

# Strategies and Techniques for Using Oracle7 Replication

Copyright © 1995 by Oracle Corporation

Revision 2.1.6.2 (95/05/08)

Dominic J. Delmolino  
Oracle Consulting  
Washington, District of Columbia, USA  
voice 415.506.4542  
Email ddelmoli@us.oracle.com  
Part A34042

## **Abstract**

Today's global organizations are looking for ways to expand the success of their client-server applications. Many organizations also want to provide a higher degree of availability for their critical systems. Vendors have been responding with various forms of replication technology to address both of these needs.

However, like many technologies, replication must be thoroughly understood by analysts and developers before they embark on an implementation schedule. Similar to a distributed systems strategy, replication tends to multiply single-system challenges by the number of sites participating in the replicated environment.

Oracle has offered a number of replication options since the introduction of the SQL\*Net networking layer. This paper explains the challenges associated with any replication implementation (regardless of software) and provides a framework for discussion of the various Oracle options available for replication.

**Key Words:** Replication, distributed systems, conflict avoidance, conflict resolution, data sharing.



## Contents

<b>1. OUR GOAL</b> .....	<b>1</b>
<b>2. BACKGROUND</b> .....	<b>1</b>
2.1 WHAT IS REPLICATION?.....	1
2.2 COMMON REPLICATION ARCHITECTURES .....	1
<b>3. REPLICATION ANALYSIS</b> .....	<b>1</b>
3.1 DATA CHARACTERISTICS.....	1
3.2 ACCESS REQUIREMENTS.....	1
3.3 DATA PLACEMENT.....	1
<b>4. REPLICATION CHOICES</b> .....	<b>1</b>
4.1 ROW-CHANGES OR PROCEDURE CALLS .....	1
4.2 SYNCHRONOUS OR ASYNCHRONOUS .....	1
<b>5. ORACLE REPLICATION OPTIONS</b> .....	<b>1</b>
5.1 SYNCHRONOUS TRIGGERS AND REMOTE PROCEDURE CALLS .....	1
5.2 READ-ONLY SNAPSHOTS .....	1
5.3 "WRITEABLE" SNAPSHOTS .....	1
5.4 MULTI-MASTERS.....	1
5.5 UPDATEABLE SNAPSHOTS.....	1
<b>6. ORACLE REPLICATION COMPONENTS</b> .....	<b>1</b>
6.1 NETWORK PROTOCOL .....	1
6.2 DATABASE LINKS.....	1
6.3 REFERENTIAL INTEGRITY CONSTRAINTS .....	1
6.4 STORED PROCEDURES .....	1
6.5 TABLE TRIGGERS.....	1
6.6 TWO-PHASE COMMIT PROTOCOL.....	1
6.7 DYNAMIC SQL AND PL/SQL .....	1
6.8 JOB QUEUE.....	1
6.9 ASYNCHRONOUS ("DEFERRED") PROCEDURE CALL QUEUE .....	1
6.10 REPLICATION CATALOG INTERFACE AND QUEUE.....	1
<b>7. REPLICATION CHALLENGES</b> .....	<b>1</b>
7.1 DATA MODIFICATION CONFLICTS.....	1
7.2 SECURITY.....	1
7.3 DATA LOADING .....	1
7.4 STRUCTURE CHANGES .....	1
7.5 PRIMARY KEY GENERATION.....	1
7.6 REPLICATION AS A DATABASE EVENT.....	1
7.7 INTEGRITY CONSTRAINTS AND DEPENDENCIES .....	1
7.8 TEXT AND IMAGE DATA .....	1
7.9 PERFORMANCE.....	1

<b>8. CONCLUSIONS</b> .....	<b>1</b>
<b>9. ABOUT THE AUTHOR</b> .....	<b>1</b>
<b>A. ENVIRONMENT PREPARATION</b> .....	<b>1</b>
A.1 NETWORK .....	1
A.2 INSTANCES .....	1
A.3 DATABASES .....	1

## 1. Our Goal

Our goal is to understand the issues and challenges associated with designing a replication environment, regardless of the implementation method chosen. The various Oracle replication options are discussed along with their differing characteristics and implementations. It is our hope that after reading and understanding this paper, readers will be comfortable and confident about their ability to design, create and manage a replicated environment.

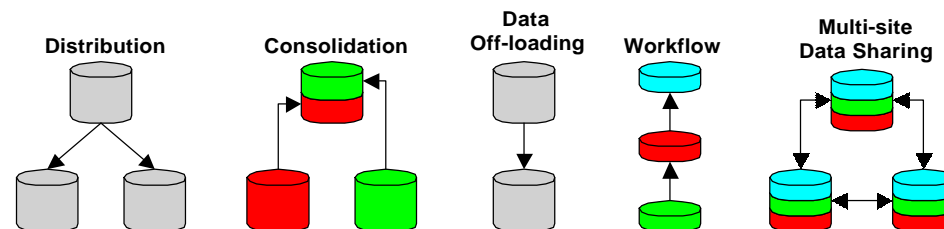
## 2. Background

### 2.1 What is Replication?

Replication is a technology that enables the placement of shared data at locations throughout an organization. It can be used to provide “fast” local access by eliminating long-distance connections between users and database servers. It can also provide corporate-wide high availability for any database application, making them resistant to single system failures.

### 2.2 Common Replication Architectures

Many replication environments are described in terms of their eventual (or desired) architecture. Customers often speak of their need for replication by referring to a “replication method” that they wish to employ. A few of the common descriptions follow:



**Distribution.** A typical example is that of a pricing list maintained by a company's headquarters and distributed to field offices. Usually the primary owner of this kind of information is the headquarters, and field offices use the information in a read-only manner.

**Consolidation.** Information on sales figures may be owned, entered and maintained at the local sales office level. This data may also be *partitioned* so that each office only sees data related to their own sales. A headquarters office may require full read-only access to sales from all offices in order to make forecasts or to anticipate demand.

**Information Off-loading.** In some companies, it may be desirable to reduce processing conflicts (in terms of system resources) between data entry activity and data analysis activity. An example may be an accounting system with a high transaction rate that also serves as a management reporting system. With replication, the data could be owned and maintained by the transaction system and replicated to another database location that could be specially tailored for read-only management reporting.

**Disaster Site Maintenance.** With full data replication, it becomes possible to maintain an accessible backup site for database operations. The “backup” site is fully accessible

and can be used to test read-only functions, or provide extra processing capability to deal with sporadic user demands. In the event of a system failure, the backup location will queue up changes to the failed system and automatically forward them after recovery is complete.

All of the prior examples employ a *static*, exclusive ownership model (often referred to as *primary site* ownership). An example of exclusive ownership where the data owner may change over time is best illustrated with a *work flow* example. This example illustrates *dynamic* exclusive ownership.

**Work flow.** A work flow model illustrates the concept of dynamic exclusive ownership by passing ownership of data from database location to database location within a replicated environment. An example is a combined order entry and shipping system that places orders and assigns ownership based on the “state” of an order (entered, approved, picked, shipped, etc.). Another example is an employee expenses reimbursement system in which expenses are routed to people and departments as part of an approval process. A replication work flow environment is extremely fault tolerant, allowing different functions along a flow to proceed independently in the event of failure at locations dedicated to other functions.

All of the above models assume *exclusive ownership* of data. However, there are several applications that can benefit from some kind of *shared ownership*, update-anywhere model.

**Multi-site Data Sharing.** Some applications have a business practice that allows multiple locations and users to update or change information related to a single piece of data. Examples might include reservation information (at a front desk or airline counter and via a central phone number) or any data being worked on by a team (such as a problem report being worked on by a team of support analysts and technicians, or a global sales licensing deal with sales representatives from many countries adding terms and conditions). Shared ownership models must be aware of the possibility for conflicting simultaneous access that may result in data inconsistency (data conflicts). Sophisticated replication environments must be able to handle conflicts as they occur, and they should provide a declarative way to describe the methods needed to resolve them. Understanding the business rules around how to handle simultaneous conflicting access to data is crucial to any shared ownership implementation.

Oracle’s replication options can enable the implementation of all of these scenarios.

### 3. Replication Analysis

Before choosing a replication architecture, it is important to understand the driving needs for replication within an organization. Blindly choosing a “common” architecture that may be inappropriate for an application artificially constrains the implementation methods that can be used. There is no current method that analysts can follow to define the requirements for a replicated environment. However, one can borrow ideas from many system requirements methods in order to help define replication needs. In particular, it makes sense to characterize the data to be replicated and to understand its data placement and access requirements.

#### 3.1 Data Characteristics

Typically, data can be described in terms of where it is created (for example, “Orders from St. Louis”, “Results from Germany”) or in terms of current state (“Orders that have

been entered”, “Orders that have been picked”, “Orders that have been shipped”). Some data can not be characterized in terms of its state or generating location, but rather in terms of its function (“Global reference data”, “Testing results”). The design of a replicated environment should begin with an investigation of the data to see if any imbedded locational information coincides with any piece of data.



**German Results**



**St. Louis Orders**



**Global Data**

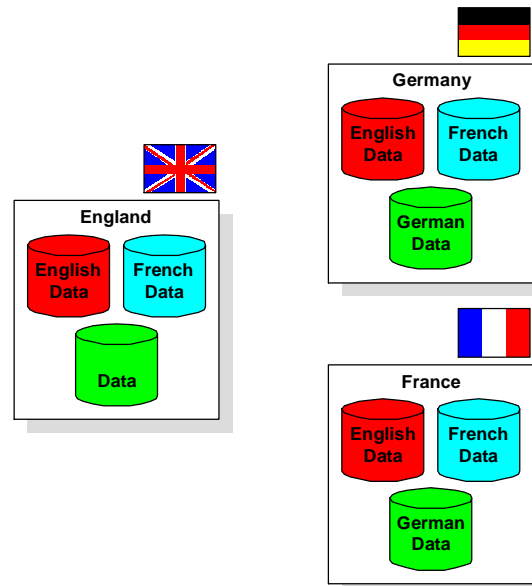
### ***3.2 Access Requirements***

Once you’ve gathered the defining characteristics of your data, it becomes easier to understand who needs access to it from which locations in your organization. You should be able to define where the data needs to be visible from (i.e. “German Results need to be visible in Germany, England and France”) and which locations need the ability to modify which data (“England and Germany need the ability to update German Results”).

### ***3.3 Data Placement***

With a matrix of data characteristics and access requirements in hand, you can begin to plan your data replication environment. Keep in mind that the requirement to access data from a location does not mean that data must be placed at that location. A plausible solution may be to buy a large, central server and some extremely robust network links between all locations. In general, replication should only be used if a client-server link to a central server provides inadequate performance, or if a measure of high-availability is required. After placing data at locations within your organization, you’ll be ready to decide how you want to keep all locations synchronized.

All data is replicated everywhere. England is a “headquarters” site that can change any data. High data availability is achieved by allowing updates to occur at more than one location for any data.



## 4. Replication Choices

Once you’ve defined your data access and placement requirements, you have several ways to maintain synchronized data across all locations. In particular, you can choose which kinds of events to replicate, *row-changes* or *procedure calls* and how to replicate them, *synchronously* or *asynchronously*.

### 4.1 Row-Changes or Procedure Calls

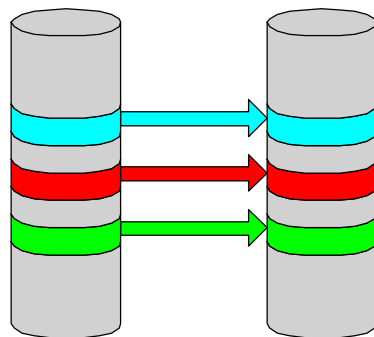
**Row-change replication.** Row-change (or data replication) forwards row changes to all other interested locations. Data changes are recorded at the database row level, and if both old and new values for the row are recorded, then the forwarded information contains everything needed to identify potential conflicts in a shared ownership model. It is the most transparent way to replicate an application’s on-line activities, which are typically based on a single-row (“Update Dominic’s address”) or small numbers or rows (“Mark all line items on this order as shipped”).

**Procedure call replication.** Procedure call (or process replication) forwards procedure calls and argument lists to all other interested locations. The procedures run individually at each location. It is the best way to replicate an application’s non-conflicting batch-oriented activities (“Give everyone a 10% raise”), since these kinds of activities deal with large row sets and are not concerned with row-by-row changes. While procedure call replication may offer performance benefits, all procedures must individually deal with any potential conflicts that arise as a result of their actions. A simplistic example is a bank that uses procedures to deposit money and add interest earned to savings accounts:

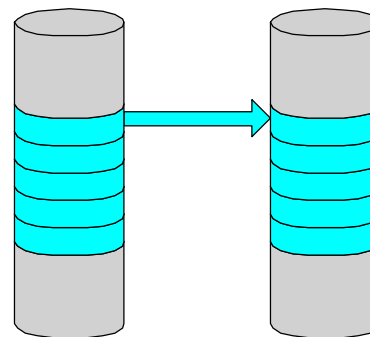
STEP	MAIN OFFICE	BRANCH OFFICE
Initial balance	\$10	\$10
Account holder deposits \$20 at bank branch (procedure call is queued to run at main office to deposit \$20)	\$10	\$30
Main office runs procedure to add earned interest of 10% (procedure call is queued to run at branch office to add 10% to the account)	\$11	\$30
Procedure calls are replicated	\$31	\$33

This data divergence conflict will be detected when using row-change replication, but will not be detected when using procedure call replication. This kind of replication is best used for batch-like operations that are run in a serial fashion from a single control point. Purge and archive operations are excellent examples of these kinds of processes.

**Row Changes**



**Procedure Call**



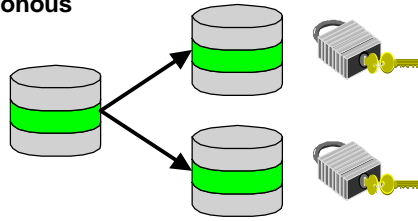
**4.2 Synchronous or Asynchronous**

For each kind of activity you wish to replicate (“Row changes against Orders”, “Procedure calls for Batch Deletes”), you may choose a method to deliver it to remote locations: *synchronously* or *asynchronously*.

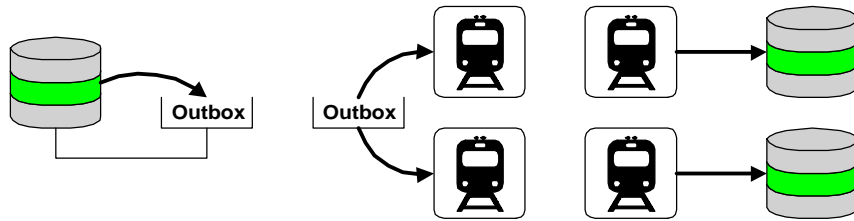
**Synchronous Replication.** Also referred to as “real-time” replication, synchronous replication immediately makes changes to all locations at the same time. Every activity is attempted at every location. If any one location is unavailable the activity does not succeed and is rolled back. Data conflicts are avoided by applying the changes to all locations within a single transaction and locking the intended data at every location. While this method guarantees strict data convergence and the most up-to-date data, it requires strong, fast network connections between highly-available servers.

**Asynchronous Replication.** Also referred to as “store-and-forward” replication, asynchronous replication immediately *records* information about replicated activities to be delivered at a later time. Using this method, local transactions are “de-coupled” from their delivery to remote locations and no distributed locks are acquired. Since no locks are acquired, there is the potential for data modification conflicts. However, this method is not dependent on a strong, fast network or highly-available servers. Data will converge to consistent values over time.

**Synchronous**



**Asynchronous**

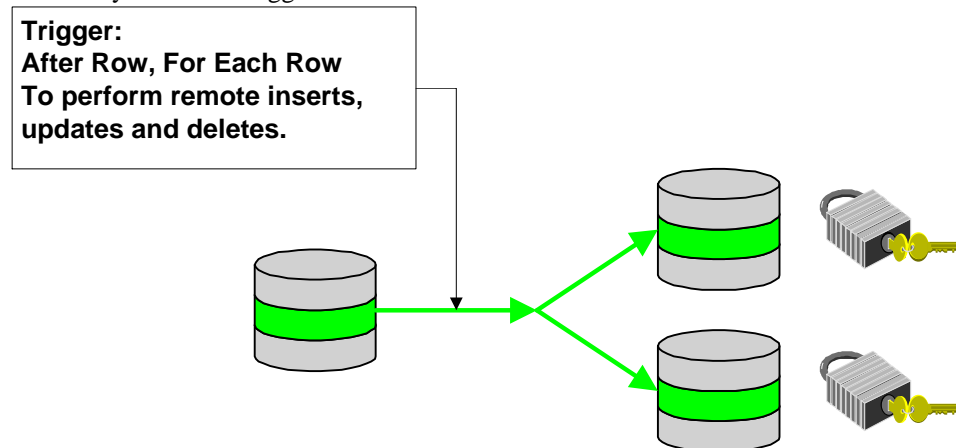


## 5. Oracle Replication Options

Oracle has offered a variety of replication methods and options since the advent of the SQL\*Net networking layer and particularly Oracle7. All of these options may be combined to form hybrids that are specifically tailored to a given environment.

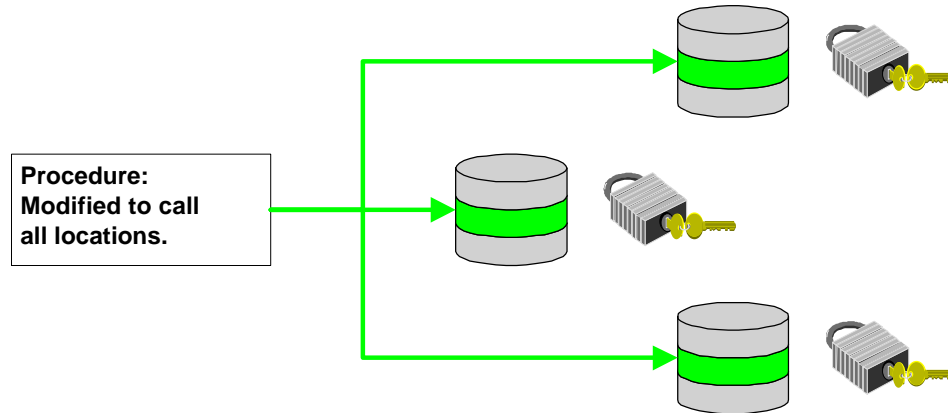
### 5.1 Synchronous Triggers and Remote Procedure Calls

With Oracle7, it is possible to write custom triggers on every table to replicate changes to remote locations. Triggers can be written to handle table changes (inserts, updates and deletes) for every row change. These triggers can perform remote changes using information from the initiating location. Transactions are transparently protected by the two-phase commit protocol and acquire distributed locks to avoid potential data modification conflicts. Impending versions of Oracle will automatically generate these kinds of synchronous triggers.



Stored procedures can be written to synchronously call their counterparts at remote locations. Like the synchronous triggers, these procedures can perform remote activity

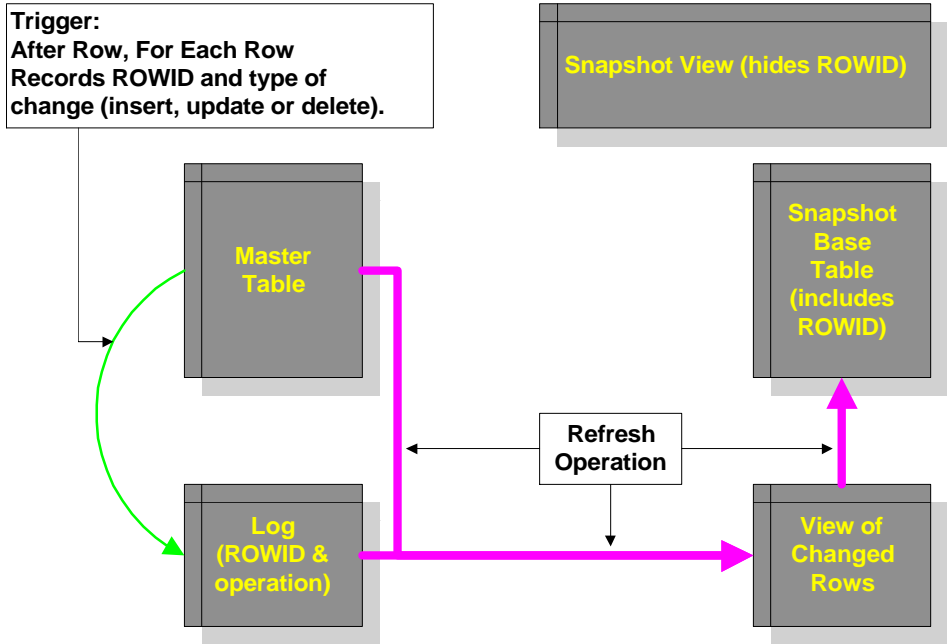
using information from the initiating location. Remote procedure calls are also transparently protected by the two-phase commit protocol.



## 5.2 Read-only Snapshots

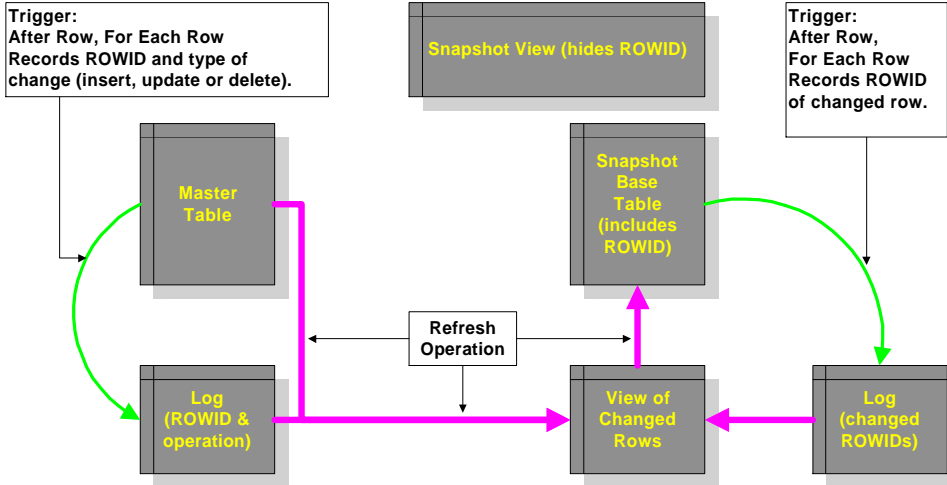
Oracle7 introduced the concept of a read-only snapshot. Read-only snapshots are objects that implement a “master-slave” architecture where the “slave” location is periodically refreshed with the most recent changes from the “master” location.

At a remote location, a user issues a *create snapshot* command which creates a table filled with the desired rows from the master location. *Where* clauses can be specified during snapshot creation to limit the number and types of rows in the snapshot. If the snapshot is based on a single table at the master location (a “simple” snapshot), these rows are related to their master counterparts using the master row’s *rowid*. (“Complex” snapshots can be created as the result of a complex query involving several tables or grouping functions. However, such snapshots cannot use the fast refresh option which depends on *rowids*). A view is placed on top of this table to hide the details of the master-slave row correspondence from the user. Applications can query this view just as they would the table at the master location. Extra indexes (non-unique) may be placed on the base table at the slave location to improve the performance of specific queries. In order to enable the fast refreshing of read-only simple snapshots, a *snapshot log* may be created on the table at the master location. The *snapshot log* is a table listing recently changed *rowids* and the action that occurred against them (insert, update or delete). The table is maintained by an *after row for each row* trigger on the master table. When a read-only snapshot is refreshed, the *snapshot log* is used to determine which rows to replace at the snapshot site. In Oracle7.1, snapshot refreshes are scheduled using the server *job queue*. In Oracle7, refreshes are handled by the *snapshot refresh processes*. Read-only snapshots cannot be modified at the slave location (since a refresh may overwrite any changed rows), and constraints and unique indexes are not allowed on them (they cannot be enforced during a refresh operation). A refresh is considered to be a single transaction within Oracle. At the end of a successful refresh, the snapshot is guaranteed to look exactly like its master table. In Oracle7.1 groups of snapshots may be refreshed together (a “refresh group”) in order to ensure consistency between them.



### 5.3 "Writeable" Snapshots

Oracle7.1 added the ability to make changes to snapshots at slave locations. By issuing the *create snapshot* command with the *for update* clause, another *snapshot log* is created. This *snapshot log* is created at the slave location and is a table that records which rows at the slave location have been changed. The table is maintained by an *after row for each row* trigger on the snapshot table. However, these changes are *not* sent back to the master location. They are used during snapshot refreshes to re-request the master rows from the master, overwriting any change at the snapshot site. These objects may be used to implement "scratchable" snapshots for "what-if" analysis, or to enforce the rule that any permanent changes must go through the master location.



## 5.4 Multi-Masters

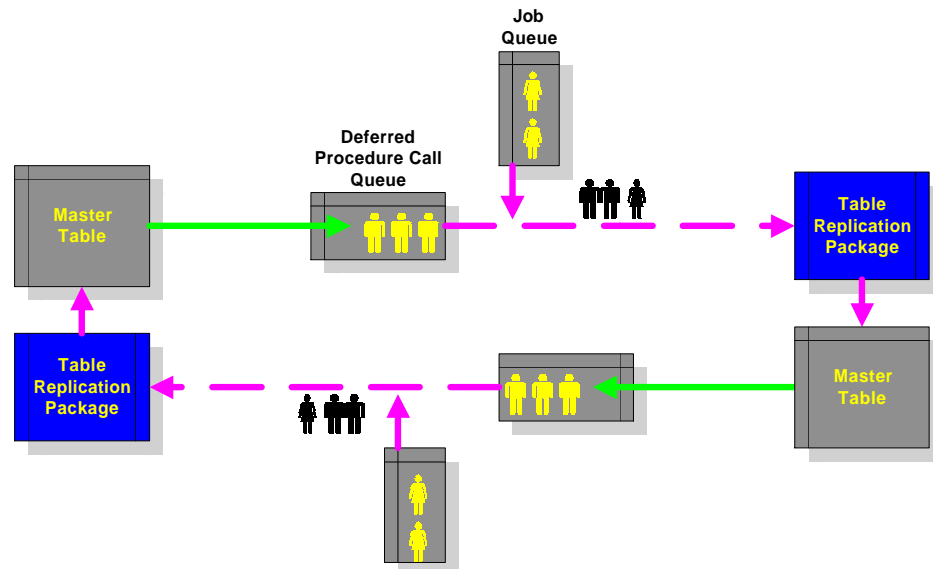
Oracle7.1's Replication Option adds a method called *multi-masters*. With the replication option, most kinds of Oracle objects can be registered and replicated to remote locations (all schema objects except sequences, database links and clusters). A global catalog of replicated objects is maintained using a PL/SQL API that is executed from a "master definition site". Replication support is automatically generated for row-changes (tables) and procedure calls.

If a table is registered with the replication catalog, a number of replication support objects are generated for it. A trigger is created to capture events that occur against the table. The trigger is an *after row for each row* trigger which places a call into the *deferred queue* tables to perform the corresponding action at remote locations for the individual row. The actual call placed into the *deferred queue* is to a procedure within a support package at a remote location. These support packages are automatically generated when the table is registered with the replication facility and are specific to each table. They contain procedures to handle incoming insert, update and delete activities on a row-by-row basis. They also contain the intelligence to make sure activities are not re-replicated.

A job in the *job queue* periodically forwards transactions in the *deferred queue* to all other remote master locations (the method is peer-to-peer broadcasting). These jobs can be scheduled on a location by location basis with differing intervals.

The entire before and after image of the row is conveyed and is required to make sure that potential conflicts can be detected. The support packages call internal conflict resolution routines to handle conflicts as they occur. Users have the ability to choose from which routines they'd like to invoke for conflicts that occur on a column by column basis. Available routines include: choose the values with the latest timestamp, add the values together, choose the values from the location with the highest priority, choose the highest value, choose the values with the earliest timestamp, discard the incoming values, overwrite the existing values, append a location identifier to the value to avoid a uniqueness violation or take the average of the conflicting values. Users can also write their own routines which can be registered with the conflict resolution methods for columns in the table. Methods can also be written to provide electronic notification of conflict detection and resolution.

In summary, multi-masters are full table, peer-to-peer, broadcast replication between participating database locations. Entire tables are replicated (no subsetting). Any changes are broadcast to all other participating database locations (no communities or routing). Changes are queued up at the originating location, and periodically "pushed" to all other locations on a location by location basis. Multiple locations can be pushed to at the same time. Failures during a push to one location do not affect any pushes to any other location, and transactions are kept in the queue for later propagation after error correction. Conflicts are resolved at every database location, with every one using the same routine to guarantee convergence.



### 5.5 Updateable Snapshots

Also available with the Oracle7.1 Replication Option are *updateable snapshots*.

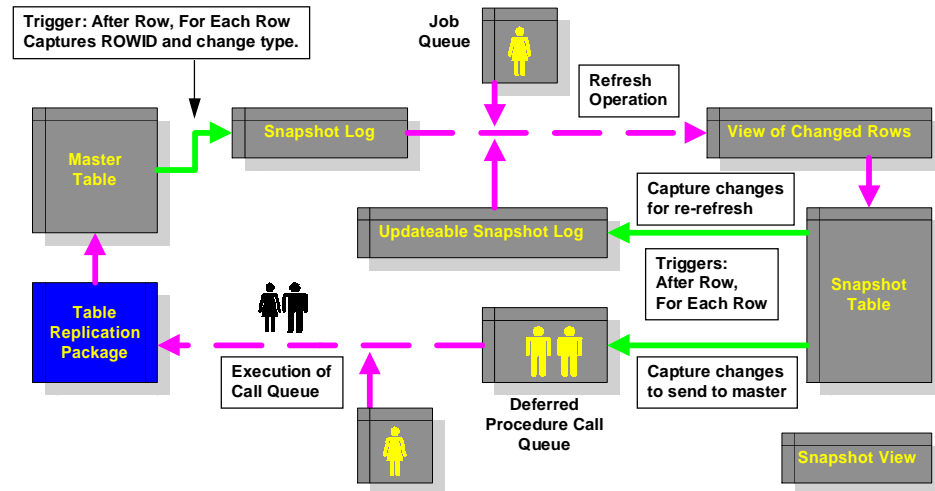
Updateable snapshots use the features of “writeable” snapshots and multi-masters to provide a combination of their operating characteristics.

In addition to the *snapshot log* at the slave location, a trigger is created to capture events that occur against the snapshot. The trigger is an *after row for each row* trigger which places a call into the *deferred queue* tables to perform the corresponding action at the snapshot’s master locations for the individual row. The actual call placed into the *deferred queue* is to a procedure within a support package at the master location. These support packages are automatically generated when the master table is registered with the replication facility and are specific to each master table. They contain procedures to handle incoming insert, update and delete activities on a row-by-row basis. They also contain the intelligence to make sure activities are not re-replicated.

During a snapshot refresh, the transactions in the *deferred queue* are first delivered to the master location, which applies them to the master table. After all transactions have been delivered to the master, a normal snapshot refresh operation occurs. Updateable snapshots abide by any decisions at the master regarding the application of changes. In particular, the master may override a change coming from a snapshot due to potential conflicts. At the end of a refresh, snapshots are guaranteed to look like their master table.

In summary, updateable snapshots are snapshots that have been enhanced to use the asynchronous procedure queue to send changes back to a snapshot master. Updateable snapshots must be horizontal subsets or full copies of single tables (no updateable complex snapshots, no vertical partitioning). Updateable snapshots send their changes only to their master (they do not broadcast), which in turn forwards those changes to all other masters. Conflicts are resolved only at master sites, and the snapshot is guaranteed to look exactly like its master after a snapshot refresh. To maintain referential integrity between snapshots, the Oracle Server allows you to define refresh groups of related

snapshots. All snapshots in a refresh group are refreshed in a transactionally consistent manner.



## 6. Oracle Replication Components

Oracle's replication options have been created to take advantage of many basic Oracle Server features. This means that all replication options are truly "building-block" technologies, written to take advantage of proven Oracle Server components. Replication has not been implemented using unproved technology, and these familiar components simplify administrative activities. Since Oracle replication options are fully implemented within the Oracle Server, all backup and recovery options also apply to a replicated environment. Customers should begin familiarizing themselves with the following Oracle Server components before embarking on a replication implementation.

### 6.1 Network Protocol

Database Servers participating in a replication environment use Oracle's SQL\*Net products to communicate with each other. Proper configuration of SQL\*Net 2.1 is required to ensure that inter server communication occurs without errors. Before setting up any replication method, all servers must be able to establish connections with each other using SQL\*Net. These connections should be tested from within a server-connected session, such as one established with SQL\*Plus, SQLDBA or Oracle Server Manager.

### 6.2 Database links

Database links are used to identify (from within an Oracle Server session) connections to remote databases. Replication options transfer information between servers identified by specific database links. An understanding of the interaction between database links and SQL\*Net, as well as the differences between *public* and *private* database links will be necessary to use them for any replication option.

### **6.3 Referential Integrity Constraints**

The Oracle Server can enforce referential integrity constraints between tables to maintain business rules on data. Primary key constraints are used to uniquely identify rows within tables and are used by some replication options to help identify conflicts. They are also used to provide information for conflict resolution.

### **6.4 Stored Procedures**

Much of the current replication features has been written using the Oracle Server PL/SQL stored procedure language. The facility is “coded” into the Server and invoked by calling the packages directly from within an Oracle Server session. Also, all of the propagation and conflict resolution routines defined are implemented as PL/SQL stored procedures that are visible to the user (for informational purposes only). A keen understanding of PL/SQL will be useful in tracking the tasks performed by Oracle replication.

### **6.5 Table Triggers**

The replication facilities use table triggers to capture changes to *only* those tables that have been registered for replication. By using triggers, the facilities capture changes from within every transaction (affecting registered tables), and those changes are protected by the full Oracle Server transaction mechanism. Since Oracle7.1 allows multiple triggers per event, those required by replication do not interfere with any potential user triggers on tables. Since these triggers are only placed on replicated tables, and since they fire only when those tables are changed, trigger activity generated by replication is proportional only to replication activity and does not depend on the overall size of a database or other activity that may be occurring.

### **6.6 Two-Phase Commit Protocol**

Oracle's transparent two-phase commit protocol protects the integrity of all transactions involving two or more databases. Since, by definition, replication involves the transfer of data between databases, it makes use of the Oracle Server two-phase commit protocol (during the delivery phase which is de-coupled from the initiating transaction) to guarantee the delivery of transactions from server to server.

### **6.7 Dynamic SQL and PL/SQL**

Multi-master and updateable snapshot replication create replication support for database objects by dynamically generating triggers, packages and stored procedures. By using dynamic SQL, these methods can use information about database objects stored in the Oracle Server data dictionary to create replication support specific to any database object.

### **6.8 Job Queue**

The Oracle Server contains a facility that allows user to submit PL/SQL blocks for repetitive batch processing. When submitting a PL/SQL block (a “job”), the user specifies a start time and repeat interval to indicate the first execution of a job and how often it should run thereafter. Jobs are stored in the Oracle Server job queue (implemented as a set of tables and fully protected by all backup and recovery

mechanisms), which is serviced by one or more background “job processes”. These job processes wake up periodically, scan the job queue for jobs that are ready to run, and execute them. Replication uses jobs to perform administrative activity on a periodic basis and to automatically forward transactions from server to server at timed intervals.

### **6.9 Asynchronous (“Deferred”) Procedure Call Queue**

Multi-masters and updateable snapshots use the Oracle Server deferred execution mechanism (“deferred transaction queue”) for data changes to replicated tables. The generic Oracle Server deferred execution mechanism allows users to place information about procedures for execution at a later time at any server. The deferred execution queue is also implemented as a set of tables and fully protected by all backup and recovery mechanisms (similar to the Oracle Server job queue).

### **6.10 Replication Catalog Interface and Queue**

All Symmetric Replication administrative commands are accessed via the PL/SQL replication catalog interface. These commands affect the *global* replication catalog (which is maintained on all servers participating in the replication environment), as well as the components required to perform replication activity. Replication administrative commands are “broadcast” to all servers in the replication environment and executed locally at each server. The replication administration queue lists the status of all commands at the local server, as well as information about global completion (displayed at the calling server). Problems with any of these components can result in errors when executing replication catalog interface commands.

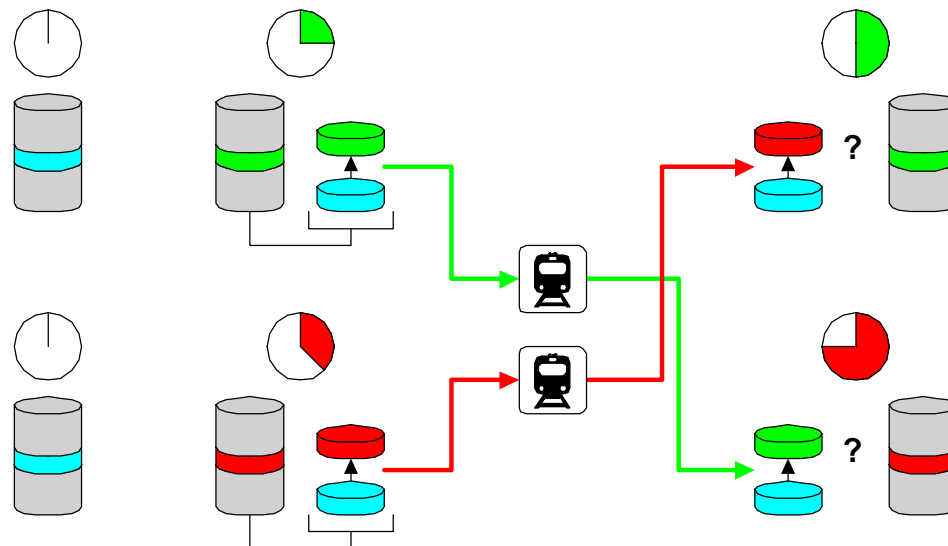
## **7. Replication Challenges**

### **7.1 Data Modification Conflicts**

The combination of access requirements and data object placement will help define potential data modification conflicts. Any business activity that asynchronously modifies a data object which exists at more than one location has the potential to cause conflicts. This may happen when users at different locations update a row to different, conflicting values.

Here is an example of an update conflict:

ACTION	DEFERRED QUEUE	NEW YORK	CHICAGO (HQ)
User in New York updates a customer's address from "250 Park Avenue" to "75 E. 45th Street"	New York to Chicago: update to "75 E. 45th Street" where address was "250 Park Avenue"	75 E. 45th Street	250 Park Avenue
User in Chicago updates same customer's address from "250 Park Avenue" to "101 7th Avenue"	Chicago to New York: update to "101 7th Avenue" where address was "250 Park Avenue"	75 E. 45th Street	101 7th Avenue
The deferred queue from New York to Chicago is "pushed"	<b>Conflict.</b> Can't find old row (someone else already updated it). Log the transaction for potential re-execution.	75 E. 45th Street	101 7th Avenue
The deferred queue from Chicago to New York is "pushed"	<b>Conflict.</b> Can't find old row (someone else already updated it). Log the transaction for potential execution.	75 E. 45th Street	101 7th Avenue



In general, whenever you allow users to update the same row from more than one location, and they update the same row within the deferred queue propagation window, data modification conflicts will occur. The following is a list of potential conflict types and when they occur:

**When Creating a Row.**

- The values for the row result in a uniqueness violation.
- Another row has the same primary key.

**When Removing a Row.**

- The row has already been removed.

- The row was changed after being removed from the originating location.

**When Updating a Row.**

- The old row was removed.
- The old changed its values.
- The new values for the row result in a uniqueness violation.

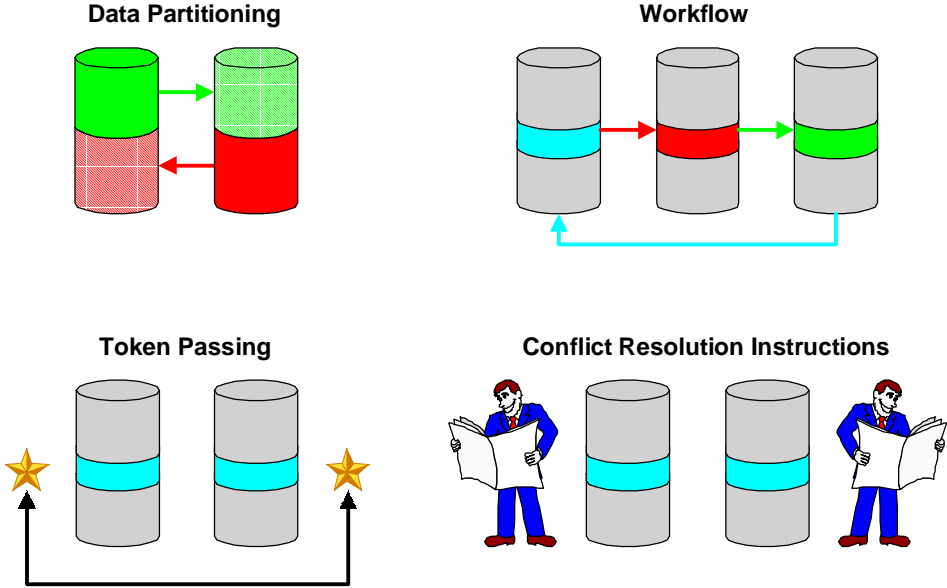
In general, there are four different ways to deal with potential data modification conflicts: *data partitioning*, *workflow*, *token passing* and *conflict resolution instructions*.

**Data Partitioning.** Data partitioning is a conflict avoidance technique where data subsets are only allowed to be modified from one location (i.e. “Chicago Orders can only be modified from Chicago”). It is often called a primary site static ownership model. While this method eliminates any potentials for conflict, it requires application changes (all applications must log into the owning location to modify its data) and does not provide any high-availability features for data modification (since all updates must be processed at only one location). It also introduces bottlenecks for all updates which pass through the owning location.

**Workflow.** Workflow is another conflict avoidance technique which makes sure that any individual data item can only be modified from one location at any given time. It is different from data partitioning in that the owning location for a data item may move from location to location. It is sometimes called a primary site dynamic ownership model. While this method eliminates any potentials for conflict, it requires application changes (all applications must make calls to the owning location to modify its data) and does not provide any high-availability features for data modification (since all updates must be processed at only one location). It also introduces bottlenecks for all updates which pass through the owning location.

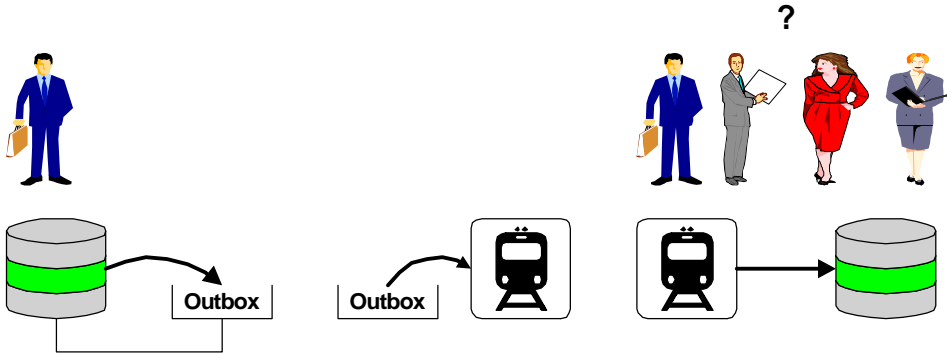
**Token Passing.** Token passing is a conflict avoidance technique where each data item contains an ownership token. Locations can broadcast their requests to “own” a data item and must get agreement from all other locations before acquiring the data item token. This method eliminates any potential for conflicts, but requires a synchronous broadcast request and agreement to hand over data item ownership. It is similar to acquiring a distributed lock on the data item.

**Conflict Resolution.** A conflict resolution method allows you to tell each database location what to do in case of a conflict emergency. It accomplishes this by handing out a consistent “playbook” to every location. When conflicts occur at a location, the system is intelligent enough to look up what to do in its “playbook” and do the right thing. Since all systems use the same set of instructions, all data values converge. No communication is required between systems to accomplish to conflict resolution. Only Oracle offers this capability in any kind of replication environment.



**7.2 Security**

Within a central system, it is relatively easy to define a security model that allows users to access and change data, while tracking and auditing their work. However, in a distributed environment there can be questions about how transactions are executed at remote locations. Should they be executed as the original user, or as a generic replication user? Given the demands of user-by-user security and auditing against the convenience of having a single replication user, there are many different security models which can be implemented.



Typical database applications have object owners (which own objects and usually don't log in to perform transaction activity), and object users (which don't own any objects, but perform transactions against objects owned by another user). With those user assumptions, there are four potential security models that may be implemented. All models refer to how transactions are executed when they are replicated to remote locations. The models are global access, single schema access, designated user access and mirrored user access.

**Global Access.** In this model, all replicated transactions are executed by the global replication administrator. No extra database links are required, but the global replication administrator does require the *execute any procedure* privilege. Keep in mind that the global replication administrator is a powerful user, and that any manually queued transactions will execute under its security domain at all remote locations. In this scenario, it is best to deny manual access to the queuing mechanism. Object users do not have to be mirrored on every location.

**Single Schema Access.** In this model, all replicated transactions are executed by an object owner. Database links are required from all object users to the object owner at the remote locations. Through these links, users have full access to the object owner schema. They are also limited to queuing transactions for one object owner schema only. However, no additional privileges are required. Object users do not have to be mirrored on every location.

**Designated User Access.** In this model, all replicated transactions are executed by a designated user. It's similar to the global access model, but allows the database administrator to specifically tailor the privileges of the designated user. The designated user may have privileges on multiple object owner schemas. Database links are required from all object users to the designated user at the remote locations. The designated user requires direct execution privileges on the replication support packages for all object owner schemas that may be affected by any replicated transaction. Object users do not have to be mirrored on every location.

**Mirrored User Access.** In this model, all replicated transactions are executed by the same user that initially submitted them. Object users must exist on all remote locations. Those mirrored users require direct execution privileges on the replication support packages for all object owner schemas that may be affected by any replicated transaction. Database links are required between mirrored users.

### **7.3 Data Loading**

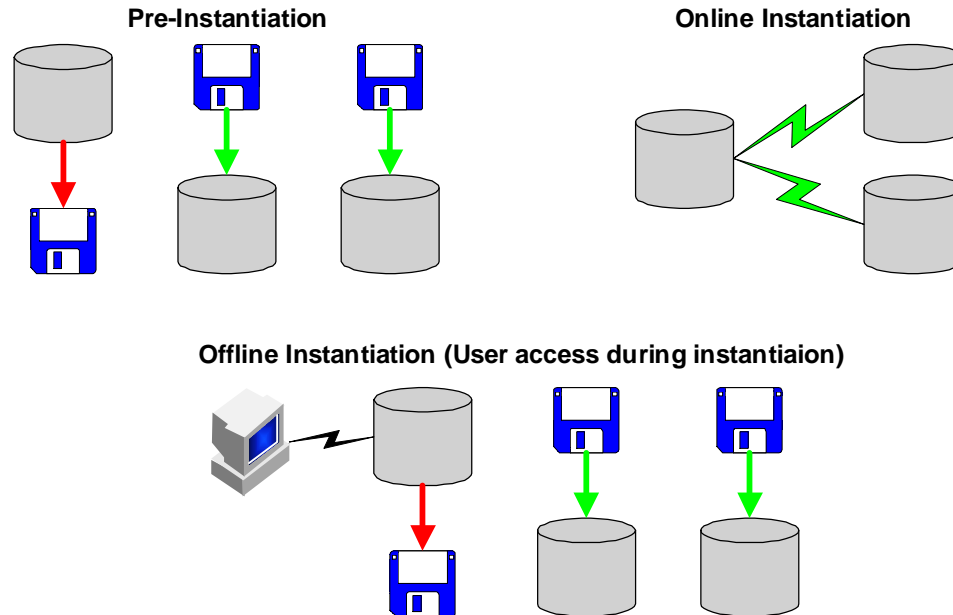
In a single, central system the problem of initially loading your system is relatively simple. In a replicated environment, the loading of a number of remote systems in a coordinated fashion is much more complex. Instantiation is the process of creating a loading a copy (or *replica*) at a "remote" location. There are several different ways to put your data copies at all locations participating in the replicated environment: *pre-instantiation*, *on-line instantiation*, and *off-line instantiation*.

**Pre-instantiation.** Copying all data to all locations before beginning the configuration of your replication environment is an easy way to guarantee identical data copies before allowing transaction activity. The method is probably a familiar one to most database administrators. No activity can take place against the data until all locations are up and configured into the replication catalog.

**On-line instantiation.** Given an initial populated location with all remote locations empty, replication can perform *on-line instantiation*. In this method, the replication facility creates objects over the network and copies their data to each location. This has the advantage of guaranteeing that all locations have the same data and gives the administrator a single point of control. However, there can be significant delay while data are copied over the network.

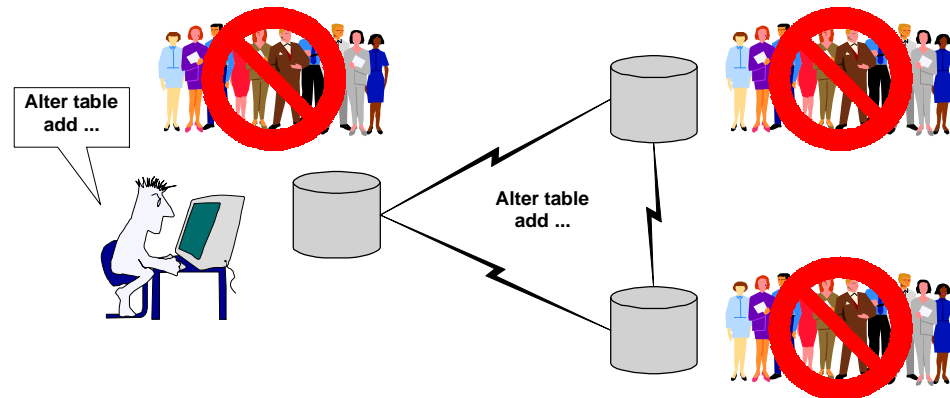
**Off-line instantiation.** An attractive option is to configure a replication environment from one location and inform it that other locations will be joining at a later date. The

initial location should “store up” changes destined for future locations and push them to the other locations as they come on line. Other locations can be loaded with data and synchronized with the initial location as they are registered into the replication catalog.



### 7.4 Structure Changes

In any database environment, it is desirable to have a quiet system during database structure changes. Usually database structure changes are scheduled to take place during off hours, without any user access or running batch programs. This kind of quiet time is difficult to achieve in a replicated environment. With Oracle’s replication options, all replicated schemas in the replicated environment must be *quiesced* before any administrative activity can take place. Pending changes must be completed against the “old” version of the database structure and any new changes must wait until all locations have the “new” version of the database structure. This means that there can be no changes to any data in any replicated schema (within the replicated environment) during the administrative period. Fortunately, Oracle’s replication options allow an administrator to alter all locations with a single command that can be broadcast out to the remote locations.



### 7.5 Primary Key Generation

Many applications use Oracle sequences to generate increasing numeric primary key values. However, in a replicated environment there is no such thing as a global sequence generator that will coordinate the generation of sequence numbers over long distances to distributed locations. Sequences at all locations must be modified to generate non-conflicting values. This can be done by ranging sequence values for each location (i.e. “New York gets 1-100, Chicago gets 101-200”), but this has a disadvantage of giving each location a limit on how many identifiers can be used. It also doesn’t handle situations where there are data volume differences between locations. A more practical solution is to include the location as part of the primary key, either by adding a column to the table or imbedding that information into the sequence (i.e. “All sequences increment by 100, New York sequences end in 01, Chicago sequences end in 02, etc.”).<sup>1</sup>

### 7.6 Replication as a Database Event

Since incoming replicated transactions are run just like normal transactions, they may trigger activities within the receiving database. However, you may wish to respond to a replicated transaction in a different manner than when you respond to a locally generated transaction.

A particular example is two replicated tables where changes to the first table result in a row being inserted into the second table (like a table “journal”). At an originating database, this process is maintained by a trigger on the first table. During replication, a row is inserted into the first table (the insert is captured for replication), the trigger is

---

<sup>1</sup> Astute customers will note that this method places a limit on the number of locations that can use this scheme (in this case, 100). If the Oracle number datatype could handle numbers up to infinity, the following method would handle an infinite number of locations: the first location’s sequences start with 1 and increment by 2, the second’s start with 2 and increment by 4, the third’s with 4 and increment by 8, etc. In general, location N starts with  $2^{(n-1)}$  and increments by  $2^n$ .

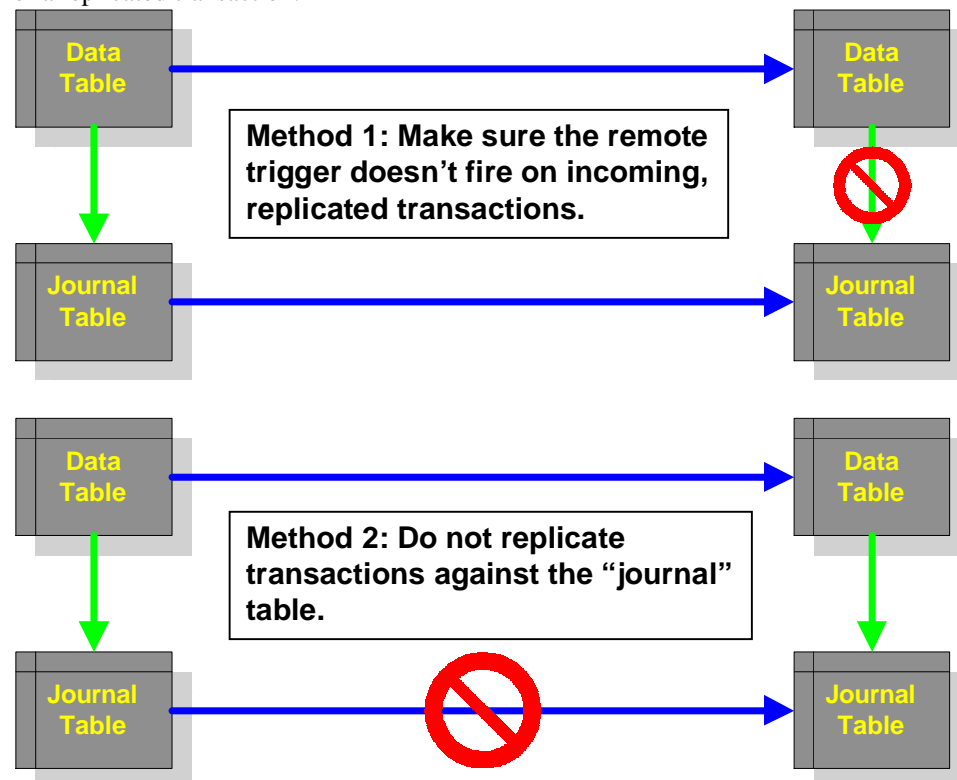
fired, and a row is inserted into the journal table (this insert is *also* captured for replication).

When these events are propagated, the original insert is run against the replica of the first table and the journalizing trigger is fired which results in a row being inserted into the journal table. However, the original insert into the journal table has also been replicated! This may result in duplicate rows or conflicts appearing against the journal table.

There are two solutions to this issue:

**Don't replicate tables which are the targets of triggers.** This means that the trigger will fire at both the originating and destination locations to maintain the information in the target, journal tables.

**Replicate all tables and make triggers aware of replication.** Each trigger can test the variables `dbms_reutil.from_remote` and `dbms_snapshot.I_am_a_refresh` to see if transactions are occurring as the result of replication activity. A simple *if-then-end if* wrapper around the trigger body will make sure that the trigger doesn't fire as the result of a replicated transaction.



### 7.7 Integrity Constraints and Dependencies

In order to support the replication of a table, many replication options require a primary key constraint. Since customers may also have other constraints on their tables, it is worth noting some characteristics of constraints as they work in a replicated environment.

**Naming Collisions.** When objects are registered with a replication facility and copied to other database locations, their complete structure (including their constraints) is sent over the network. Many customers created tables and let the Oracle Server use system generated names (i.e. SYS\_C99999) for any constraints (in particular, NOT NULL is a constraint that also receives its own name). Since constraint names must be unique within a schema (not within a table), this may cause problems when these table structures are replicated to another database location (if the schema at the target location has already “generated” constraints with the same name). The solution to this problem is to explicitly name all constraints (even NOT NULL) to avoid potential conflicts in system-generated names.

**Primary Key Indexes.** Related to the naming problem is the linkage between primary key constraint names and the names of the indexes used to enforce the primary key constraint. In order to replicate a table in a consistent manner, you should make sure that the name of the index and the name of the constraint are the same. This can be done by disabling and re-enabling the constraint, which drops and re-creates the index with the proper name. (This disconnect between the name of the index and the name of the constraint typically occurs in environments that had created *unique* indexes to enforce the primary key prior to creating the primary key constraint).

**Dependencies.** A common problem when replicating a group of objects is making sure that all referenced objects exist at the target database location. For example, in order to replicate a view, you must make sure you replicate the tables that the view is based on before copying the view definition to the target database location. The replication facility is intelligent enough to make sure objects appear in the correct order at the target database location, but you must make sure that any referenced objects outside the replicated set are installed prior to initiating the copy.

## 7.8 Text and Image Data

Currently there are no automatic facilities to handle replication of and updates to data types that can conceivably be up to 2GB in size. This means that LONG and LONG RAW data cannot be replicated with any replication option (all replication methods share this limitation, regardless of vendor). Alternative means must be used to move these large objects around a replicated environment, or their use should be questioned to see if a smaller datatype can perform their function.

## 7.9 Performance

In any replication system, there are two components to performance: *storing* and *forwarding*. In a synchronous system, these procedures are combined into a single operation. Both the store and forward must occur within the transaction to make sure real-time synchronous replication takes place. In asynchronous replication, the *store* and *forward* activities are de-coupled, and their performance can be measured independently.

**Storing.** Storing is the act of recognizing and recording the information necessary to replicate a transaction. It should not be related to how many locations will be receiving the eventual transaction. It should only be related to events which need to be replicated, and should not have to deal with any other database activities. Failure to record a transaction should have minimal impact on the operation of the source database.

Oracle's replication support objects (triggers and packages) follow these rules to minimize the costs of storing replicated transactions.

**Forwarding.** Forwarding is the act of delivering and applying transactions at remote locations. Its performance depends on the relative speeds of the delivering and receiving systems, as well as the network throughput. It needs to enforce transactional ordering and to guarantee the delivery of replicated transactions. It should compress as much information as possible into a stream of deliverable information. It should be able to handle intermittent network failures and provide robust restart and throttling options. Oracle's replication delivery mechanism (the deferred procedure queue) follows these rules to guarantee the integrity of replicated transactions.

Tuning a replication environment means understanding how to isolate and direct efforts at each of these components. Both components incur a cost in any replicated environment (regardless of vendor). Understanding how each method implements these mechanisms will allow administrators to tune them accordingly.

## 8. Conclusions

Any replicated database environment can enable corporate-wide access to enterprise shared data. Benefits include local access and highly available applications across the national or global organization. This paper has highlighted many of the issues surrounding the use of any replication product, particularly Oracle's replication options. After this paper, readers should feel comfortable with replication and their ability to design and build a corporate replicated environment for any database application.

## 9. About the Author

Dominic Delmolino is a Principal Consultant / Northeast Technical Manager with Oracle's Server Technologies Consulting team and currently coordinates Oracle's replication consulting. During his 5 years with Oracle Corporation, he has visited over a hundred clients around the world performing installation, upgrade, migration and database design and analysis engagements. His specialties include replication, distributed systems and configuration planning and management. Dominic is based in Washington, District of Columbia, USA but may be reached at his California voice-mail (415.506.4542) or on the internet at [ddelmoli@us.oracle.com](mailto:ddelmoli@us.oracle.com).

## A. Environment Preparation

This environment preparation section is intended as a convenient guide for those customers wishing to set up and use Oracle7.1's Replication Option.

### A.1 Network

Proper configuration of SQL\*Net 2.1 is required for replication between Oracle Server database locations. Database administrators should pay special attention to correct *listener.ora* and *tnsnames.ora* files on every location participating in the replicated environment. In particular, every location should be able to connect to every other location using an alias from the local *tnsnames.ora* file. It will ease administrative burdens if the *tnsnames.ora* file is global to the replicated environment, and each database location has one and only one alias. Keep in mind that these aliases will be

used by database links for communication from within database sessions, so special care should be taken to make sure all of them work properly.

### A.2 Instances

Each Oracle Server instance must be configured correctly for replication to work. Several parameters must be set to enable the Oracle Server job queue, triggers required for replication, and an enlarged shared memory pool area. The following parameters are involved in replication:

PARAMETER NAME	RECOMMENDED VALUE	COMMENTS
<b>job_queue_processes</b>	greater than or equal to 1	The number of processes that handle jobs submitted to the job queue.
<b>job_queue_interval</b>	greater than or equal to 1	How often the job queue processes wake up (in seconds) to check the job queue for pending jobs.
<b>compatible</b>	7.1.0.0	Enables multiple triggers per event type
<b>shared_pool_size</b>	greater than 10MB (greater than 20MB if possible)	The number of bytes in the shared memory area required to handle replication PL/SQL
<b>global_names</b>	true	Enforces the matching of database link names with target database names
<b>db_domain</b>	domain name for database	Recommended when using database global naming standards

### A.3 Databases

Each database participating in a replicated environment must be configured with the database objects necessary for symmetric replication. Symmetric Replication requires the Oracle Server distributed option. In addition to the normal Oracle Server database catalog, the procedural option must be installed (usually by running the **catproc.sql** Oracle Server administrative script while connected internally in a **sqldba** session). Since the asynchronous procedure queue and replication catalog are owned by the **system** user, it may be advantageous to follow Oracle Service's OFA Standard regarding the **system** schema. That standard recommends a separate tablespace to contain objects owned by **system** as opposed to the database kernel objects owned by **sys**. Database administrators may also appreciate the dedication of a tablespace for replication information and queues.

The replication objects may be installed by running the Oracle Server administrative script **catrep.sql**. This script installs the replication queues and catalog under the **system** user, and creates the PL/SQL replication packages in the **sys** schema.

There are two special database users required by symmetric replication: the *sys surrogate user* and the *replication administrator*.

**Sys surrogate user.** Since all of the installed symmetric replication packages are owned by **sys**, many replication administration commands run under the security domain of **sys**. Some of these commands are distributed and require access to **sys** on remote locations.

In order to eliminate the requirement for database links connecting the **sys** users on all master locations (a potential security hole), we can create a *sys surrogate user* on every master location. This user has the subset of privileges required by **sys** for the distributed administration commands. The following set of commands (run while connected as **sys**) will create a *sys surrogate user*:

```
rem Create surrogate user
create user surrogate user name identified by surrogate
user password;
rem Grant privileges necessary for replication
administration
execute dbms_repcat_auth.grant_surrogate_repcat(
'surrogate user name' );
```

Note that no one will be logging in as the *sys surrogate user*. These commands should be run on every database location in the replicated environment.

**Replication Administrator.** The replication administrator is the “control user” for all *master* replication environments on a particular database location. All *master* replication commands should be run while connected to the replication administrator user. A *replication administrator* can be responsible for managing all replicated schemas (a *global administrator*) or just a schema in which it owns objects (a *schema administrator*). The *replication administrator* is a powerful database user, with the ability to create, drop, alter and perform commands against any database object (in the case of a *global administrator*). Proper configuration of this user is essential for symmetric replication. Since many of the replication administrative activities will be performed by a job in Oracle Server job queue, and since jobs cannot inherit privileges through a database role, all privileges must be directly granted. The following set of commands (run while connected as **sys**) will create a *replication administrator*:

```
rem Create replication administrator
create user replication administrator user name identified
by replication administrator password;
rem For a global administrator:
execute dbms_repcat_admin.grant_admin_any_repschema(
'replication administrator user name' );
rem For a schema administrator:
execute dbms_repcat_admin.grant_admin_repschema(
'replication administrator user name' );
```

These commands should be run on every *master* database location in the replicated environment.

**Snapshot Administrator.** Every snapshot schema is administered individually by the schema owner in a fashion similar to a schema administrator. All *snapshot* replication

commands should be run while connected to the snapshot administrator user for each schema. The following set of commands (run while connected as **sys**) will create a *snapshot administrator*:

```
rem For a snapshot schema administrator:
execute dbms_repcat_admin.grant_admin_repschema( 'snapshot
schema administrator user name' );
```

These commands should be run on every *master* and *snapshot* database location for every snapshot schema in the replicated environment.

**Database links.** In addition to special users, every database location in the replicated environment requires database links. These database links are used to broadcast replication administration commands and transfer data from one database to another. Database links are required between all master database locations and between all snapshots and their associated master database location. Three kinds of links are required:

OWNER	LINK	COMMENT
public	<i>target database name</i> <b>using</b> 'SQL*Net alias'	Global database name of target database. No username or password clause. All private links use this link to lookup the SQL*Net alias.
<b>sys</b>	<i>target database name</i> <b>connect</b> <b>to</b> <i>sys surrogate user</i> <b>identified by</b> <i>sys surrogate</i> <i>user password</i>	<b>sys</b> connects to its surrogate user on other locations.
<i>replication administrator</i>	<i>target database name</i> <b>connect</b> <b>to</b> <i>replication administrator</i> <i>user name</i> <b>identified by</b> <i>replication administrator user</i> <i>password</i>	The <b>replication administrator</b> connects to itself on other locations.
<i>snapshot administrator</i>	<i>target database name</i> <b>connect</b> <b>to</b> <i>snapshot administrator user</i> <i>name</i> <b>identified by</b> <i>snapshot</i> <i>administrator user password</i>	The <b>snapshot administrator</b> connects to itself on other locations.

All database links should be tested (from every database location to every other database location) before starting to configure symmetric replication. If any database link is invalid, symmetric replication will fail in many different ways.