Advanced Database
8trd157

Lab8
(no report)
Tutorial on Asynchronous Database Replication

Paul Girard, Ph.D.

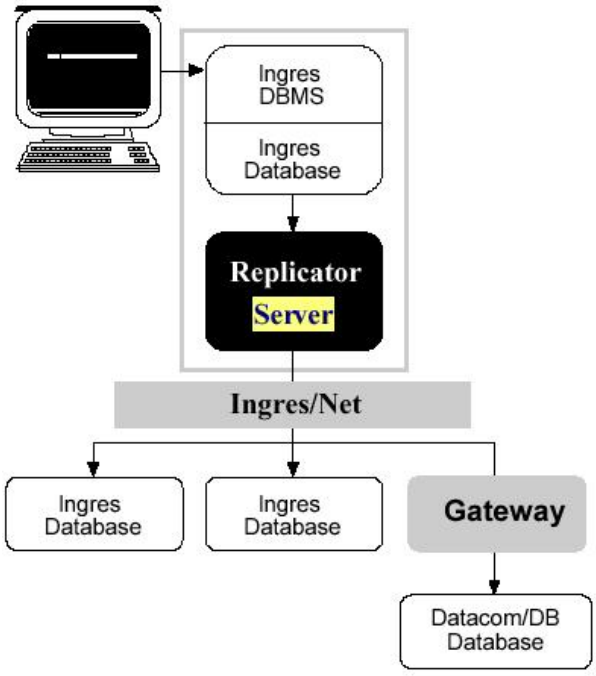
Replication is the act of maintaining identical copies of a defined set of data *(tables, columns, rows)* in more than one database. Replication allows continuous access to your databases, protects against catastrophic failures, improves performance, and can maintain data across different database systems.

1 Ingres/Replicator

- **Ingres/Replicator** is a database connectivity system that manages replication. You can replicate between different databases on the same machine or on different machines on the other side of the world, as long as they are in a distributed network *(Ingres/Net)*. So, instead of having all your users throughout the world access the same database in one location, you can have a replicated database in every region.

- *Ingres/Replicator* works in the background to replicate data asynchronously by transaction. Data integrity is enforced through a **two-phase commit protocol**. Updating is done to the local database **asynchronously,** or **as fast as possible**, so it is transparent to local users. *Ingres/Replicator* queues each replicated transaction and uses two-phase commit.

- *Ingres/Replicator* also allows you to monitor your replicated transactions and run reports on your replication system.



2 Advantages of Ingres/Replicator

- **Improved performance**
  1) When multiple sites need access to the same data, providing replicated copies of the data at each local site reduces network traffic and improves response times. 2) Also, by distributing the users over more than one machine, the load on any single machine is reduced, overcoming system bottlenecks.

- **Fault tolerance**
  If a node in the network becomes unavailable, replicated changes to be sent to that node are queued in order and are

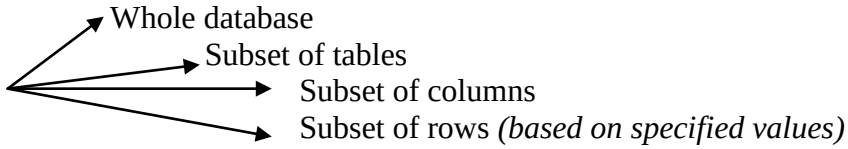
sent when the node comes back online. Since your data exists in more than one location, loss of data on one machine is not catastrophic.

- **Flexibility**
  Local databases, heterogeneous databases, local and remote DBMS servers can be added or removed at any time. It can also support non-Ingres databases through the gateway *Ingres/Enterprise*.

3 Replication Options

The configuration options fit many specific needs.

- **Data to be replicated**



- **Replication strategies**
  - One direction with a single target *(master to slave)*
  - One direction with multiple targets *(master to slaves)*
  - Both directions between two databases *(master to master)*
  - Many directions between many databases
  - Combination of any of these options

- **Replication Routing**
  Each replication route for the replicated data must be specified to get to all its targets.

- **Replication Schedule**
  The user may specify specify if the replication must be done continuously, once a day, or on demand.

4 Replicator Terminology

This section defines some terms associated with key replication concepts.

4.1 A **Consistent Distributed Data Set** *(CDDS)* defines what data is replicated and where it resides. The CDDS is the unit of organization in *Ingres/Replicator*. A CDDS can consist of

- the **whole database** OR
- a **subset of tables** *(for example, branch offices may not need all the information that the head office has)* OR
- a **subset of columns** *(vertical partitioning).* Only selected columns in selected tables are replicated. OR
- a **subset of rows** based on values *(horizontal partitioning)* where only certain rows from selected tables are replicated based on the values of one or more columns.

4.2 A **CDDS Target Type** defines within a CDDS which database is allowed to alter the data and how *Ingres/Replicator* acts on each database within the CDDS. A CDDS has a target type defined in every database that is contained in the CDDS. The target types are:

➢ **Full peer**	generates changes to be replicated, accepts replicated changes and can detect collisions. A *full peer* designation is required for production CDDSs where interactive users or other processes first manipulate data. This target consumes more disk space and CPU cycles than any other target type.
➢ **Protected read-only**	accepts replicated changes and can detect collisions
➢ **Unprotected read-only**	accepts replicated changes but cannot detect collisions

4.3 A **data propagation path** *(dd_path)* defines what route the data takes to reach target databases within a CDDS. Replicator moves information from one database to another according to the data propagation paths for that CDDS. In a data propagation path, a database can be one of three types: **originator, local,** or **target.**

➢ **Originator**	The database where a user made a change to a replicated table.
➢ **Local**	The database that propagates the change to the target. This is the last database that the change goes through before reaching the target. The local database can be the same as the originator database.
➢ **Target**	The database that receives the change. Every database in the CDDS must be a target *(except the originator in that particular path).* The target and the originator database cannot be the same in a given path.

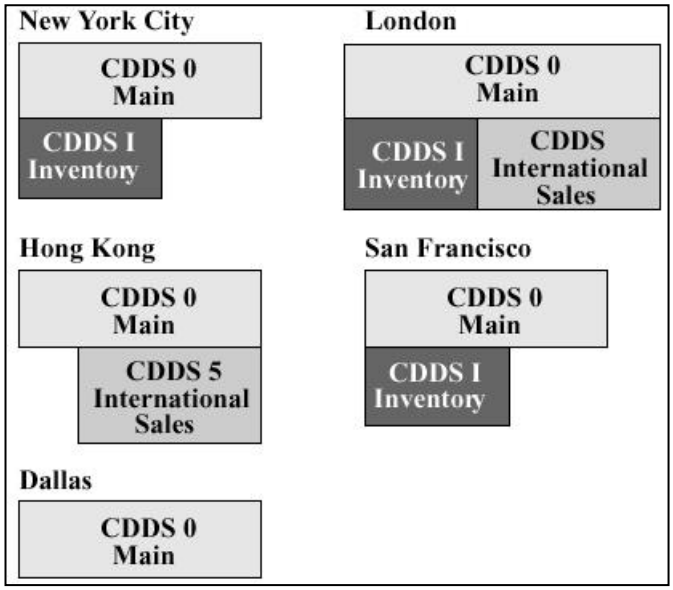
Example

Data Propagation Paths for CDDS 1 in San Francisco which is set up to transmit inventory information to New York and London using New York as a relay.

Origin	Local	Target
SFO	SFO	NYC
SFO	NYC	LON

Data Propagation Paths for CDDS 1 in New York

Origine	Local	Cible
NYC	NYC	SFO
NYC	NYC	LON



4.4 A Replication key is used to identify information specific to Ingres/Replicator during processing of a replicated row, the system generates a key for each such row. The key consists of:

- A database number that identifies the replicated database that manipulated the row
- The ID of the DBMS server transaction that manipulated the row
- The sequence number within the transaction

4.5 A **collision** occurs when more than one user updates *(insert, update, or delete)* transaction the same replicated row in different databases. *Ingres/Replicator* can preserve the history of changes made to each replicated row. If the same rows in two databases do not match in the last transaction, *Ingres/Replicator* detects a collision. There are five possible situations that cause collision between a source and target database:

- Insert transaction detects a duplicate key
- Update transaction is attempted on a record that no longer exists
- Update transaction is attempted on a record that does not match the original
- Delete transaction is attempted on a record that no longer exists
- Delete transaction is attempted on a record that does not match the original

The replication system must be designed to reduce the probability of collisions but even in well-designed databases, collisions can occur *(ex. system failure when it becomes necessary to switch between replicated databases).* For this reason alone, a plan is needed to handle collisions in a replication system. All collisions are counted as errors.

Collision handling happens at the CDDS level. When defining a CDDS, the **collision mode** must be specified. There are two ways to handle collisions, *automatically* or *manually.* Each method has advantages and disadvantages.

> **Automatic resolution** takes less time but may produce unexpected results. With *automatic resolution*, when two records collide, one record prevails over the other. *Automatic resolution* should not be used if the information contained is important.
>
> **Manual resolution** gives more control and may be the only way to resolve a conflict. With *manual resolution*, collisions can be resolved manually by editing the base table, the shadow table, and the distribution queue.

Collision Modes

- **Passive Detection** This mode detects insert collisions, update and delete collisions where the row to be updated or deleted does not exist. This is the default setting. The Replicator server does not resolve the collision. This mode requires a manual collision resolution.

- **Active Detection** This mode detects all collisions, but does not resolve them. Before propagating the row, the Replicator server searches the target database to see if the row already exists. If it detects a collision, however, the Replicator server does not resolve it. The collision is an error and the Replicator server action is dependent on the error mode setting. This mode requires a manual collision resolution.

- **Benign Resolution** This mode detects and resolves benign collisions *(when the same row with the same transaction ID arrives at the same target more than once).* Before propagating the row, the Replicator server searches the target database to see if the row already exists. If the row does exist and it came from the same transaction, the Replicator server issues a warning message and removes the operation from the distribution queue. Otherwise, the server action is dependent on the error mode setting. This mode requires a manual collision resolution on all but benign collisions.

- **Priority Resolution** This mode detects and resolves all collisions according to assigned priorities. Before propagating the row, the Replicator server searches the target database to see if the row already exists. If the Replicator server detects a collision, it resolves it by comparing the priority numbers assigned to the underlying rows. The row with the highest priority number prevails. If the priorities are the same or do not exist, then the row with the lower database number survives the collision. Priority numbers are assigned in a priority collision resolution lookup table. In this mode, all collisions are resolved automatically.

- **Last Write Wins** This mode detects and resolves all collisions according to transaction timestamps. Before propagating the row, the Replicator server searches the target database to see if the row already exists. If the Replicator server detects a collision, it resolves it by comparing the transaction timestamps. The row with the later timestamp prevails. If the timestamps are *identical,* the row with the lower database number survives the collision. In this mode, all collisions are resolved automatically.

For all propagation error modes, when a server detects an error it logs the error and issues e-mail messages to any users on the mail notification list. Server behavior depends on each of the 5 propagation error modes.

Error Handling

1) Skip Transaction *(default setting)* : The Replicator server:

- Rolls back the current replication transaction
- Processes the next transaction
- Retries the transaction in error during the next processing of the distribution queue

2) Skip Row : The Replicator server continues from the error with no rollback performed. The record in error is skipped and processing proceeds to the next record.

3) Quiet CDDS : The Replicator server rolls back the current replication transaction. After the rollback of the transaction in error, the Replicator server quiets the CDDS on the database where the transaction error occurred. The Replicator server continues processing the remaining replicated transactions for the same CDDS on other databases, and for other CDDSs.

4) Quiet Database : The Replicator server rolls back the current replication transaction. After the rollback of the transaction in error, the Replicator server quiets all CDDSs in the database where the transaction error occurred. The Replicator server continues processing the remaining replicated transactions for other databases.

5) Quiet Server : The Replicator server:

- Rolls back the current replication transaction
- Stops processing of the remaining batch of replicated transactions
- Changes its status from active to quiet mode

5 Components of Ingres/Replicator

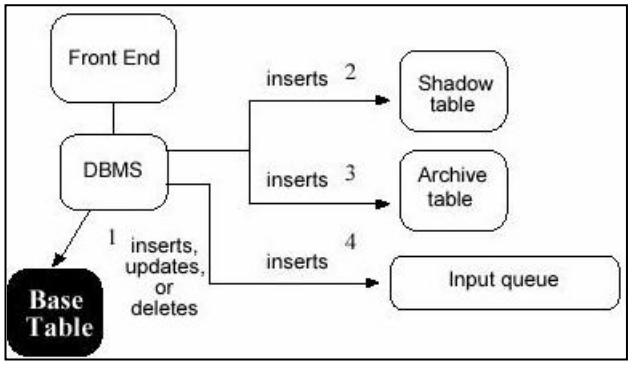
Ingres/Replicator has 3 components :

- the **change recorde**r,
- the **distribution thread** and
- the **replicator serve**r.

5.1 The **Change Recorder** is responsible for recording all changes made to base tables registered for replication. A queue of transactions that need to be distributed by the *distribution threads* is known as the **input queue**. The data contained in the support tables is also used for collision resolution. When the *Change Recorder* function is activated the following process begins:

1) A replicated transaction key for that row is generated internally.

2. *The Change Recorder* inserts a new row into the **shadow table** with the replicated transaction key and the key columns of the base table.



3. In *full peer* situations, if the change is an update or delete, the *Change Recorder* inserts the image of the row associated into a new row in the **archive table**. In this way the *shadow* and *archive* tables provide the sequential history of every replicated row.

4. The *Change Recorder* inserts the replicated transaction key into the *input queue* which now contains the information necessary to complete the replication, except the destinations.

5.2 The **Distribution Threads** put the destination address on the replication that needs to be propagated. The *distribution threads* take the information from the input queue and expand it to fully specify what data needs to be replicated to which targets. The expanded information is stored in the **distribution queue**. The distribution threads perform the following tasks:

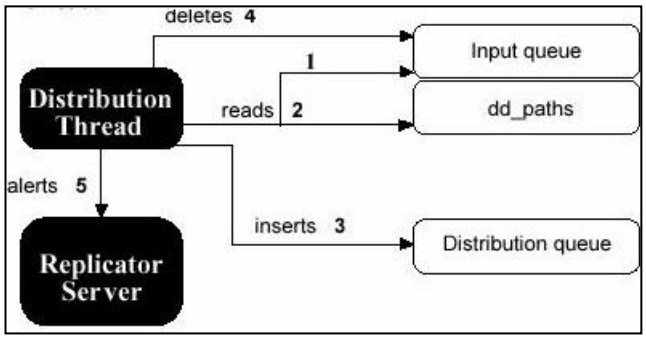
1. A *distribution thread* reads a record from the **input queue** to obtain the replication information.

2. A *distribution thread* obtains destination information from the propagation paths table called **dd_paths.**

3. A *distribution thread* inserts a row into the **distribution queue** for every destination of the replication.

4. The *distribution thread* deletes the replication's rows from the input queue.

5. The *distribution thread* alerts the *Replicator server* associated with this replication

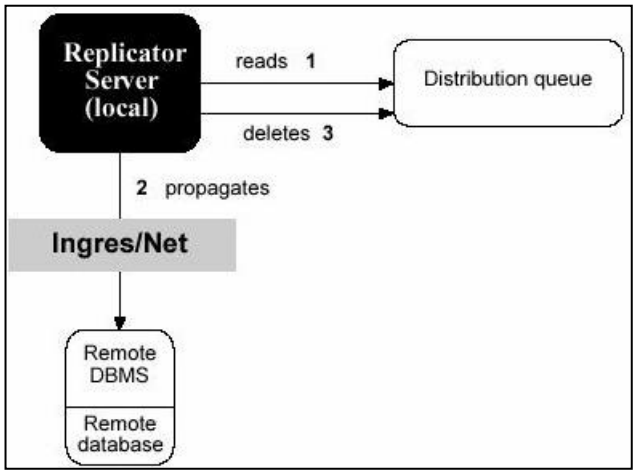


5.3 The **Replicator Server** is a stand-alone process that sends changes prepared by the *distribution threads* to their respective targets. Once the transaction has been transmitted, the replications are secured on the target database and removed from the *distribution queue*. The *replicator server* is also responsible for collision detection and resolution. After the *distribution threads* alert it, the Replicator server completes the following tasks:

1. The *Replicator server* reads the distribution queue.

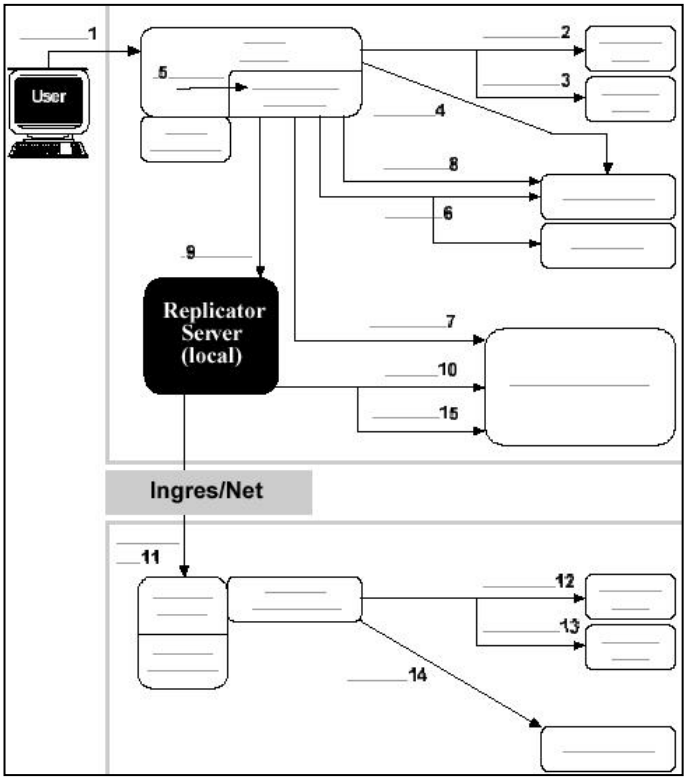
2. The *Replicator server* propagates each replicated row to the target database using direct SQL or database procedures over *Ingres/Net.*

3. After propagating a set of rows in an original user transaction by CDDS *(Consistent Distributed Data Set)*, the server deletes the corresponding rows from the *distribution queue*. The *Replicator server* uses **two-phase commit** to update two databases.



Summary of Replication

The following sequence includes all the tasks described in the *Change Recorder, Distribution Threads,* and *Replicator Server* sections.



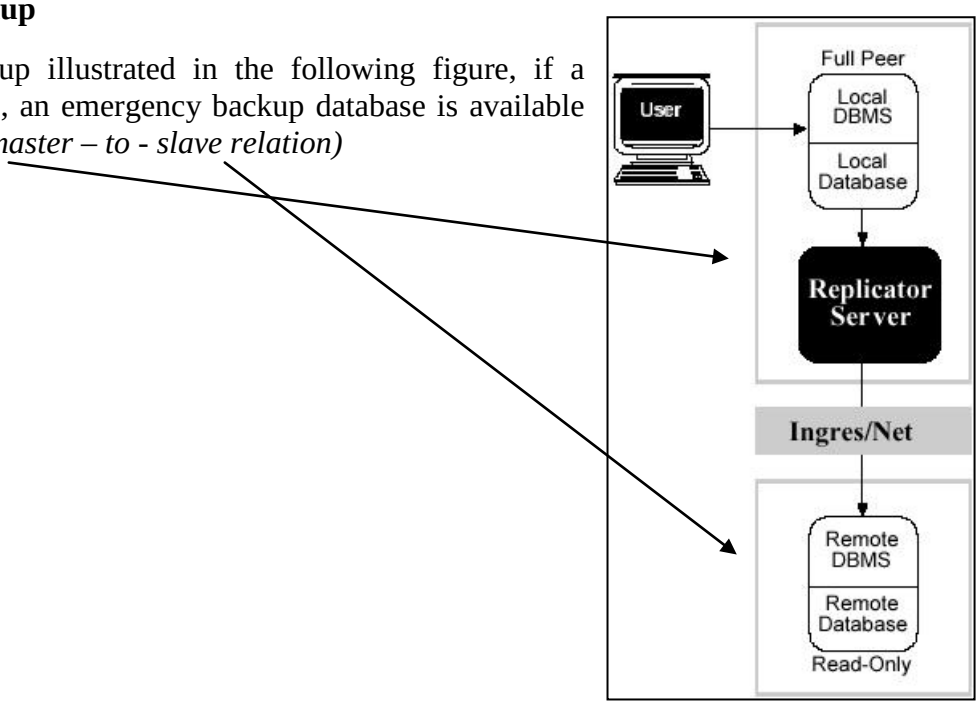
1. A user updates the local database through the local DBMS server.
2. The *Change Recorder* updates the *shadow table*.
3. The *Change Recorder* updates the *archive tabl*e.
4. The *Change Recorder* adds a row to the *input queue*.
5. After a commit, the *distribution threads* read the *input queue* and the *propagation paths table* to determine whether and to where the replication needs distributing.
6. The *distribution threads* update the *distribution queue* with the replication and its destination information.
7. The *distribution threads* delete the replication from the *input queue*.
8. The *distribution threads* alert an active *Replicator server*.
9. *Replicator server* reads the *distribution queue.*
10. *Replicator server* updates the remote database using remote SQL or database procedures.
11. If the remote database is a full peer or protected read-only target, the *Replicator server* updates the corresponding *shadow table*.
12. If the remote database is a full peer target, the *Replicator server* updates the corresponding *archive table*.
13. If the remote database is a full peer target, a row is added to the *remote input queue*.
14. The appropriate row is deleted from the local database's *distribution queue*. The *Replicator server* deletes the corresponding rows from the *local distribution queue*.
15. The changes at the remote and local databases are secured using *two-phase commit*.

6 Ingres/Replicator Design on Network

There are a number of possible replication schemes that you can use to propagate transaction data throughout your distributed processing environment. The way you choose to design *Ingres/Replicator* on your network will depend on the needs of your business and the purpose of your application.

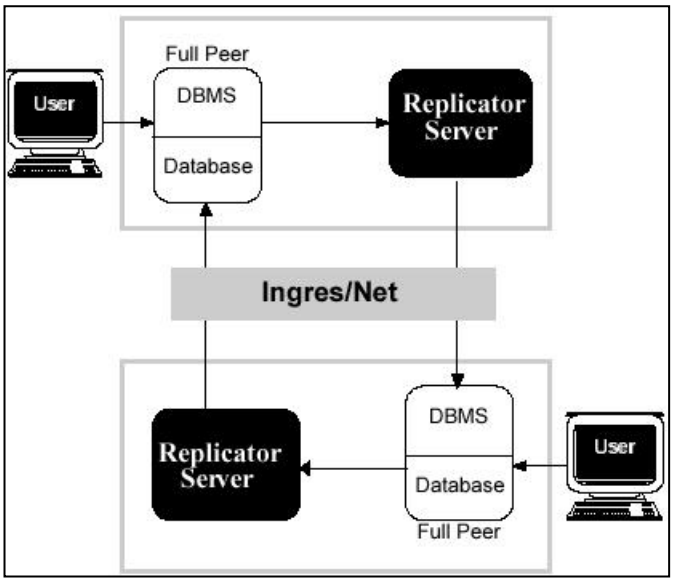
6.1 Central-to-Backup

In a central-to-backup illustrated in the following figure, if a source database fails, an emergency backup database is available to resume activity. *(master – to - slave relation)*



6.2 Peer to Peer

In **peer-to-peer** replication, illustrated in the following figure, each replicator site plays a sending and receiving role on a peer basis with other target databases. There is no hierarchical relationship between the various servers. *(master – master relation).*

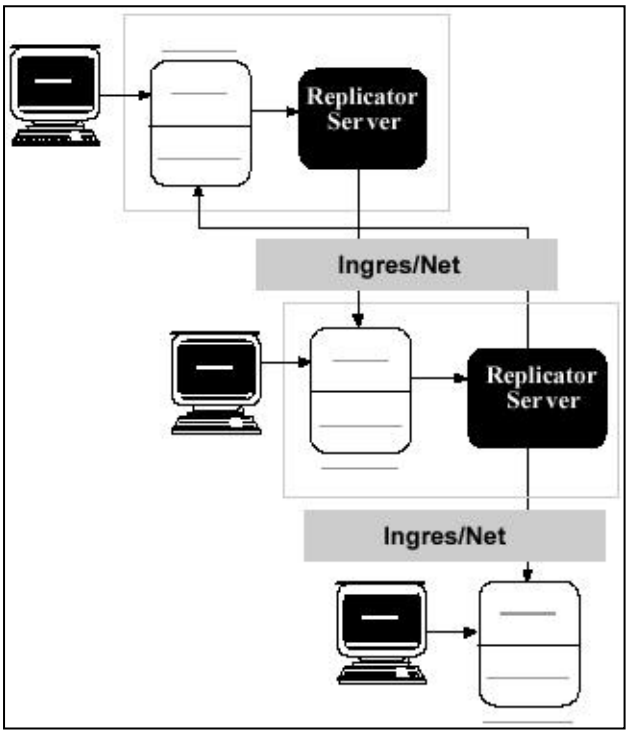


The peer-to-peer arrangement implies that each replicator site is functioning autonomously, using its own replicated database. All sites have the same information by having their own copies of an enterprise database.

A peer-to-peer scheme may have more than two, with each database transmitting directly to each of the other databases.

6.3 Type cascade

In a **cascade** replication, illustrated in the following figure, data from a source database is moved to a target database. That target database in turn moves the data to yet another target database. Every database that propagates the replication to another database must be designated as full peer.



This scheme is useful in that the source database does not have to be responsible for distributing the data it creates throughout the entire replication system.

(master → master → ...)

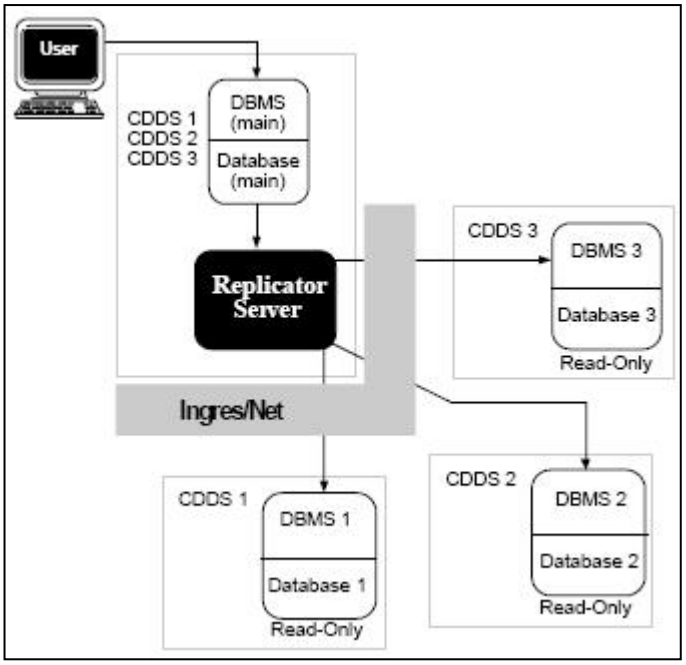
6.4 Central-to-Branch

In a **central-to-branc**h scheme, illustrated in the following figure, subsets of a central database are designated for replication to branch sites.

The diagram shows read-only targets, but central-to-branch is often combined with peer-to-peer. In this arrangement, branches can then update information from headquarters or create their own data and send the new data back to headquarters.

(master – slave) or

(master – master)

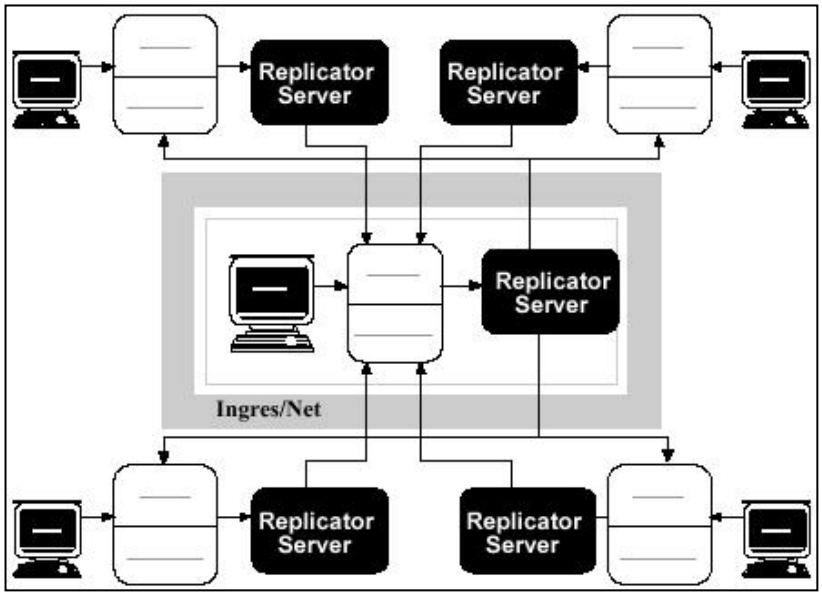


6.5 Hub-and-Spoke

In a **hub-and-spoke** scheme, illustrated in the following figure, the hub database has a peer-to-peer relationship with each of its spokes.

Implicit in this relationship is a cascade replication, in which each of the spokes receives replicated data whenever the hub database or any of the other spoke databases are manipulated.

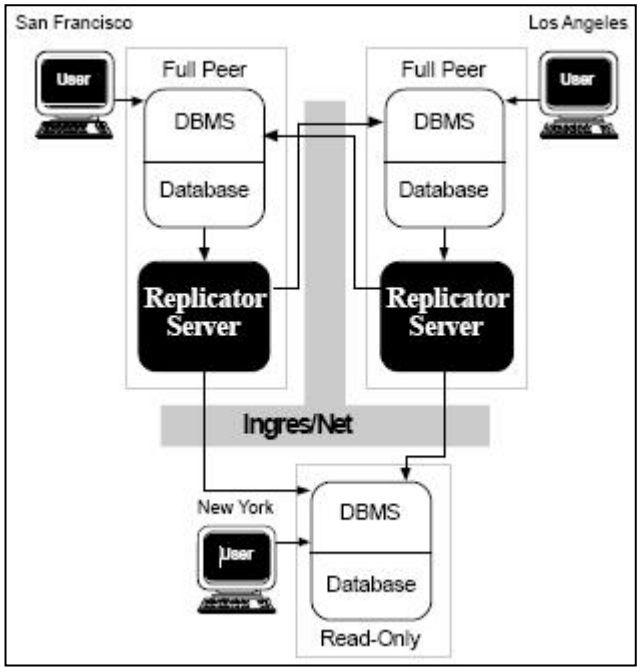
(master – master – master)



6.6 Combination Design Schemes

Because no one technique may accommodate the needs of environment, more than one design technique may be used.

For example, a site may need to use data for decision support applications where no updates to data are required. The following figure illustrates tolerant peer-to-peer and cascade scenario. Data from San Francisco to Los Angeles on a peer-to-peer basis and to New York on a protected cascade basis. Data from Los Angeles is replicated to San Francisco on peer basis and to New York on a protected read-only cascade basis. In the event of a failure at New York, decision support is performed Francisco or Los Angeles with a degree of performance impact.



7 Methodology

This example will start a central-to-backup database replication between the master database **bd1a** on sunensa.uqac.ca and the slave database **bd1b** on sunensb.uqac.ca.

7.1 Creating Ingres/Replicator Catalogs

The **repcat** utility creates and populates the *Ingres/Replicator* catalogs and creates Ingres/Replicator database events.

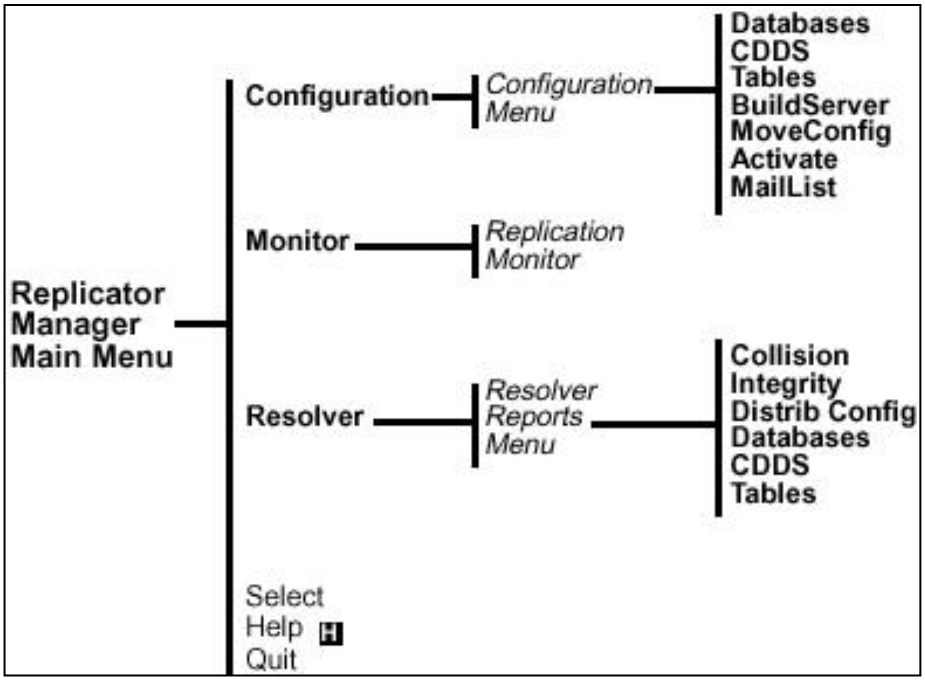
```
sunensa:pgirard> repcat bd1a
Creating Replicator catalogs on database 'bd1a' ...
Replicator catalogs for database 'bd1a' created successfully.
```

```
sunensb:pgirard> repcat bd1b
Creating Replicator catalogs on database 'bd1b' ...
Replicator catalogs for database 'bd1b' created successfully.
```

7.2 Replicator Manager

The first time, the **Replicator Manager** must be started from a database that has at least one full peer CDDS. Ingres/Replicator must be configured on such a database and then move the configuration to all databases that participate in the replication scheme.

Main Menu of *repmgr*

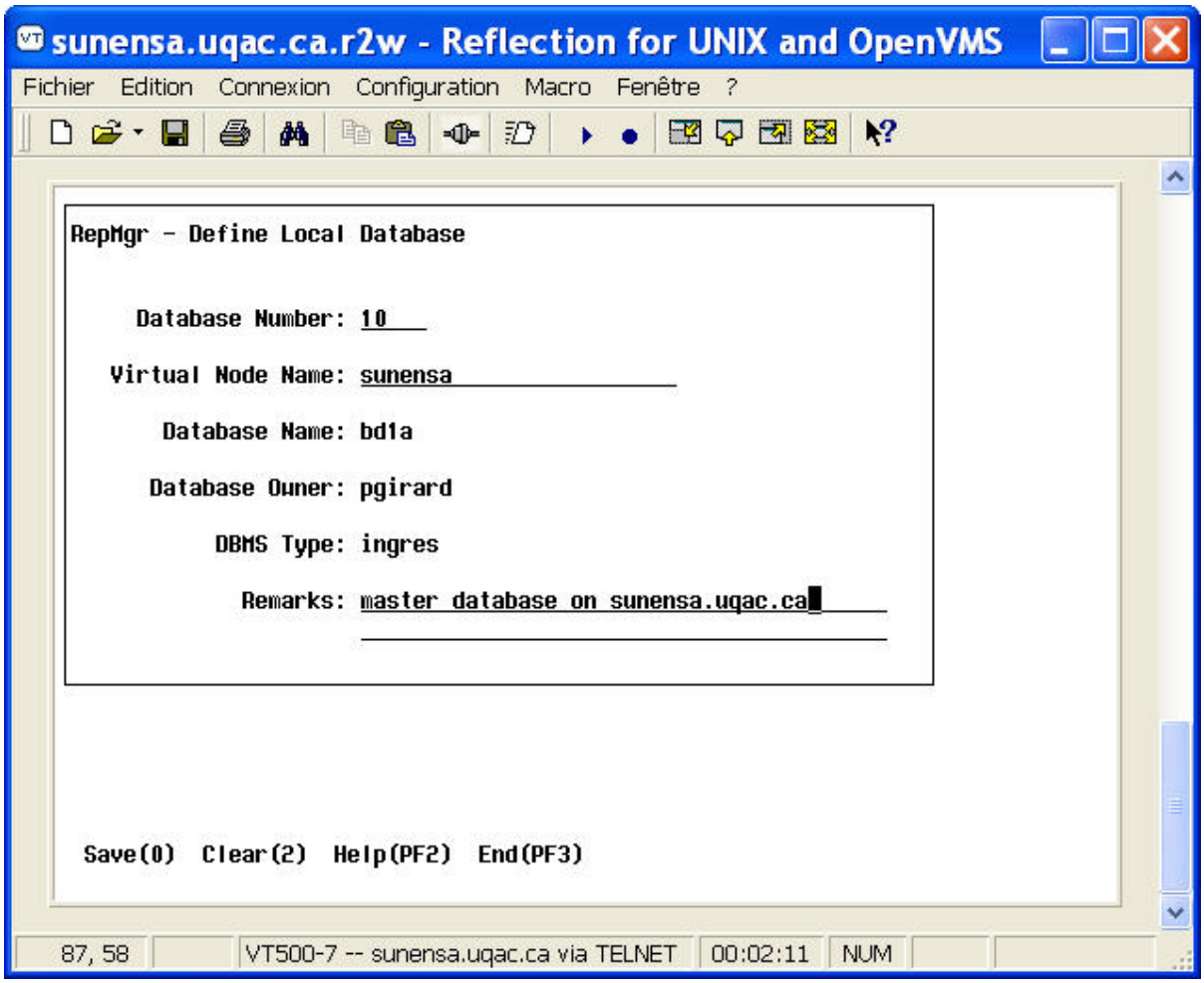


7.1 Start the replicator manager on *bd1a*

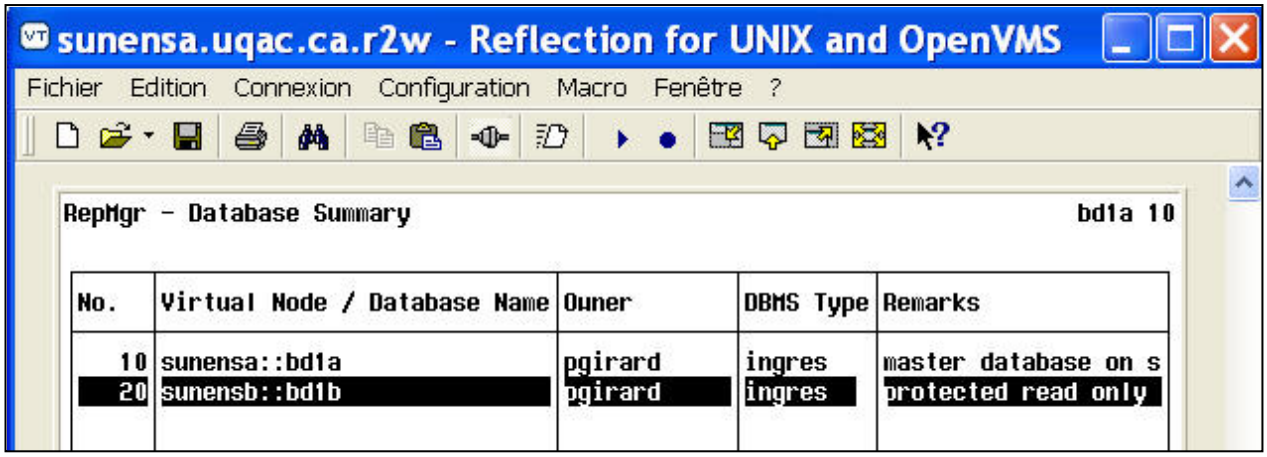
```
sunensa:pgirard> repmgr bd1a
```

7.2 Define all databases sending *(bd1a)* or receiving *(bd1b)* data from the replicator. The following figure illustrates the form to be filled to define a database.

Our example defines the database *bd1a* on the virtual node *sunensa*, 10 is the identifier given to this database.

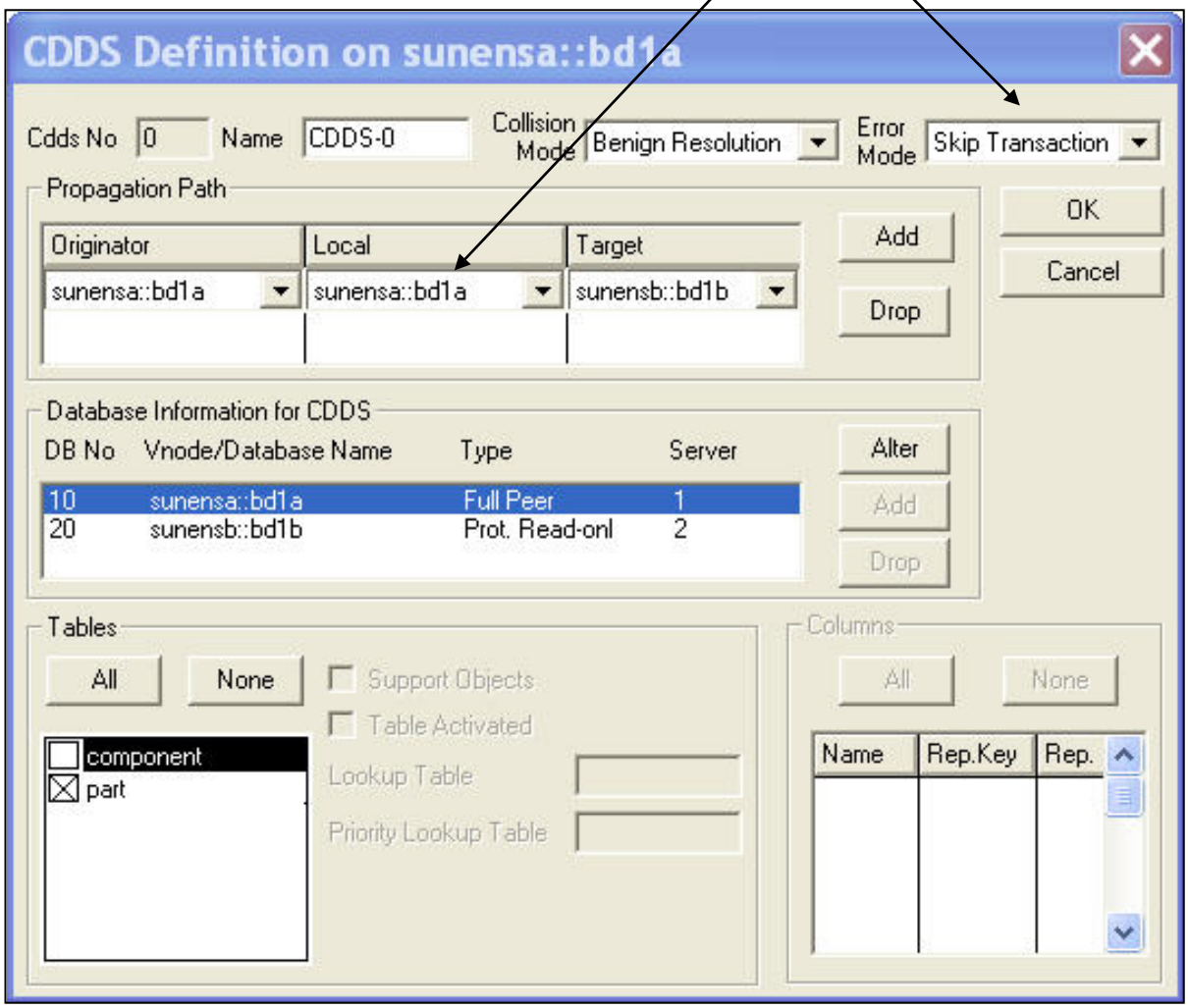


the same is done on *bd1b* identified by 20 on *sunensb*. The result is illustrated in this figure



7.3 Definition of a CDDS

The same work may be done using **Visual DBA**. This example defines the CDDS-0 made of the table *part* . The *data propagation path* is defined by the origin and local database is *bd1a* on *sunensa* and the target is *bd1b* on *sunensb*. The collision mode *(not possible in this design)* benign resolution is chosen with an error mode of skip the transaction. So any change to the table *part* in *bd1a* will be replicated on the same table in *bd1b*.

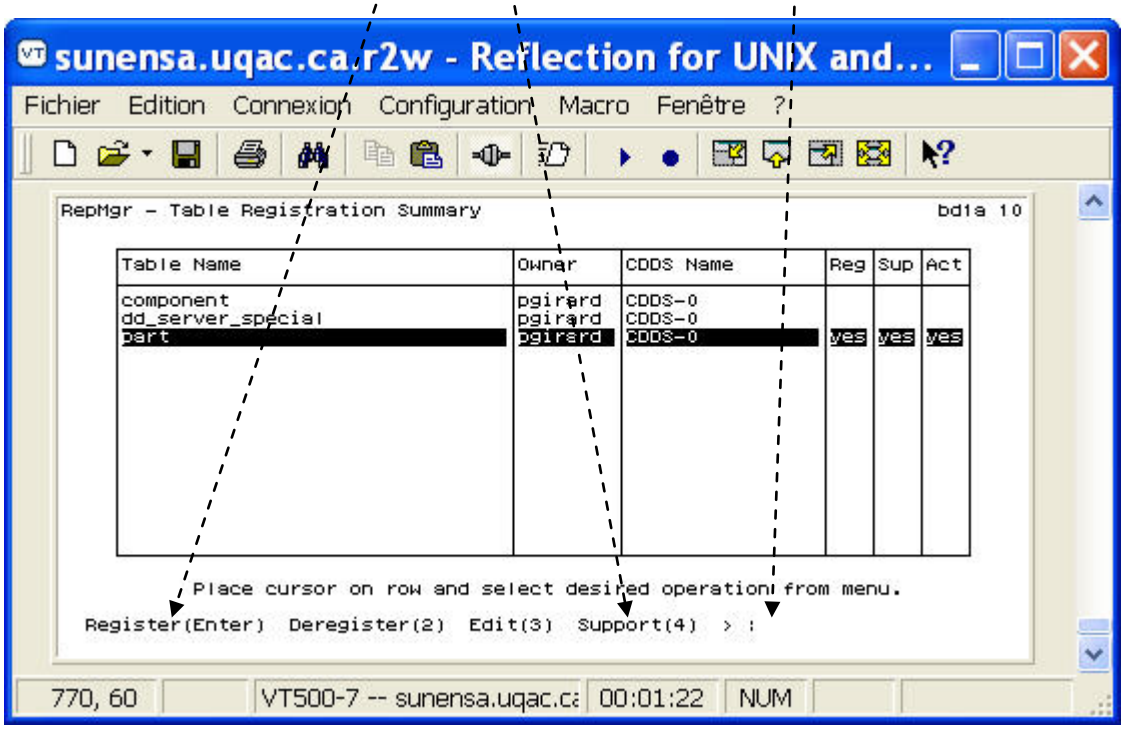


7.4 Activation of tables in the CDDS

This table *part* must be registered, supported and activated by using the appropriate options in repmgr or Visual DBA. The following messages confirm that *part* is now supported.

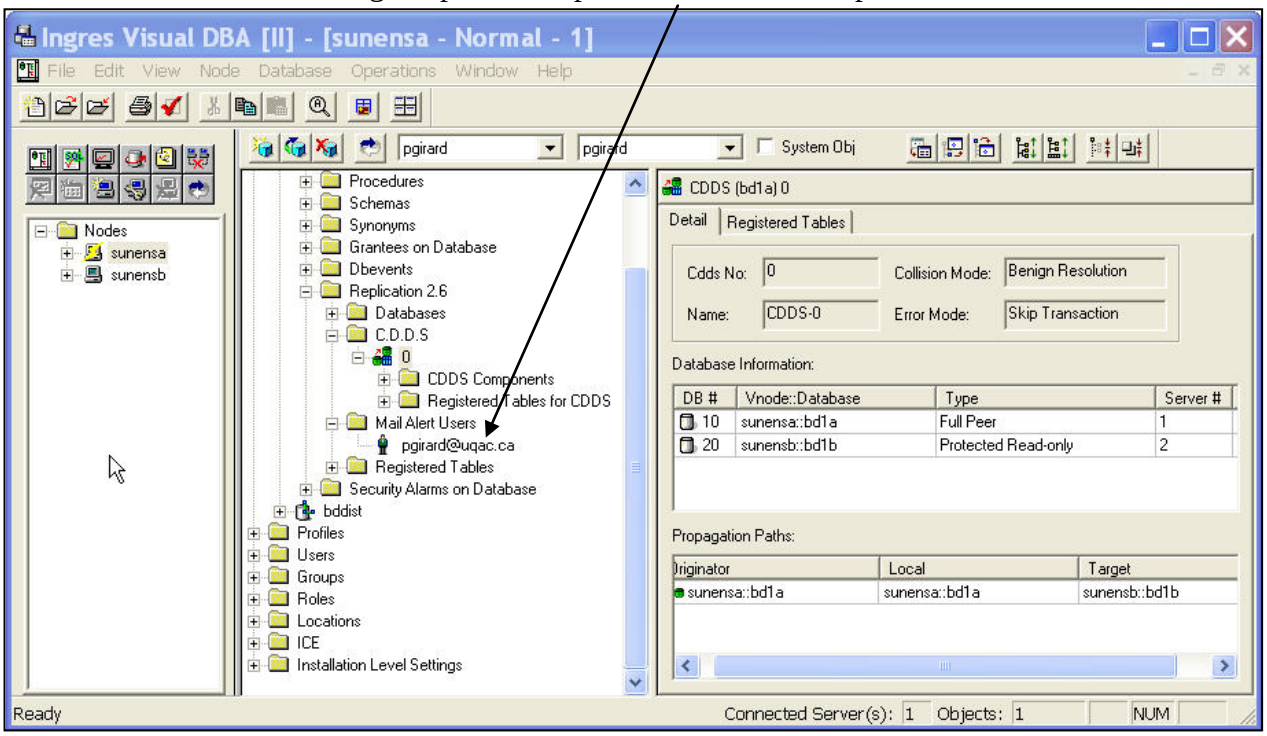
```
Processing table 'pgirard.part' . . .
Generating support tables for 'pgirard.part' . . .
Creating procedure 'part00002rmi' . . .
Creating procedure 'part00002rmu' . . .
Creating procedure 'part00002rmd' . . .
Support procedures for 'pgirard.part' have been created . . .
```

This figure illustrates that *part* is registered, supported and now activated with the menu options.



7.5 Specify an email address in case of replicator errors

Visual DBA showing the previous options with an email specified in case of error



7.6 Move configuration from *bd1a* on sunensa to *bd1b* on sunenb

Using the **repcat** menu, the configution made on *bd1a* may be moved automatically to *bd1b*.

```
RepMgr - Configuration Menu                                    bd1a 10

  Databases    Define Databases in the Replication Environment
  CDDS         Define Consistent Distributed Data Sets
  Tables       Register Tables for Replication
  MoveConfig   Move Configuration Information Between Databases
  Activate     Activate or Deactivate Change Recording
  MailList     Define List of Users to Receive Mail on Error
```

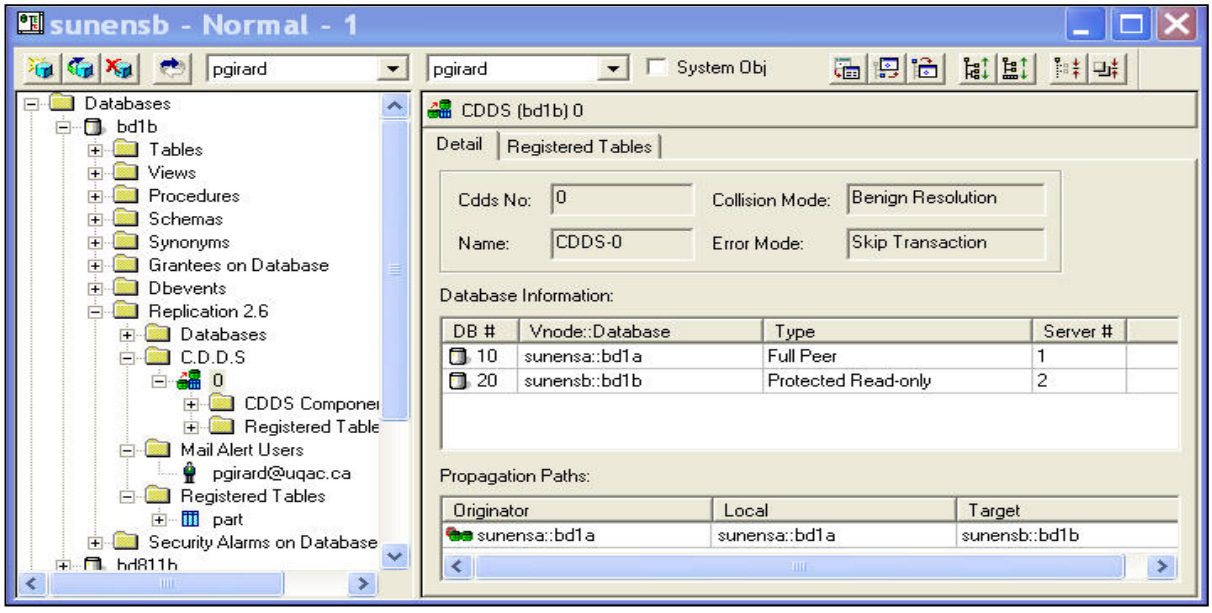
Execution of MoveConfig

```
Moving contents of Replicator catalogs to 'sunensb::bd1b' . . .
Processing table 'pgirard.part' . . .
Generating support tables for 'pgirard.part' . . .
Creating procedure 'part00002rmi' . . .
Creating procedure 'part00002rmu' . . .
Creating procedure 'part00002rmd' . . .
Support procedures for 'pgirard.part' have been created . . .
```

After moving the configuration, the next figure illustrates the database in *bd1b*. *bd1b* has been declared protected read only and the *bd1a* is considered to be the master database.

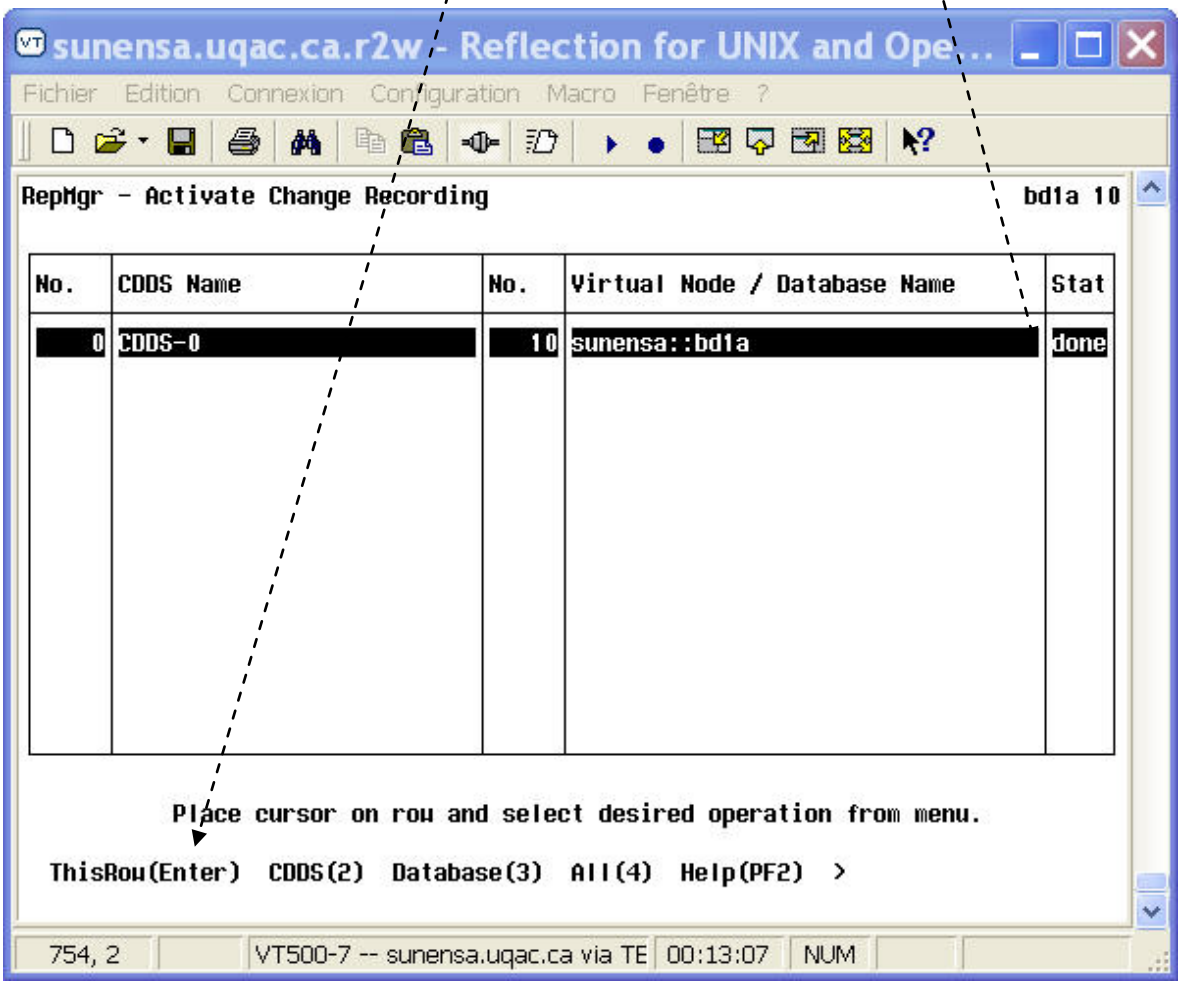
```
RepMgr - Database Summary                                      bd1b 20

 No.  Virtual Node / Database Name   Owner     DBMS Type  Remarks
  10  sunensa::bd1a                 pgirard   ingres     master database on s
  20  sunensb::bd1b                 pgirard   ingres     protected read only
```

Visual DBA on *bd1b* after moving config

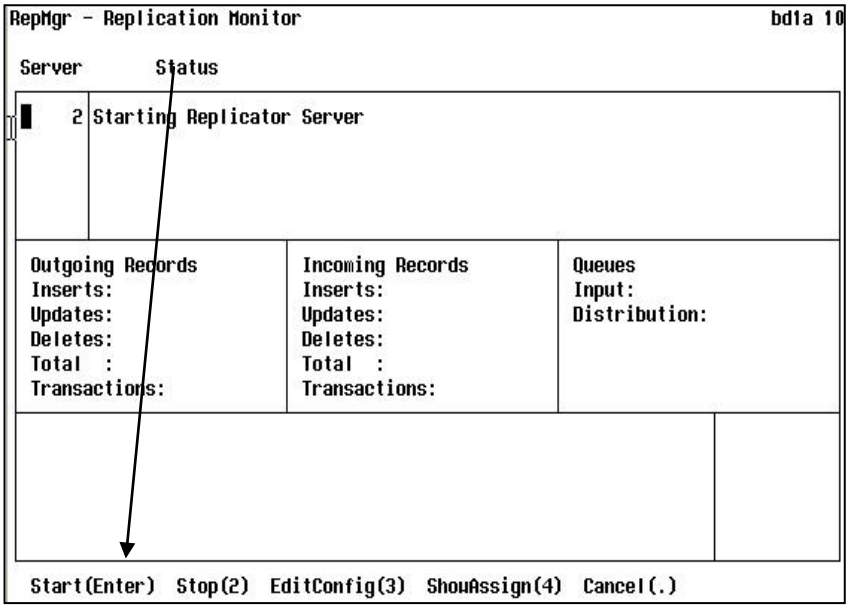


7.7 Activation of the recording of replicator

From the main menu of *repmgr*, activate the change for the CDDS-0. A status will indicate *done*.



7.8 Start the replicator server



7.9 Test a replication

Update *part* in *bd1a , commit the transaction* and do a select on *part* in *bd1b*

bd1a* update on table *part

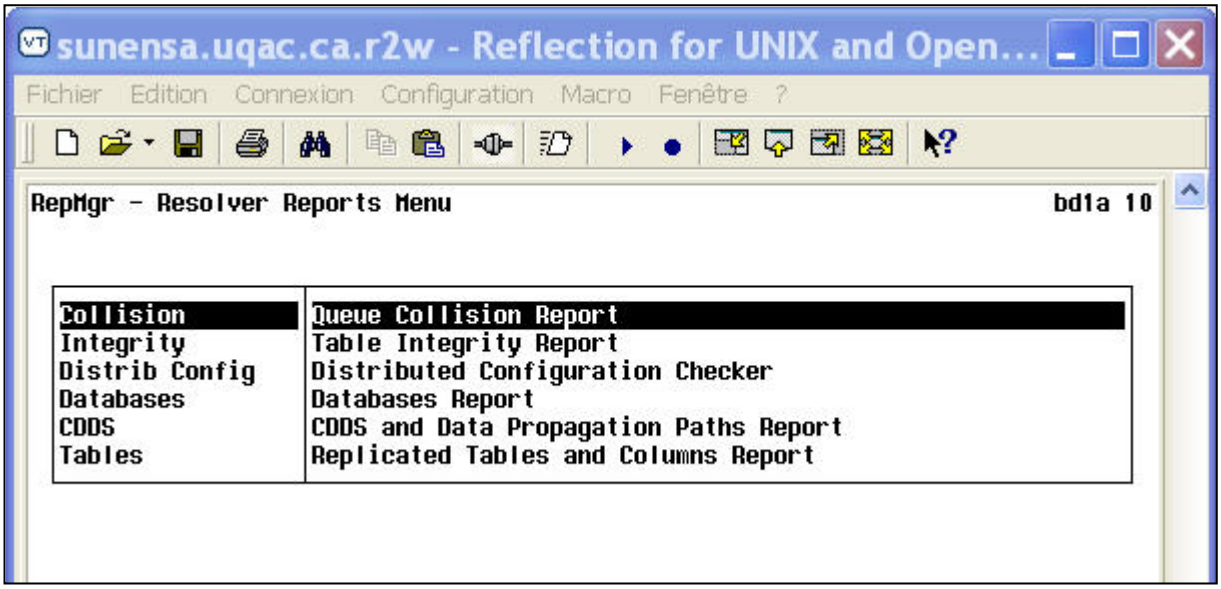
```
sunensa:pgirard> sql bd1a
INGRES TERMINAL MONITOR Copyright 2003 Computer Associates Intl, Inc.
Ingres SPARC SOLARIS Version II 2.6/0305 (su9.us5/00) login
Sun Mar 23 21:04:00 2008

* select * from part where part_id=1001;\g
Executing . . .

R,,,,,,7,,,,,,,,,,,,,,,7,,,,,,7,,,,,,7,,,,,,7,,,,,,,,,,7,,,,,,,,,,,,,,,,,,T
.part_i.part_name      .stock_.order_.min_qt.unit      .unit_price          .
5,,,,,,/,,,,,,,,,,,,,,,/,,,,,,/,,,,,,/,,,,,,/,,,,,,,,,,/,,,,,,,,,,,,,,,,,,6
.  1001.motor 1        .     0.     1.     5.unit      .            $2500.00.
F,,,,,,8,,,,,,,,,,,,,,,8,,,,,,8,,,,,,8,,,,,,8,,,,,,,,,,8,,,,,,,,,,,,,,,,,,G
(1 row)
continue
* update part set min_qty=10 where part_id=1001;\g
Executing . . .

 (1 row)
continue
* select * from part where part_id=1001;\g
Executing . . .

R,,,,,,7,,,,,,,,,,,,,,,7,,,,,,7,,,,,,7,,,,,,7,,,,,,,,,,7,,,,,,,,,,,,,,,,,,T
.part_i.part_name      .stock_.order_.min_qt.unit      .unit_price          .
5,,,,,,/,,,,,,,,,,,,,,,/,,,,,,/,,,,,,/,,,,,,/,,,,,,,,,,/,,,,,,,,,,,,,,,,,,6
.  1001.motor 1        .     0.     1.    10.unit      .            $2500.00.
F,,,,,,8,,,,,,,,,,,,,,,8,,,,,,8,,,,,,8,,,,,,8,,,,,,,,,,8,,,,,,,,,,,,,,,,,,G
(1 row)
continue
* commit;\g
```

The replicator server of *bd1a* sends this replicated row to *bd1b*

```
sunensb:pgirard> sql bd1b
INGRES TERMINAL MONITOR Copyright 2003 Computer Associates Intl, Inc.
Ingres SPARC SOLARIS Version II 2.6/0305 (su9.us5/00) login
Sun Mar 23 21:04:00 2008

* select * from part where part_id=1001;\g
Executing . . .

R,,,,,,7,,,,,,,,,,,,,,,7,,,,,,7,,,,,,7,,,,,,7,,,,,,,,,,7,,,,,,,,,,,,,,,,,,T
.part_i.part_name      .stock_.order_.min_qt.unit      .unit_price          .
5,,,,,,/,,,,,,,,,,,,,,,/,,,,,,/,,,,,,/,,,,,,/,,,,,,,,,,/,,,,,,,,,,,,,,,,,,6
.  1001.motor 1        .     0.     1.    10.unit      .            $2500.00.
F,,,,,,8,,,,,,,,,,,,,,,8,,,,,,8,,,,,,8,,,,,,8,,,,,,,,,,8,,,,,,,,,,,,,,,,,,G
(1 row)
continue
* commit;\g
```

7.10 Monitor the replicator

All basic elements can be monitored by selecting the appropriate option in ***repmgr***



Example of a collision report

```
 Displayed File: /tmp/rpcollsn.rpt

                              Ingres Replicator
                            Queue Collision Report
                          Local Database:  10  bd1a

UPDATE collision for table 'pgirard.part'
Local Database
Source DB:  10    Transaction ID:  1217237696    Sequence No:  1
Column information for 'pgirard.part' is missing.  Record deleted?
Remote Database Number: 20, Node:  sunensb, Name:  bd1b
Column information for 'pgirard.part' is missing.  Record deleted?

UPDATE collision for table 'pgirard.part'
Local Database
Source DB:  10    Transaction ID:  1217238580    Sequence No:  1
Column information for 'pgirard.part' is missing.  Record deleted?
Remote Database Number: 20, Node:  sunensb, Name:  bd1b
Column information for 'pgirard.part' is missing.  Record deleted?

UPDATE collision for table 'pgirard.part'
Local Database
```