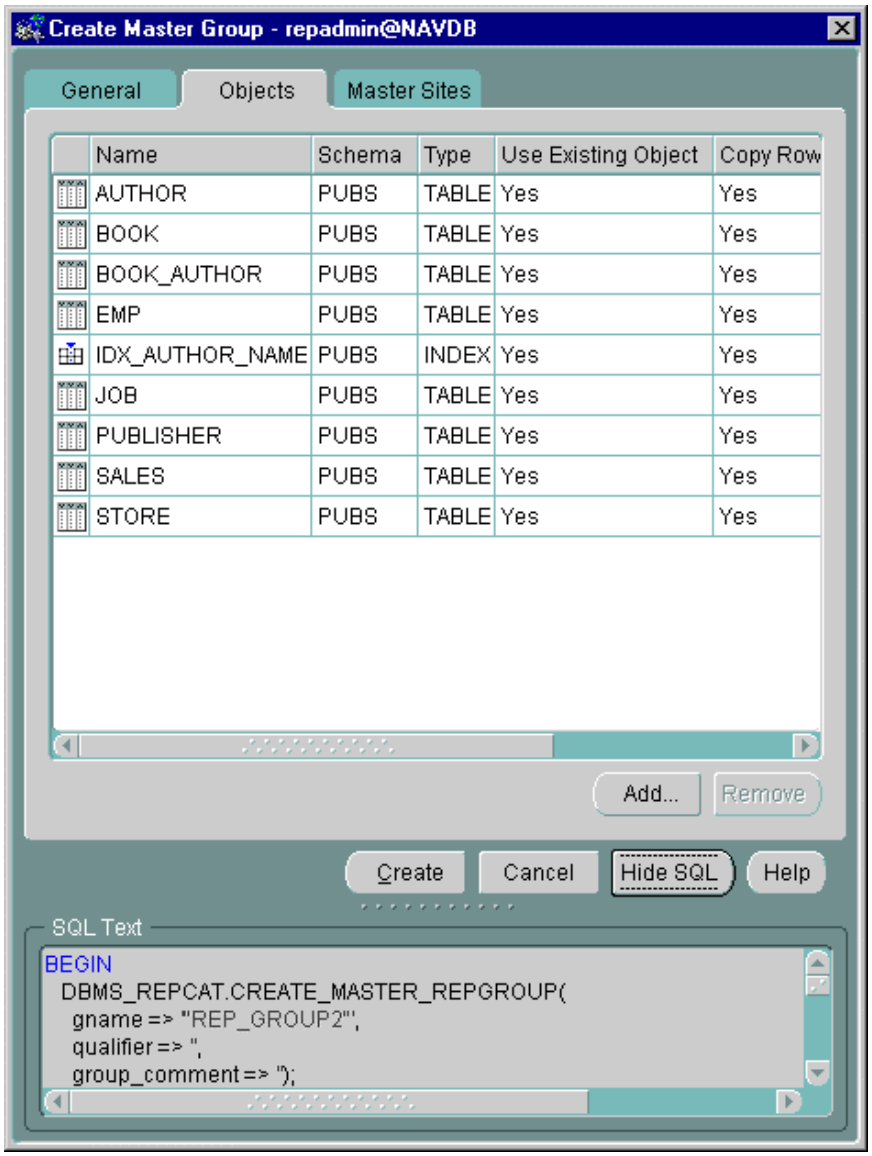

Fig 6.12 *OEM Create Master Group Objects List*

We selected the Show SQL button to see a list of the PL/SQL commands the wizard will use to create the master group. It is highly recommended that you copy and paste these commands to a text file. If the wizard fails, you will be able to execute each command individually to locate the problem. Select Create to execute the commands and create the master group. OEM will indicate that it has successfully created the master group when it has finished.

Now that we have the master group REP_GROUP2 created, and replication activities are running, it is time to propagate that group to another master site.

Maintaining Multi-Master Replication

Replication is now set up and running. As proud as you may be of this achievement, you are still not out of the woods. That's because you have to keep it running. On top of that, you have to maintain the system to include adding and removing objects from master groups and loading and removing data. There is also the issue of resynchronizing data after fixing problems.

Monitoring Replication

Chapter 5, Monitoring Updatable Materialized Views, discussed methods to monitor the replication jobs and the deferred error queue. Monitoring multi-master replication is monitored in the same way (via *dba_jobs*), with the exception that each master site must be monitored.

Each master site has three jobs that support replication: push, purge, and *do_deferred_repcat_admin jobs*. Each job must continue to run for replication to function.

Deferred Error Queue

When a receiver gets a data transaction from the deferred transaction queue but can not apply it, the transaction is placed in the deferred error queue. To insure data integrity, once there is a transaction in the error queue, all following transactions end up in the error queue and replication basically stops on that master site. To determine if there are errors in the deferred error queue use the following sql statement:

```
select count(*) from deferror;
```

Any returned number greater than zero indicates that there are errors in the queue. The transactions in the error queue must be applied or deleted from the queue for replication to function. Chapter 5 details the scripts to monitor and correct failed replication jobs and errors in the error queue.

Key Conflicts

There can be a number of reasons that transactions end up in the error queue. One of the most common is due to key conflicts. This happens when a table with a defined primary key and a row from another master site conflicts with a

row already in the table. Oracle will not allow the conflicting row to be inserted into the table.

In a normal environment, the transaction would simply rollback. In a replication environment, the row has already been applied at the originating master site, so a roll back is not possible. This transaction will end up in the error queue unless conflict resolution is defined.

Key conflicts do not just happen on primary keys. Remember when you added that table without a primary key and used *set columns* to identify columns that uniquely identified rows? The replication support defined a key on those columns, and any transaction that violated the uniqueness of those columns was rejected. That is why we said that using *set columns* would, for the purpose of replication, define a primary key on those columns.

Conflict resolution will be further discussed in Chapter 7.

Adding and Removing Objects

Adding and removing objects is easy, provided they are not large. Simply stop replication activity. Add the object to the master group at the master definition site and restart replication. The new object will be propagated to all master sites in the replication environment.

```
connect repadmin/repadmin@navdb

BEGIN
      DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY(
      gname => '"GROUP1"');
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     gname => '"GROUP1"',
     type => 'TABLE',
     oname => '"CUSTOMER"',
     sname => '"PUBS"',
     copy_rows => TRUE,
     use_existing_object => TRUE);
END;
/

BEGIN
      DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
      gname => '"GROUP1"');
```

```
END;
/
```

In the code above, we added the *customer* table to the master replication group. When we resume replication activity, the *customer* table will be replicated to all other master sites. But what if the table contains 5 million rows of data? It may take days to propagate the table to the remote sites. In this case, you will want to use one of the methods discussed below.

Moving Large Data Sets

Replication propagates data one change at a time. If you insert 300 rows into a replicated table, those 300 rows are propagated and applied one row at a time to insure data integrity.

Oracle supports parallel propagation across the database links, but the rows must be inserted at each master site in order. This is not a problem for even relatively high transaction rates on OLTP systems, however, if you load a million rows of data, it could take hours or days to propagate. As if that were not bad enough, the table that you load the data into will be read-only until the propagation is completed.

Another example is in the replication of the PUBS schema. If that schema contained millions of rows of data, the replication creation could take days. There must be a better way to load data into a replicated table. There is, and it is called offline instantiation.

Offline Instantiation

Offline instantiation allows you to create the master group object on the remote sites and then place replication on top of those objects. While this method can greatly speed up the creation or loading of replication objects, it is not without its own issues. The base objects must be read-only from the time the data begins moving until replication can assume control. If the base tables are updated, then the copy will not be in sync.

The mechanics of offline instantiation are simple, the execution is not. Basically, you create your master group with false "copy_rows". This tells the master group to use the rows already at the remote site. No data checks are performed. If the tables are not in sync, then they will begin replication out of sync and stay that way.

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     gname => '"GROUP1"',
     type => 'TABLE',
     oname => '"BOOK"',
     sname => '"PUBS"',
     copy_rows => FALSE,
     use_existing_object => TRUE);
END;
/
```

Now comes the hard part, how do you get the data to the remote site? The difficulty is that the base tables must remain in read-only state until transfer is completed. Any changes and the data at the two sites will be out of synch.

Export, Import Method

The tried and true method is to export the data at the base site, move it across the network, and import it at the remote site. This method was about the only thing available before Oracle9i. By piping the export to a compression program, the dump file can be reduced to speed the movement across the network. The most time consuming part is, of course, the import. Oracle9i introduced a new method called Transportable Tablespaces.

Transportable Tablespaces

Transportable tablespaces allow the DBA to export only the tablespace metadata from the data dictionary and then move the data in datafiles to the remote site. Once the metadata is loaded into the remote site's data dictionary, the tablespace can be used. Since there is no import of data (except a small amount of metadata), transportable tablespaces can greatly speed the process of propagating the initial data to remote master sites.

Loading or Resynchronizing Data

So far, we have discussed moving large amounts of data before creating the replication environment. But what about after replication is created and running? You can always stop replication activity on the master group, but that places all objects in the group in read-only status. To stop replication and not place the objects in read-only status, you must use the *dbms_reputil* package.

```
execute DBMS_REPUTIL.REPLICATION_OFF;
```

This disables all replication triggers so that you can modify data without replicating the changes. You can now load or remove data without propagating the changes. When finished, reactivate the triggers.

```
execute DBMS_REPUTIL.REPLICATION_ON;
```

Once the replication triggers are active, further changes will be propagated. As always, there is a catch. It is up to the DBA to insure that no user changes are made to any site while the replication triggers are off. Also, to insure data integrity, all sites should load the data simultaneously to insure against conflicts. Data conflicts are discussed in the next chapter.

Monitoring with Oracle Enterprise Manager

OEM has become a powerful tool, and where it really shines is in monitoring replication. If you sign on as repadmin and navigate down to the Advanced Replication Administration folder, OEM displays a graphic of the replication scheme, identifying sites with errors and sites currently propagating transactions (Figure 6.13). You will quickly learn the different icons, but until you do, click on the Legend button for a listing.

In Figure 6.13 you can see that the database link is active and that no master site currently has transactions in the error queue. You can also see that the MYDB.WORLD master site supports 2 tables, replicating to another site as materialized views.

This one quick view gives you a detailed report of your replication status. The only detail lacking is the state of the replication jobs at each site. If a job is broken, transactions will build up in the deferred transaction queue, but they will not generate errors. For that reason, you need to implement a script like the ones detailed in Chapter 5 to monitor jobs at each master site.

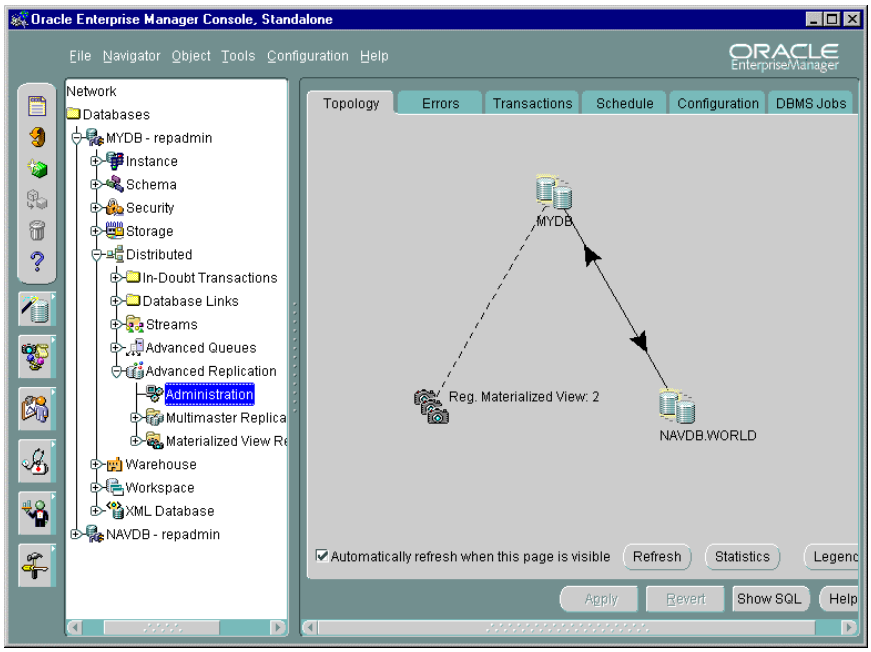

Fig 6.13 *OEM Replication Administration*

Conclusion

We have discussed the basics of setting up and administrating a simple multimaster replication scheme, as well as some additional items that need to be considered when planning a multimaster scheme, such as the trusted and untrusted security models. Some of the main points included:

- Plan your implementation in detail. Anticipate how you are going to move the possible large amounts of data to remote master sites.
- Build the replication and then test it thoroughly. DOCUMENT EVERYTHING!!!
- Develop a system to constantly monitor the replication status.

That about covers the basics of multi-master replication. Your master sites should be happily replicating transactions back and forth. But you are still not quite finished. No matter how simple your replication scheme is, if it involves advanced replication, you should implement methods to resolve key conflicts. And that is what we will cover in the next chapter.

Chapter 7 will discuss the causes of key conflicts and the available methods to automatically resolve them.