# Oracle7™ Server Distributed Systems, Volume I: Distributed Data

**Release 7.3**

February 1996

Part No. A32543–1

Oracle7™ Server Distributed Systems, Volume I: Distributed Data, Release 7.3

Part No. A32543–1

Primary Authors: Jason Durbin, Laura Ferrer
Contributors: John Frazzini, Mark Hill, Merrill Holt, Ken Jacobs, Mark Jarvis, Valerie Kane, Virginia Lyons, Ed Miner, Valarie Moore, Maria Pratt, Gordon Smith, Sandy Venning, Steve Viviant, Rick Wessman

# Preface

This manual describes the concepts necessary for implementing Oracle7 Server, release 7.3 in a distributed environment It also introduces the tools and utilities available to assist you in implementing and maintaining your distributed system.

## How *Oracle7 Server Distributed Systems, Volume I* is Organized

**Chapter 1: Understanding Distributed Systems**

This chapter provides an introduction to client–server model, distributed system and networking concepts and terminology. It is recommended reading for anyone planning to implement or maintain a distributed system.

**Chapter 2: Network Administration**

This chapter provides an overview of networking related issues in a distributed environment including setting up a network, adding and dropping sites (nodes) from a distributed system, monitoring network services, etc.

**Chapter 3: Network Administration Utilities**

This chapter discusses some of SQL*Net's administrative capabilities and utilities. It also discusses some of the tasks you perform to configure and maintain your network.

**Chapter 4: Database Administration**

This chapter provides guidance on implementing and maintaining databases in a distributed system and provides an introduction to the Oracle utilities provided for database administration, for example, Oracle Server Manager.

**Chapter 5: Distributed Updates**

This chapter describes how Oracle's maintains the integrity of distributed transactions involving updates through its transaction recovery mechanism.

**Chapter 6: Security Issues**

This chapter discusses both network and database security issues that are unique to distributed systems. Because open systems are inherently vulnerable unless security measures are taken to prevent breaches, the issues discussed in this chapter are quite important.

**Chapter 7: Application Development**

This chapter describes the special considerations that are necessary if you are designing an application to run in a distributed system. Topics covered include: where data should be located for fast access, how to name objects uniquely throughout the distributed system, how to control connections through database links, how to design distributed queries, and how to use the XA Library.

**Appendix A: Operating System Dependencies**

This appendix is a summary of all the operating system–specific references contained within this manual.

## Conventions Used in This Guide

The following conventions are used in code fragments in this guide:

| | |
|---|---|
| UPPERCASE | Uppercase text identifies text that must be entered exactly as shown. For example: `SQLPLUS` *`username/password`* / `INTO TABLENAME 'table'` |
| *lowercase italics* | Lowercase italics text is used for emphasis and to indicate glossary terms. It also identifies a variable for which you should substitute an appropriate value. Parentheses should be entered as shown. For example: `VARCHAR (`*`length`*`)` |
| Vertical bars \| | Vertical bars indicate alternate choices. For example: `ASC \| DESC` |
| Braces { } | Required items are enclosed in curly braces, meaning you must choose one of the alternatives. For example: `{`*`column_name`*` \| `*`array_def`*`}` |
| Square brackets [ ] | Optional items are enclosed in square brackets. For example: `DECIMAL (`*`digits`*` [ , `*`precision`*` ])` |
| <*operator*> | SQL operators are indicated by <*operator*>. For example: `WHERE `*`x <operator>`*` x` |
| Ellipses ... | Repeated items are indicated by enclosure in square brackets and ellipses. For example: see below |

```
WHERE column_1 <operator> x
  AND column_2 <operator> y
 [AND ...]
```

## Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of this manual is a Reader's Comment Form which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form has been used or you would like to contact us, please contact us at the following address:

Oracle7 Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA  94065
Fax: (415) 506–7200

# Contents

CHAPTER

*1*

# Understanding Distributed Systems

This chapter introduces the concepts behind distributed systems and explains how the client–server model, networks, and distributed databases can create an efficient distributed environment. Replication issues (basic replication, the advanced replication option, among others) are discussed in *Oracle7 Server Distributed Systems, Volume II.*

This chapter covers the following topics:

- the client–server model
- Oracle7 Server and the client–server model
- introduction to distributed systems
- networking issues
- SQL Statement execution in a distributed system

It is assumed that you are familiar with the concepts and terminology in the *Oracle7 Server Concepts* manual. Much detailed information is also provided in the *Oracle7 Server Administrator's Guide*, and other Oracle7 Server documentation. For introductory Networking issues, you should be familiar with *Understanding SQL*Net*.

Information in this book is relevant only to Oracle7 Server, release 7.3 and higher with the distributed option.

## Introduction

A *distributed system* is one in which both data and transaction processing are divided between one or more computers connected by a network, each computer playing a specific role in the system.

Understanding distributed systems requires a knowledge of a number of areas including system architecture, networking, transaction processing, security, among others. *Oracle7 Server Distributed Systems, Volume I* provides you with an introduction to the basic concepts and terminology required to understand distributed systems. It also discusses the components of a distributed system (for example, computers, workstations, networks, and security).

*Replication* (insuring that the data at all sites in a distributed system reflects any changes made anywhere in the system) is discussed in detail in *Oracle7 Server Distributed Systems, Volume II*, which covers both basic replication (standard with the distributed option) and the advanced replication option.

In this chapter, the client–server model is discussed first because this model is essential to distributed systems. You will be introduced to the concept of computers that act as clients and / or servers. Subsequently, you will be shown how the client–server model fits into a distributed system and how networks facilitate

Subsequent chapters of *Oracle7 Server Distributed Systems, Volume I* introduce networking concepts, terminology, and related Oracle networking products. Utilities for both network and database administration are also discussed, as is *transaction recovery management* (the mechanism that insures that all sites in a distributed system participating in a transaction that updates data do the same thing).

Although developing applications for a distributed system is similar to developing for non–distributed systems, the application development chapter discusses certain issues that can affect an application in a distributed system. It also documents Oracle's support for X / Open's XA Library.

*Oracle7 Server Distributed Systems, Volume I* does not attempt to instruct you in implementing your distributed system. All of this information already exists in the Oracle7 Server and Oracle Network Products documentation sets. This book does provide pointers, wherever possible, to the manuals that contain instructions specific to completing certain tasks.

## The Client–Server Model and Distributed Systems

The client–server model is basic to distributed systems. It is a response to the limitations presented by the traditional mainframe client–host model, in which a single mainframe provides shared data access to many dumb terminals. The client–server model is also a response to the local area network (LAN) model, in which many isolated systems access a file server that provides no processing power.

Client-server architecture provides integration of data and services and allows clients to be isolated from inherent complexities, such as communication protocols. The simplicity of the client-server architecture allows clients to make requests that are routed to the appropriate server. These requests are made in the form of transactions. Client transactions are often SQL or PL/SQL procedures and functions that access individual databases and services.

## The Components of the Client–Server Model

The client–server model consists of three parts:

- the client
- the server
- the network

### The Client

The client is the machine (workstation or PC) running the *front–end* applications. It interacts with a user through the keyboard, display, and pointing device such as a mouse. The client also refers to the client process that runs on the client machine.

The client has no direct data access responsibilities. It simply requests processes from the server and displays data managed by the server. Therefore, the client workstation can be optimized for its job. For example, it might not need large disk capacity, or it might benefit from graphic capabilities.

This simple client view is necessary to allow many different client–server implementations, ranging from PCs to mainframes and different client interfaces to store and retrieve data. To support this client environment, Oracle7 fully implements the ANSI/ISO SQL standard and interfaces.

### The Server

The server is the machine that runs Oracle7 software and handles the functions required for concurrent, shared data access. It is often referred to as the *back–end*. Server also refers to the server process that runs on the server machine.

The server receives and processes SQL and PL/SQL statements originating from client applications. The server can also be optimized for its duties. For example, it can have large disk capacity and fast processors. It can also take the load of disk I/O, printing, file transfer, and so on.

### The Network

The network enables remote data access through client–server and server–to–server communication. Oracle's Network Products allow databases and applications to reside on different machines with different operating systems while still communicating as peer applications.

## What is the Client–Server Model?

Certain features are always present in the client–server model.

There is a server process that can process requests from one or more client processes concurrently over a network connection. The client machine provides front–end application software for accessing the data on the server (it may include a graphical interface).

The client initiates transactions, the server processes the transactions (though it can also activate stored procedures), triggers, and stored business rules. Typically, there is a structured query language (for example, SQL) that can be used to access data stored on the server side.

Other aspects of a client–server architecture are:

- the application program interface (API) and how it processes service requests between a client and a server,
- the network communication protocols and facilities that link the client and server, and
- the system hardware and software requirements.

## Benefits of the Client–Server Model

A client–server system is one that uses network resources and shared processing (by both client and server), to provide front–end applications with concurrent, shared data access.

Many clients can share the resources provided by a single server, thus moving the non–critical data and functions to the desktop workstation, leaving the server free for critical processing needs.

Other benefits include:

- Client applications can concentrate on requesting input from users, requesting desired data from the server, and then analyzing and presenting this data using the display capabilities of the client workstation or the terminal (for example, using graphics or spreadsheets).
- Client applications can be designed with no dependence on the physical location of the data. If the data is moved or distributed to other database servers, the application continues to function with little or no modification.
- Oracle7 exploits the multitasking and shared–memory facilities of its underlying operating system. As a result, it delivers the highest possible degree of concurrency, data integrity, and performance to its client applications.
- Client workstations or terminals can be optimized for the presentation of data (for example, by providing graphics and mouse support) and the server can be optimized for the processing and storage of data (for example, by having large amounts of memory and disk space).
- If necessary, Oracle7 can be *scaled* for future growth. As your system grows, you can add multiple servers to distribute the database processing load throughout the network (*horizontal scaling*).

  Alternatively, you can replace Oracle7 on a less powerful computer (such as a microcomputer) with Oracle7 running on a minicomputer or mainframe to take advantage of a larger system's performance (*vertical scaling*). In either case, all data and applications are maintained with little or no modification, since Oracle7 is portable between systems.
- In networked environments, shared data is stored on the servers, rather than on all computers in the system. This makes it easier and more efficient to manage concurrent access.

- In networked environments, inexpensive, low–end client workstations can access the remote data of the server effectively.
- In networked environments, client applications submit database requests to the server using SQL statements. Once received, the SQL statement is processed by the server, and the results are returned to the client application. Network traffic is kept to a minimum because only the requests and the results are shipped over the network.
- A client–server system provides independence between application components and reduced maintenance costs.

## Server–to–Server Communication

Client-server architecture alone does not always meet the needs of a single logical database. The server also requires advanced server–to–server capabilities. These include SQL language requests to Oracle7 Servers and remote procedure calls (RPCs) to Oracle7 Servers.

It must be possible to make changes in the location, number, and function of servers in a complex environment as the information processing needs of the organization change over time. It is only through a complete server–to–server implementation that this can be achieved.

## Oracle7 Server and the Client–Server Model

Oracle Corporation has implemented a server technology in which multiple servers are accessible to clients through the services provided by a single server. Clients can communicate with multiple servers (and gateways) through advanced server–to–server communication, as shown in Figure 1 – 1.

Oracle7 Server provides a client with a single consolidated view of an organization's data and services. An Oracle7 Server client's view of a distributed system is that of a single logical database comprising a distributed database with multiple independent databases.

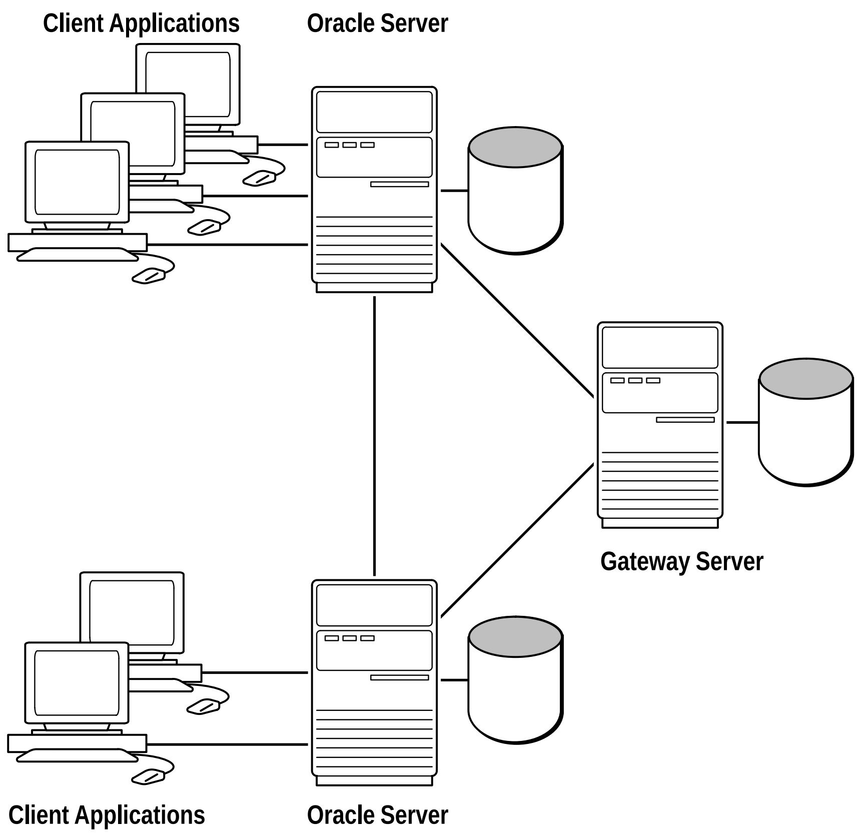


**Figure 1 – 1 Oracle7 Using Dedicated Server Processes**

An essential requirement for a server in a client–server environment is a *symmetric implementation*. This means that any Oracle7 Server must and will support all of the services needed to implement the client's view of a single logical database.

The services implemented by Oracle7 include:

- transaction management
- replication (basic replication, included with the distributed option, and the advanced replication option)
- SQL distributed language requests
- remote procedure calls (RPCs)
- access to non–Oracle data and services through open gateways

## Server Configurations

Oracle7 Server can be configured in three ways, as a:

- dedicated server
- combined user / server process (single–task server)
- Multi–Threaded Server (MTS)

### Dedicated Server

Figure 1 – 2 illustrates Oracle7 running on two computers using the dedicated server architecture.

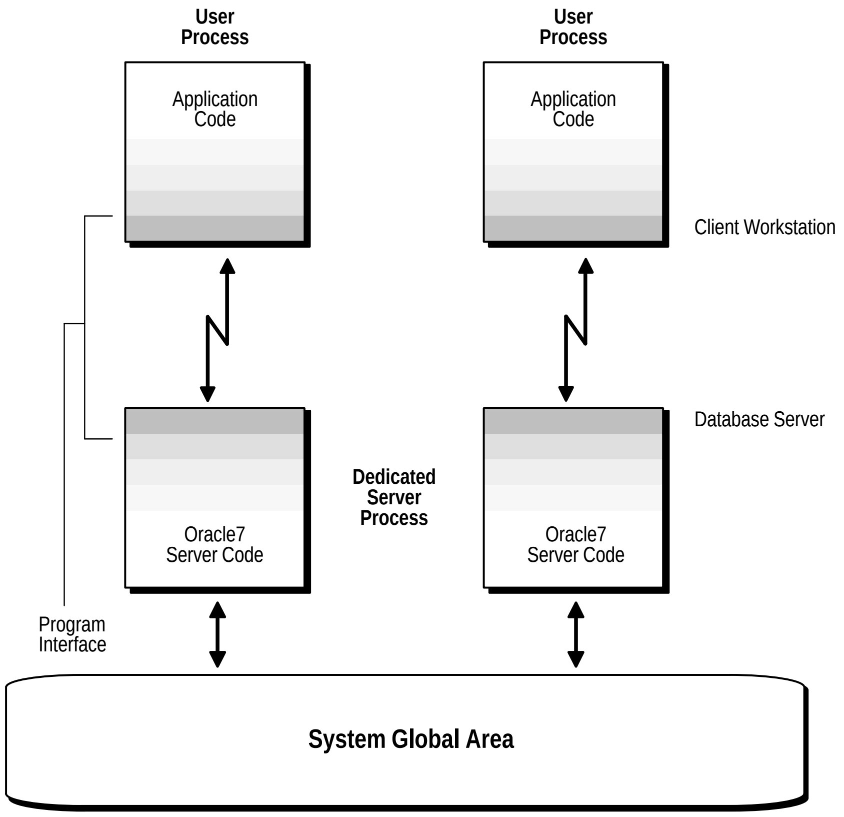


**Figure 1 – 2 Oracle7 Using Dedicated Server Processes**

Notice that, in this type of system, the database application is executed by a user process on one machine, and the associated Oracle7 Server code is executed by a server process on another machine. These two processes are separate, distinct processes.

## Dedicated Server Process

The separate server process created for each user process is called a *dedicated server process* (formerly referred to as "shadow process") because this server process acts only on behalf of the associated user process. In this configuration (sometimes called two–task Oracle7), every user process connected to Oracle7 has a corresponding dedicated server process.

Therefore, there is a one–to–one ratio between the number of user processes and server processes in this configuration. Even when the user is not actively making a database request, the dedicated server process remains (although it is inactive and may be paged out on some operating systems).

The dedicated server architecture of Oracle7 allows client applications being executed on client workstations to communicate with another computer running Oracle7 across a network.

This configuration of Oracle7 is also used if the same computer executes both the client application and Oracle7 Server code and, the host operating system cannot maintain the separation of the two programs if they are run in a single process. A common example of such an operating system is UNIX.

### The Program Interface

The program interface allows communication between the two programs. In the dedicated server configuration, communication between the user and server processes occurs using different mechanisms:

- If the system is configured so that the user process and the dedicated server process are run by the same computer, the program interface uses the host operating system's inter–process communication mechanism to perform its job.
- If the user process and the dedicated server process are executed by different computers, the program interface also encompasses the communication mechanisms, such as the network software and SQL*Net, between the programs.

OSDoc

**Additional Information:** These communications links are operating system– and installation–dependent; see your Oracle operating system–specific documentation and *Understanding SQL*Net* for more information.

**Additional Information:** See "The Program Interface" in Chapter 1 of the *Oracle7 Server Concepts* manual for additional information.

## Combined User/Server Process

Figure 1 – 3 illustrates the combined user / server configuration of Oracle7. Notice that in this configuration, the database application and the Oracle7 Server code all run in the same process, termed a *user* process.

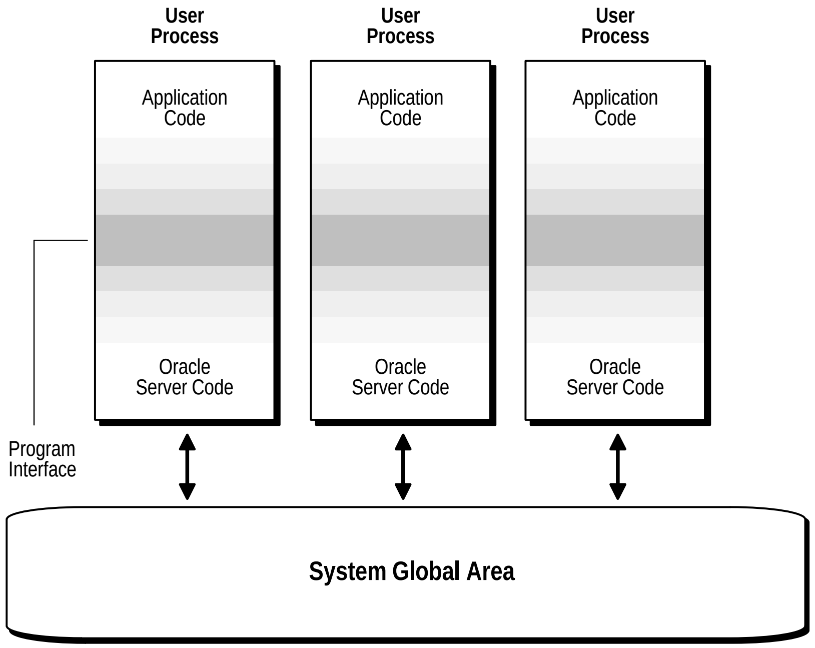


**Figure 1 – 3 Oracle7 Using Combined User/Server Processes**

This configuration of Oracle7 (sometimes called *single–task* Oracle7) is only feasible in operating systems that can maintain a separation between the database application and the Oracle7 code in a single process (such as on the VAX VMS operating system). This separation is required for data integrity and privacy. Some operating systems, such as UNIX, cannot provide this separation, so they must have separate processes run application code and server code to prevent damage to Oracle7 by the application.

**Note:** The program interface is responsible for the separation and protection of Oracle7 Server code and is responsible for passing data between the database application and the Oracle7 user program. For more information about the program interface, see the *Oracle7 Server Concepts* manual.

Only one Oracle7 connection is allowed at any time by a process using the above configuration. However, in a user–written program, it is possible to maintain this type of connection while concurrently connecting to Oracle7 using a network (SQL*Net) interface.

## Multi–Threaded Server

The *multi–threaded server (MTS)* configuration (or “shared server” configuration) allows many user processes to share very few server processes.

In a non–multi–threaded server configuration, each user process requires its own dedicated server process; a new server process is created for each client requesting a connection. A dedicated server process remains associated to the user process for the remainder of the connection.

In a multi–threaded server configuration, client processes connect to a SQL*Net *listener process* which provides the network address of a *dispatcher process* to the client. The client then connects to this dispatcher process. Requests for services from the client are placed in a request queue where they wait for the next available server process. Results are returned to the client by the dispatcher process. In this way, the listener routes client requests to the next available shared server process.

The advantage of the multi–threaded server configuration is that system overhead is reduced, so the number of users that can be supported is increased. For more information about the multi–threaded server and the network listener, see “How SQL*Net Establishes Connections to the Multi–Threaded Server” on page 2 – 24. and “SQL*Net and the Network Listener” on page 2 – 20.

## Contrasting Dedicated Server Processes and Multi–Threaded Server Processes

Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer, and the server process dedicated to the clerk’s user process remains idle. The server process is not needed during most of the transaction, and the system is slower for other clerks entering orders.

The multi–threaded server configuration eliminates the need for a dedicated server process for each connection. A small number of shared server processes can perform the same amount of work, and the memory required for each user is relatively small. Because less memory and process management are required, more users can be supported.

## The Database Server

A database client–server system is a subset of the client–server model. The machine on which the database resides is the database server. Typically, the database also holds stored procedures, event alerts and triggers. It also provides services, such as row–level locking, security, logging, recovery, concurrency management, among others. Other types of servers include file servers, mail servers, and name servers.

The database server allows many users to access data from a single location. However, this architecture can be extended to allow many users to access data from many databases, and it allows those databases to cooperate in maintaining consistency.

Oracle7 Server provides extensive server functionality. The simplicity of the client-server architecture allows clients to make requests that are routed to the appropriate server. These requests are in the form of transactions. Client transactions can be SQL or PL/SQL procedures and functions that access the individual databases and services.

## Front–End Client Applications

A *front–end application* queries a host or server–based database and extracts information for use with report writers, spreadsheets, and so on. It may also provide protocol processing and has access to server–based resources.

The front–end application runs on a workstation and provides a character–based or graphic interface to help the user to access the remote data store(s). These applications can be written in–house or purchased from Oracle Corporation or third–party vendors. Oracle Corporation also provides many tools for the development of these applications.

The function of client–side applications is multi–faceted:

- data query (including parallel query)
- report writing
- third or fourth generation language (3GL/4GL) applications
- transaction processing
- application development
- CASE
- decision support

## Back–End Services

Besides acting as a database server, a server can provide other multi–user service applications for mail, document management, and so on.

### SQL

The most basic tool for data access is the structured query language (SQL). Many SQL and SQL–generating tools have been developed for accessing remote data. Although the front–end application presents a much simplified method of creating queries for the user, the result of the query that is passed to the server is SQL.

### Remote Procedure Calls

Similar to local procedure calls where frequently used code is stored as a procedure or routine, remote procedure calls (RPCs) allow the same functionality over a network of systems. When a program executes the stored code, it passes on any necessary parameters in the call.

Because the procedure is located remotely, however, it must be coded to understand what to do at the remote location, and how to return the results to the requesting application.

## Network Issues in a Client–Server System

Although a client–server system can consist of a server process and client process that exist on the same machine, this book assumes that the typical client–server system consists of a client machine and a server machine.

Client–server systems cannot function without communication. Designing any client–server system requires a knowledge of and an ability to implement appropriate networking using SQL*Net, Oracle Names and other network products.

Other communications software may be supplied by Oracle Corporation but is often purchased separately from the hardware vendor or a third–party software supplier.

The simplest network connection, single client to single server, requires only a single protocol, usually provided by the host operating system. DECnet, TCP / IP, LU6.2, and ASYNC are examples.

However, most companies require connections from many types of machines using different operating systems and communication protocols. Such systems grow quickly in complexity.

SQL*Net uses the communication protocols or application program interfaces (APIs) supported by a wide range of networks to provide for a distributed Oracle7 system. A *communications protocol* is a set of standards, implemented in software, that govern the transmission of data across a network. An *API* is a set of program functions or calls that allow an application to make use of, or communicate with, an underlying program or system.. In the case of networks, the API provides the means to establish remote process–to–process communication over a communication protocol

Communication protocols define the way a network transmits and receives data. In a networked environment, Oracle7 Server communicates with client workstations and other Oracle7 Servers using SQL*Net. SQL*Net supports communications on all major network protocols, ranging from those supported by PC LANs, to those used by the largest mainframe computer systems.

Without SQL*Net, an application developer must manually code all communications in an application that operates in a networked, distributed processing environment. If the network hardware, topology, or protocol changes, the application has to be modified accordingly.

However, by using SQL*Net, the application developer does not have to be concerned with supporting network communications in a database application. If the underlying protocol changes, the database administrator makes some minor changes, while the application continues to function with no modification.

## How SQL*Net Works

SQL*Net drivers provide an interface between Oracle7 processes running on the database server and the user processes of Oracle7 tools running on other computers of the network.

The SQL*Net drivers take SQL statements from the interface of the Oracle7 tools and package them for transmission to Oracle7 over one of the supported industry–standard, higher–level protocols or programmatic interfaces. The drivers also take replies from Oracle7 and package them for transmission to the tools over the same higher–level communications mechanism. This is all done independently of the network operating system.

OSDoc

**Additional Information:** Depending on your operating system, the SQL*Net software may include the driver software and start an additional Oracle7 background process. See your Oracle operating system–specific documentation for details.

### Database Links

For client machines to access a remote database on the server, the communications protocol must be told where the database resides through a database link. A *database link* is a string that uniquely identifies to the communication software the location and name of the remote database.

**Note:** Oracle7 requires the database link name to be the same as the global database name (or service name). For more information on naming issues in a distributed system, see page 2 – 26.

### Location Transparency

Oracle7 server provides the means to make data objects such as tables in remote databases look like they are in the local database to an application developer or user of that data object. Once a client–server system has been set up, users should be able to access the remote database with complete location transparency, provided the database link has been properly defined. For more information on location transparency, see page 2 – 6.

## Introduction to Distributed Systems

An Oracle7 distributed system can be a blend of distributed database and distributed processing systems.

### Distributed Processing and Distributed Databases

Distributed processing and distributed databases are not the same thing, although they have similarities. In a distributed processing system, processing data (searching for information, storing results) is distributed. In a distributed database, the data is distributed in databases on more than one machine.

### Distributed Database System Basics

A distributed database system appears to a user as a single server but is, in fact, a set of two or more servers. The data on each server can be simultaneously accessed and modified via a network. Each server in the distributed system is controlled by its local database administrator (DBA), and each server cooperates to maintain the consistency of the global database.

Note that in Figure 1 – 4, the workstations are the clients and connect to the database servers over the communications network. The two servers, HQ and SALES also communicate over the network to maintain data consistency, as changes to the SALES database may have impact on the HQ database as when data replication has been implemented.

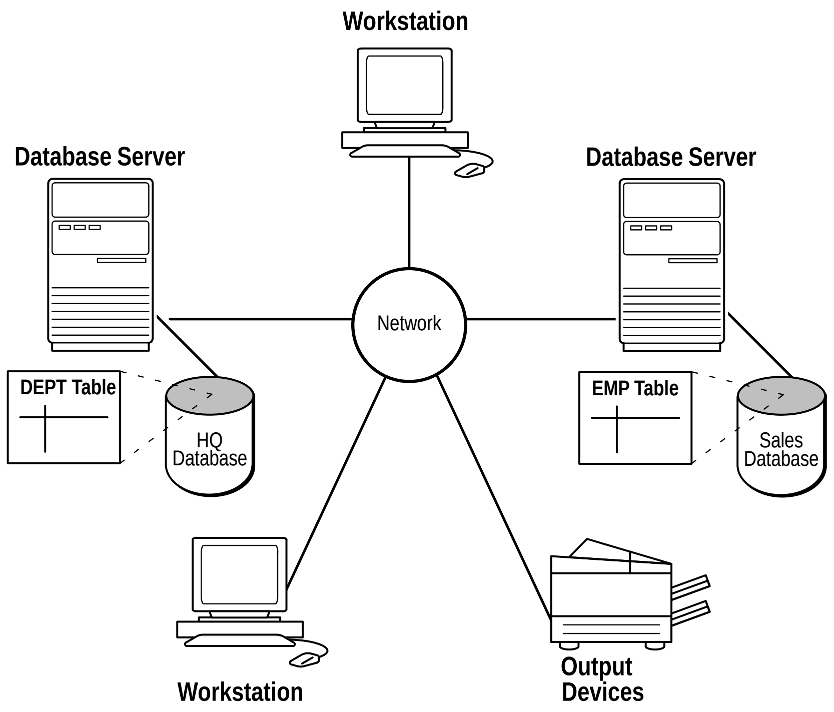
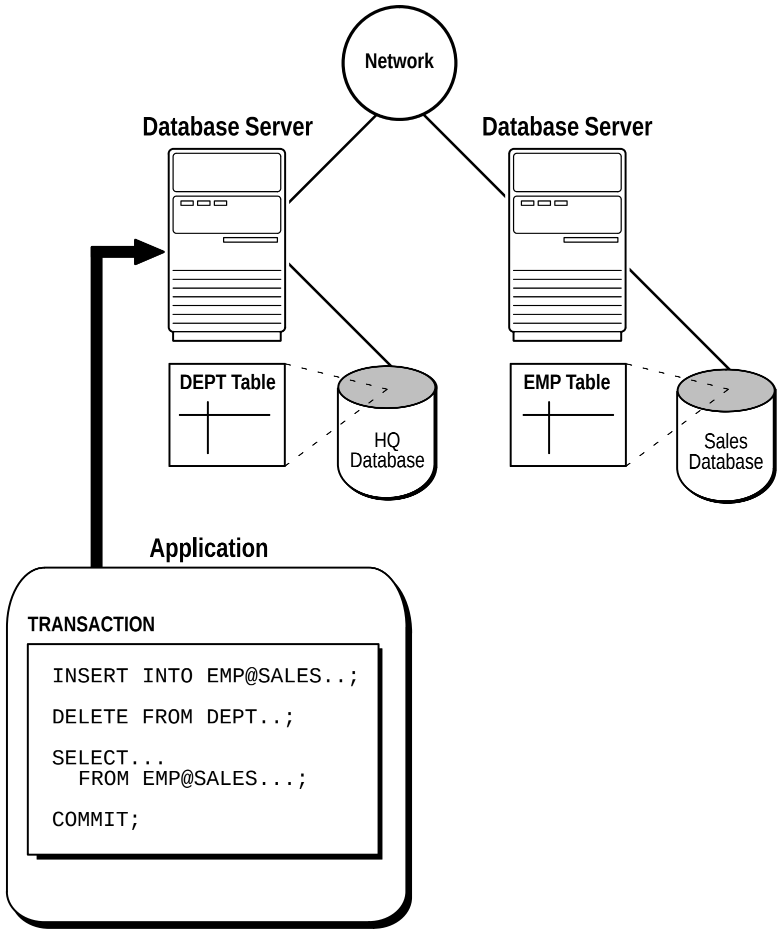


**Figure 1 – 4  A Distributed Environment**

Figure 1 – 5 illustrates how the HQ and SALES database servers work together.



```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

**Figure 1 – 5 An Example of Cooperative Server Architecture**

Note also that the INSERT statement includes the location of the database to be accessed (SALES) because the client from which the SQL statement is issued is connected directly to the HQ database server only.

## Concepts and Terminology

The following sections outline some of the general terminology and concepts used to discuss distributed systems.

### Nodes

A node in a distributed system can be a client, a server, or both. Every computer in a system is a node.

For example, in Figure 1 – 5, the HQ node acts as a server when the DELETE statement is issued against the table DEPT.

It acts as a client when it issues the INSERT and SELECT statements against remote data in the table EMP which resides in the SALES database.

### Replication

The ability to insure reliable data replication is an extremely important (and potentially complex) factor in a distributed system. Data replication means that any given data object can have several stored representatives at several different sites and that, if each representative is potentially updatable, there must be a mechanism for insuring that all representatives reflect the changes.

Oracle7 Server provides a variety of mechanisms for replicating your data. The methods you select will depend on your specific needs:

- read–only snapshots
- Oracle's symmetric replication facility
  - updatable snapshots
  - N–way master replication

If you simply need to view the data at multiple sites, without updating it, you might choose to use read–only snapshots. If you need to be able to update multiple copies of the same data, you will need to use Oracle's symmetric replication facility.

See page 5 – 2 for a brief introduction to replication and how it fits into the scheme of a distributed system.

*Oracle7 Server Distributed Systems, Volume II* provides an introduction to the Oracle7 Server replication capabilities and detailed instructions on how to implement and maintain replication for your system.

## Direct and Indirect Connections

A client can connect directly or indirectly to a server. In Figure 1 – 5, when the client application issues the first and third statements for each transaction, the client is connected directly to the intermediate HQ database and indirectly to the SALES database that contains the remote data.

## Site Autonomy

Site autonomy means that each server participating in a distributed system is administered independently (for security and backup operations) from the other servers. Although all the servers can work together, they are distinct, separate repositories of data and are administered individually. Some of the benefits of site autonomy are:

- Nodes of the system can mirror the logical organization of companies or cooperating organizations that need to maintain a mutually autonomous relationship.
- Local data is controlled by the local administrator. Therefore, each administrator's *region* of responsibility is smaller and more manageable. A region is a group of global object names administered by a single entity, which could be one person or a group of people.
- Independent failures are less likely to disrupt other nodes of the distributed system. The global database is partially available as long as one database and the network are available. No single database failure need halt all global operations or be a performance bottleneck.
- Failure recovery is usually performed on an individual node basis.
- A data dictionary exists for each local database.
- Nodes can upgrade software independently.

## Name Resolution

A schema object (for example, a table) is accessible from all nodes that form a distributed system. Therefore, just as a non–networked local database architecture must provide an unambiguous naming scheme to distinctly reference objects within the local database, so must a distributed system use a naming scheme to ensure that objects throughout the system can be uniquely identified and referenced.

To resolve references to objects (a process called *name resolution*) within a single database, the database usually forms object names using a hierarchical approach. For example, within a single database, each schema has a unique name, and that within a schema, each object has a unique name.

Because uniqueness is enforced at each level of the hierarchical structure, an object's local name is guaranteed to be unique within the database, and references to the object's local name can be easily resolved.

Distributed systems simply extend the hierarchical naming model by enforcing unique database names within a network. As a result, an object's *global object name* is guaranteed to be unique within the distributed system, and references to the object's global object name can be resolved among the nodes of the system.

See page 2 – 26 for more information on global naming issues.

## Remote/Distributed Queries and Updates

A *remote query* is a query that selects information from one or more remote tables, all of which reside at the same remote node.

A *remote update* is an update that modifies data in one or more tables, all of which are located at the same remote node.

> **Note:** A remote update may include a subquery that retrieves data from one or more remote nodes. Because the update is performed at only a single remote node, however, the statement is classified as a remote update.

A *distributed query* retrieves information from two or more nodes.

A *distributed update* modifies data on two or more nodes. A distributed update is possible using a procedure or trigger, which includes two or more remote updates that access data on different nodes. Statements in the program unit are sent to the remote nodes, and the execution of the program succeeds or fails as a unit.

## Remote and Distributed Transactions

A *remote transaction* is a transaction that contains one or more remote statements, all of which reference the same remote node. A *distributed transaction* is any transaction that includes one or more statements that, individually or as a group, update or query data on two or more distinct nodes of a distributed system. If all statements of a transaction reference only a single remote node, the transaction is remote, not distributed.

## Remote Procedure Calls (RPCs)

Oracle7 supports RPCs with full PL/SQL datatypes as parameters and return values. Because PL/SQL datatypes are a superset of SQL datatypes, PL/SQL procedures and functions are ideal for managing Oracle7 services on remote servers. All parameters and return values provided by a remote server can be examined and stored by the calling server. This allows the local server to maintain the client's calling interface and semantics – even if the service is reimplemented, or the remote site decides to change the RPC's interface.

## Access to Non–Oracle Data and Services through Oracle Open Gateway Technology

Oracle Open Gateway Technology provides access to non–Oracle data through gateway servers, which are a part of a distributed system like any other distributed server. Open Gateway technology is tightly integrated with the Oracle7 Server. This permits integration of both SQL and non–SQL data and services. For more information, see your Oracle Open Gateway documentation.

Gateway servers access the target system directly. Oracle7 client applications do not connect directly to a gateway server, but indirectly by first connecting to an integrating server. An integrating server communicates with a gateway server in the normal Oracle7 server–to–server manner using SQL*Net. See Figure 1 – 1.

A gateway server is a single process and does not start background processes. On some platforms, such as MVS, the gateway server starts once, and maintains multiple user sessions in memory, where one session handles the requests from a single Oracle7 client application. On other platforms, such as UNIX platforms, a gateway server starts for each user session.

### Transparent Gateway Server

A *transparent gateway server* emulates an Oracle7 Server and usually resides in the target system environment. The database administrator creates database links and local synonyms at all Oracle7 Servers that require access to the data source. The gateway server is then transparent for Oracle7 client applications that access what appear to be Oracle7 tables or views.

A client application connects directly to an integrating Oracle7 Server, which is responsible for connecting to the transparent gateway server. A transparent gateway does not execute PL/SQL stored procedures, but a stored procedure on an Oracle7 Server can issue SQL statements that access the data source via the gateway.

An Oracle7 client application queries and modifies a data source using ANSI/ISO SQL via the transparent gateway. Transparent gateways always perform automatic data conversion. In some cases, this is driven by gateway data definition language (GDDL) for non–SQL data sources.

A target system is unlikely to have all Oracle7 functionality. For queries, missing functionality is often fulfilled by the gateway server or integrating server. For example, where a target system has a limited capability to conditionally retrieve data, the gateway can make up for this missing functionality by means of a post–filter.

#### Procedural Gateway Server

A *procedural gateway server* emulates Oracle7 Server's remote procedural capabilities and usually resides in the target system environment. The database administrator creates database links and local synonyms at all Oracle7 Servers that require access to the data source. The gateway server is then transparent for Oracle7 client applications, which access what appear to be Oracle7 PL/SQL stored procedures.

A client application connects directly to an integrating Oracle7 Server, which is responsible for connecting to the procedural gateway server.

A procedural gateway does not execute SQL requests. An Oracle7 client application executes calls at a target system using PL/SQL remote procedure calls. Automatic data conversion to and from the datatypes of the arguments in the procedure call and the call at the target system is driven by gateway data definition language.

## Transaction Recovery Management

An efficient system, distributed or non–distributed, must guarantee that all statements in a transaction are either committed or rolled back as a unit, so that the data in the logical database can be kept consistent. The effects of a transaction should be either visible or invisible to all other transactions at all nodes. This should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a non–distributed system are discussed in Chapter 5. In a distributed system, Oracle7 must coordinate transaction control over a network and maintain data consistency, even if a network or system failure occurs.

*Transaction recovery management* guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. Transaction recovery management also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers. Transaction recovery management is described in Chapter 5.

## Transparency

The functionality of a distributed system must be provided in such a manner that the complexities of the system are transparent to both users and administrators.

For example, a distributed system should provide methods to hide the physical location of objects throughout the system from applications and users. *Location transparency* exists if a user can refer to the same table the same way, regardless of the node to which the user connects. Location transparency is beneficial for the following reasons:

- Access to remote data is simplified because the users do not need to know the location of objects.
- Objects can be moved with no impact on end–users or applications.

A distributed system should also provide query, update, and transaction transparency. For example, standard SQL commands, such as SELECT, INSERT, UPDATE and DELETE, should allow users to access remote data without the requirement for any programming. Transaction transparency occurs when the DBMS provides the functionality described below using standard SQL COMMIT, SAVEPOINT, and ROLLBACK commands, without requiring complex programming or other special operations to provide distributed transaction control:

- The statements in a single transaction can reference any number of local or remote tables.
- The database server guarantees that all nodes involved in a distributed transaction take the same action. They either all commit or all roll back the transaction.
- If a network or system failure occurs during the commit of a distributed transaction, the transaction is automatically and transparently resolved globally. That is, when the network or system is restored, the nodes either all commit or all roll back the transaction.

A distributed architecture should also provide facilities to transparently replicate data among the nodes of the system. Maintaining copies of a table across the databases in a distributed system is often desired so that:

- Tables with high–query and low–update activity can be accessed faster by local user sessions because no network communication is necessary.
- If a database that contains a critical table experiences a prolonged failure, replicates (or copies) of the table in other databases can still be accessed.

A database server that manages a distributed system should make table replication transparent to users working with the replicated tables.

Finally, the functional transparencies explained above are not sufficient alone. The distributed system must also perform with acceptable speed.

**National Language Support (NLS)**

Oracle7 supports heterogeneous client/server environments where clients and servers use different character sets. The character set used by a client is defined by the value of the NLS_LANG parameter for the client session. The character set used by a server is its database character set. Data conversion is done automatically between these character sets if they are different. For more information about National Language Support features, see the *Oracle7 Server Reference.*

## SQL Statement Execution in a Distributed System

SQL statement execution of remote and distributed statements is essentially the same in a distributed system as in a non–distributed database. However, depending on the type of statement issued, the location of statement execution can vary.

### Remote Queries and Updates

All remote queries and remote updates are sent to the remote node for statement execution. The remote node executes the statement and returns any results back to the local node, which returns them to the user or application.

### Distributed Queries and Distributed Updates

The statements in a procedure or trigger that constitute a distributed update are sent individually to the correct remote node for execution. Results are returned to the local node and then to the calling user or application.

### Values for Environmentally–Dependent SQL Functions

In a distributed system, environmentally–dependent SQL functions, such as SYSDATE, USER, UID, and USERENV, are evaluated always with respect to the local node regardless of where the statement (or portion of a statement) is executed. The USERENV function is supported only for queries.

### Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, the referenced objects must match, and the bind types of any bind variables must be the same. If available, shared SQL areas can be used for the local and remote handling of any statement (or decomposed query).

### Optimization of SQL Statements in a Distributed System

The optimization approach available for SQL statements can vary depending on the type and complexity of the statement.

CHAPTER

*2*

# Network Administration

This chapter introduces networking concepts and terminology that anyone planning to implement a distributed system must be familiar with. It also discusses the tasks that network administrators must perform to set up client–server systems and integrate Oracle's network products into a distributed system.

For related information of interest to DBAs, see Chapter 4, "Database Administration". Chapter 3, "Network Administration Tools", provides an introduction to Oracle's network administration tools.

It is assumed that you have read and are familiar with the concepts and terms in Chapter 1, "Understanding Distributed Systems", and in *Understanding SQL*Net*.

**Note:** This chapter is not intended to be a step–by–step guide to implementing networked distributed systems. References in most sections are provided to the relevant Oracle Network Products and Oracle7 Server documentation which does contain step–by–step instructions and detailed reference material.

## Oracle's Network Products

Oracle's Network Products provide all the tools necessary and an ideal architecture with which to implement client–server, server–to–server, and distributed systems. This section describes the Oracle's Network Products and how they work together to provide the basis for distributed systems.

### Oracle's Network Products Architecture

Oracle's Network Products architecture enables integration with most network services, including the transport, naming, and security services provided at the network level. User applications are insulated from these encapsulated services so programmers need not write special code to take advantage of them.

This architecture insures that user code developed in one network environment can be redeployed in a different network environment, without the need for changes. This flexibility provides you with the freedom to adopt any type of new computing or networking platform (or integrate existing systems) without regard to connectivity issues at the user or application level.

### SQL*Net

SQL*Net uses the communication protocols or application program interfaces (APIs) supported by a wide range of networks to provide for a distributed Oracle7 system. A *communications protocol* is a set of standards, implemented in software, that govern the transmission of data across a network. An *API* is a set of program functions or calls that allow an application to make use of, or communicate with, an underlying program or system. In the case of networks, the API provides the means to establish remote process–to–process communication over a communication protocol

SQL*Net's networking products transparently integrate disparate clients, servers, and gateways into a unified information resource, using any combination of industry–standard or proprietary network protocols.

**Additional Information:** See *Understanding SQL*Net* for more detailed conceptual information about SQL*Net.

### MultiProtocol Interchange

A *MultiProtocol Interchange* is a node that acts as a translator of communication protocols using protocol adaptors. SQL*Net version 2 works with the MultiProtocol Interchange to allow clients and servers to communicate transparently across networks running dissimilar protocols. A single node anywhere on the network that is loaded with two or more protocol stacks and a MultiProtocol Interchange enables all nodes on the attached networks to transparently connect to services on the other side of the Interchange. The MultiProtocol Interchange is discussed in more detail on page 2 – 8.

SQL*Net's layered architecture, shown in Figure 2 – 1, allows standard applications to run transparently over any type of network by simply using the appropriate Oracle Protocol Adapter.

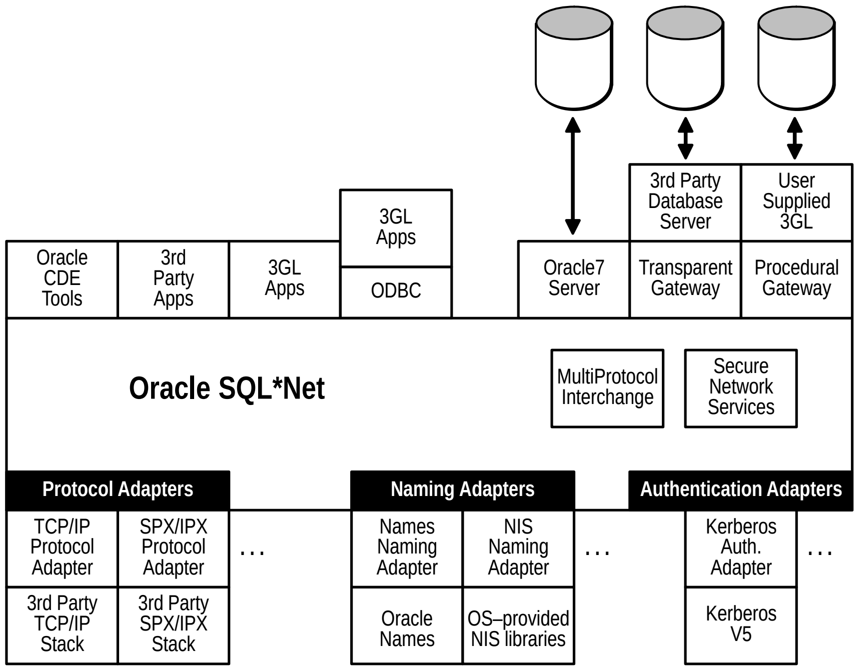


**Figure 2 – 1 Oracle's Network Products Architecture**

## Other Services

Oracle's Network Products include several services that are critical for managing large–scale distributed environments, such as an enterprise–wide distributed naming service—Oracle Names.

Also provided are Native Naming Adapters for use with existing name services such as NIS (Network Information Services), DCE's CDS (Distributed Computing Environment's Cell Directory Service), Novell's NDS (NetWare Directory Services), and Banyan's StreetTalk.

These naming adapters enable customers to integrate Oracle into their existing naming environment, while preserving their existing network infrastructure.

A centralized configuration management facility—Oracle Network Manager—provides both a topological and hierarchical view of the network, enabling administrators to easily configure services, such as databases and name servers in the network.

Where network security is required, the optional Secure Network Services product adds full data–stream encryption and integrity checking to SQL*Net. Included with Secure Network Services version 2.0 are authentication adapters such as Kerberos V5, which allow users who have the appropriate credentials to automatically and securely access any Oracle application or server without specifying a user name or password.

## Oracle Network Products and Distributed Systems

In today's highly distributed information systems, networking is one of the most important architectural components. This section describes some of the ways that Oracle network products can be implemented to create an efficient distributed system.

### Transparency in a Distributed System

The functionality of a distributed system must be provided in such a manner that the underlying complexities of the system are transparent to both users and administrators. A network implementation in a distributed system must be able to deliver:

- network transparency
- location transparency
- application transparency

#### Network Transparency

SQL*Net's layered architecture allows standard applications to run transparently over many types of protocols by simply using the appropriate Oracle Protocol Adapter. Thus, Oracle applications developed with a local database can be distributed across a network to access the same, or a similarly formatted, Oracle database with no changes to the application.

SQL*Net is responsible for forwarding application requests for data from an Oracle client or server to a server and returning the results to the initiator of the query. From the application developer's or application user's perspective, all data interaction using SQL*Net is invisible to the application and user.

Additionally, it is possible to change the network structure beneath the application without changing the application. This capability is known as *network transparency.*

Figure 2 – 2 illustrates a possible distributed systems scenario.

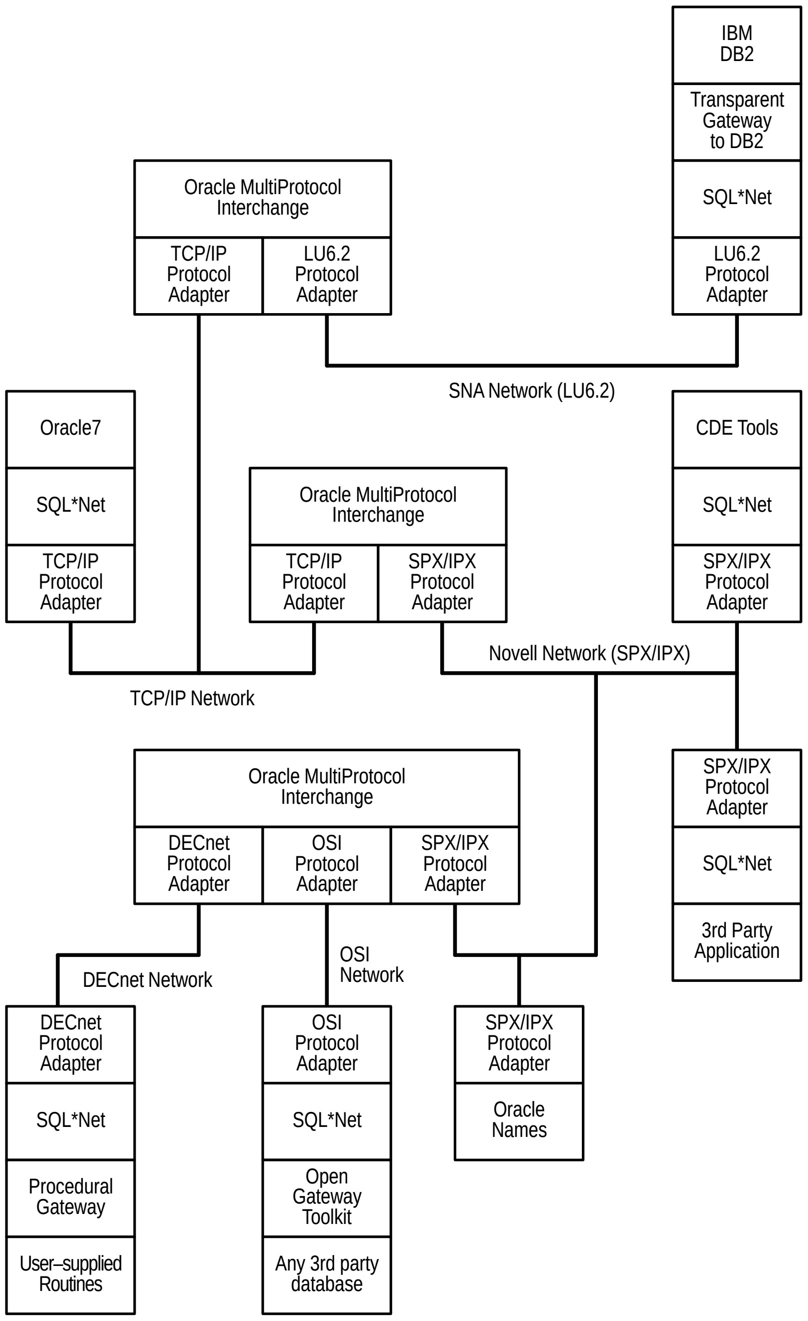


**Figure 2 – 2  Possible Distributed Systems Scenario**

### Location Transparency: Database Links, Synonyms, and Service Names

Oracle7 server provides the means to make data objects such as tables in remote databases look to an application developer or user of that data object like they are in the local database. This is called *location transparency*. The *database link* and *synonym* allow the database in which they are created to identify a remote data object and make the location transparent. Establishing and maintaining location transparency is a joint function of the network administrator and the DBA. The network administrator is typically responsible for database links which are discussed later in this chapter, while views, synonyms and other forms of transparency are the responsibility of the DBA. They are discussed in Chapter 4, "Database Administration".

Location transparency removes all need for references to the location of the data from applications when the synonym is used. Should the location of the remote table be moved to another machine or database, only the synonym and database link need be updated to reference the new location. No changes to applications are required.

If the database link connection is specified as a service name (or symbolic name) in the network configuration file (TNSNAMES.ORA), the database links accessing the data do not have to be changed if the remote database is moved. The only update required is to the TNSNAMES.ORA file. Similarly, if Oracle Names is used, only the central network definition needs to be changed.

TNSNAMES.ORA, and other configuration files are created using Oracle Network Manager. The configuration files are described in Appendix A of *Understanding SQL*Net*. For detailed instructions on creating the network definition and the configuration files for SQL*Net and other Oracle networking products, see the *Oracle Network Manager Administrator's Guide*.

## Application Transparency

Another benefit of SQL*Net's encapsulated network architecture is that client–server applications can be built on one class of system and deployed on another. An application system back–end developed on a PC server can, for example, be redeployed on a large minicomputer server without the end–users needing to know that the location of the application has changed.

This flexibility means that server systems can be selected for current requirements, rather than for some large future need. Likewise, a system developed on a mainframe can be moved down to smaller, more cost–effective servers without reworking any of the code. This flexibility also allows exactly the same application code that is running on a large, central processor complex to be run on small, workgroup servers on remote PC LANs.

## SQL*Net and Network Environment Independence

Just as SQL*Net connects clients and servers that operate on different nodes on a network, it also connects database servers across networks to facilitate distributed transactions. For example, when an application requests data from a remote database, the local database server communicates with the remote database through the network, using network communications software and Oracle's SQL*Net.

SQL*Net's advantage is that it runs on most networks. The particular type of network protocol, brand, or topology does not matter. In fact, it is feasible for a distributed system implemented using SQL*Net to work over different types of communication networks simultaneously.

### Media/Topology Independence

SQL*Net supports most third–party network software packages, which in turn, support a wide variety of network hardware devices. On some platforms, a single Oracle Protocol Adapter can operate on hundreds of different network interface cards. This compatibility allows you to deploy applications in virtually any network environment, including Ethernet, Token–Ring, FDDI (Fiber Distributed Data Interface), ATM (Asynchronous Transfer Mode), wireless, and others.

When a request for a connection is made successfully, SQL*Net passes control of the connection to the underlying protocol. At that point, all media and / or topologies supported by the underlying network protocol are indirectly inherited by SQL*Net. SQL*Net allows the network protocol to use any means of data transmission, such as Ethernet, Token Ring, FDDI, or SDLC, to accomplish the low–level data link transmission between the two computers.

In addition, because SQL*Net connects to the network infrastructure through standard, high–level protocols, it also works with network components at lower levels such as bridges, routers, gateways, and packet switches.

### Protocol Independence

SQL*Net provides protocol independence to its applications. Any application built on any computer running any protocol can be distributed without change to other computers running other protocols. An application using SQL*Net can run over any network protocol. SQL*Net's architecture provides the industry's broadest support for network transport protocols, including TCP / IP (Transmission Control Protocol / Internet Protocol), Novell SPX / IPX (Sequenced Packet Exchange / Internet Packet Exchange), IBM LU6.2, DECnet, OSI, and others.

In addition to supporting different protocols, SQL*Net also supports many vendor's protocol stacks, eliminating the need to purchase and install additional protocol support hardware or software. Without changing your existing infrastructure, you can transparently connect any combination of PC, UNIX, legacy, and other systems, using the network software you already have.

Any connection that works reliably at the protocol level will work with, and be transparent to SQL*Net, regardless of the number of physical connections and transformations the packets go through between the two machines.

When connectivity is required between different high–level protocols, such as from SPX / IPX to TCP / IP, the Oracle MultiProtocol Interchange can provide automatic protocol conversion, a task that cannot be performed at lower levels in the network stack. This means that networks running different protocols can communicate with each other.

## The MultiProtocol Interchange

Oracle's client–server and server–to–server models provide connectivity across multiple network protocols.

### Data Access Across Transfer Protocols

Oracle's MultiProtocol Interchange can be used with SQL*Net to enable transparent data access across protocols, allowing a client using one protocol to communicate with a server using a different protocol. This way, clients and servers running different network protocols can communicate using only their native protocols. This eliminates the need to purchase and maintain multiple protocol stacks.

All of the advanced Oracle7 capabilities, such as basic replication, the advanced replication option, stored procedure calls, and automatic transaction recovery mechanisms, can operate transparently across any number of protocol boundaries.

Multiple Interchanges can be combined to provide multistage protocol conversion, all transparent to programmers and users alike. Applications simply ask for services by name, and SQL*Net automatically calculates the most efficient route to take through the network and establishes the connection.

When the network topology changes, such as when a new server is added to the network, users and applications are completely unaware of the change, because SQL*Net transparently calculates a new route automatically at request time. Where there are multiple possible routes, SQL*Net will use the most efficient route based on high–level weighting provided by the network administrator.

This product is described in detail in the *Oracle MultiProtocol Interchange Administrator's Guide*. For information on configuring the MultiProtocol Interchange, see the *Oracle Network Manager Administrator's Guide.*

## MultiProtocol Interchanges in the Client–Server Configuration

With SQL*Net version 2, the client and server can belong to different communities connected by one or more MultiProtocol Interchange(s). A *community* is a group of computers that can communicate using the *same transport level protocol*, such as TCP / IP. That is, computers that use the same protocol are members of the same community.

For example, a MultiProtocol Interchange can be installed on a node that is loaded with two protocol stacks, TCP / IP and DECnet. Then, it can enable a network running TCP / IP to communicate with a network running DECnet. The result is a higher–level application network in which any two applications can communicate. Using an Interchange as an intermediary, applications on the client and server machines can communicate even though they are using different protocols. Any data exchanged in the client–server applications is forwarded through the Interchanges along the path.

Figure 2 – 3 shows a connection between a client (protocol A) and a server (protocol B) in adjacent communities. A MultiProtocol Interchange joins the two networks. SQL*Net and an Oracle Protocol Adapter specific to Protocol A are installed on the client while SQL*Net and an Oracle Protocol Adapter specific to Protocol B are installed on the database server. The Interchange has adapters for both Protocol A and Protocol B. In a sense, it is bilingual (or poly–lingual). When communication is requested between the two communities, the MultiProtocol Interchange translates between the two protocols.

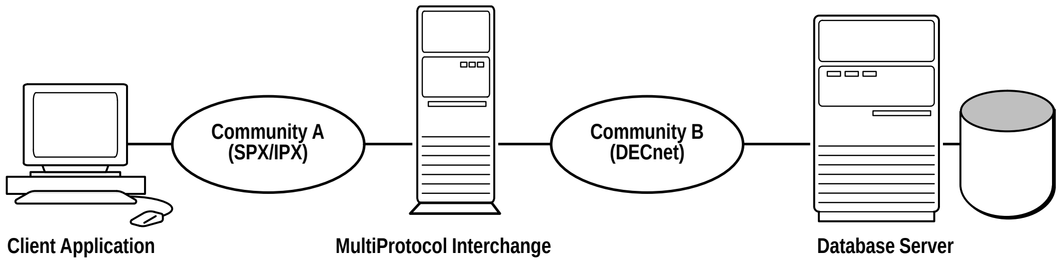


**Figure 2 – 3  Heterogeneous Networking with a Client–Server Connection**

## MultiProtocol Interchanges in the Server–to–Server Configuration

In a server–to–server configuration, this same heterogeneous network capability is extended to include database–to–database communications. Two types of server–to–server connections are possible using SQL*Net:

- a direct connection between two servers in the *same* community
- a connection between servers in *different* communities through one or more MultiProtocol Interchanges

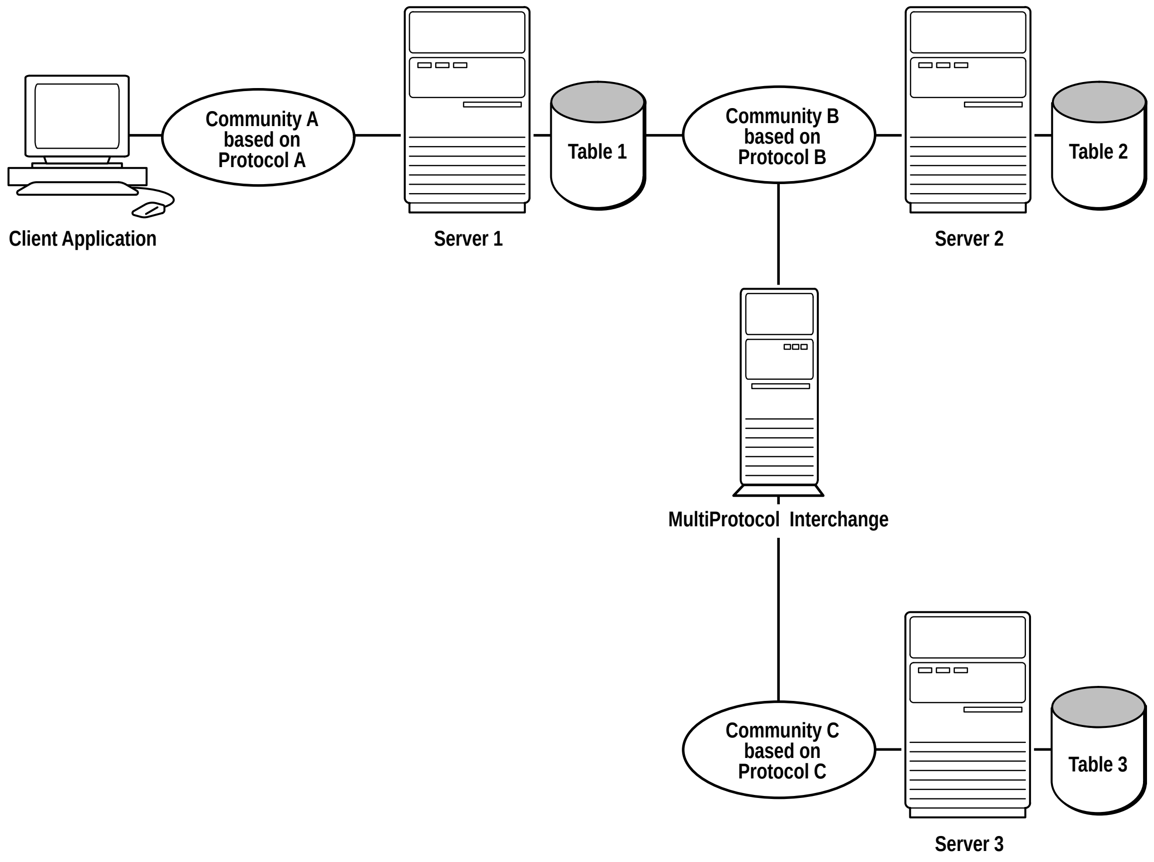

Community A based on Protocol A
Client Application
Server 1
Table 1
Community B based on Protocol B
Server 2
Table 2
MultiProtocol Interchange
Community C based on Protocol C
Server 3
Table 3

**Figure 2 – 4 Heterogeneous Networking in a Distributed Database Transaction**

The example in Figure 2 – 4 shows both types of connections.

In this example, Server 1 is a member of two communities, Community A and Community B. A client application in Community A accesses the database server (Server 1) within the same community. Server 1 determines that the transaction must be distributed further to retrieve data from tables in Server 2 and Server 3. Server 1 initiates a connection to Server 2 in Community B to which Server 1 also belongs. Server 1 also initiates a connection to Server 3 through the MultiProtocol Interchange installed between Community B and Community C.

Server 1 does not have to use an MultiProtocol Interchange to initiate an additional request for data from Community B since it belongs to both Community A and Community B, but it must use an Interchange to access a server in Community C.

Note that using the Interchange imposes no new restrictions on a SQL*Net connection. If used in a client–server connection, clients have a standard peer–to–peer connection between the client and server although they are in different communities.

Similarly, if a server initiates a connection with another server through an Interchange using a database link, the standard database link restrictions apply.

## SQL*Net Version 2 Architecture

This section provides a more detailed discussion of SQL*Net and the role it plays in distributed systems.

SQL*Net version 2 uses the Transparent Network Substrate (TNS) and industry–standard network protocols to connect a client to a server and establish an Oracle session.

The next few sections describes the following architectural concepts:

- Transparent Network Substrate
- SQL*Net's communication role
- Distributed processing with SQL*Net
- SQL*Net operations
- SQL*Net and the network listener

## Transparent Network Substrate (TNS)

Forming the basis for Oracle networking products, the Transparent Network Substrate (TNS) enables Oracle to provide a network of applications above all existing networks of computers. With TNS, peer–to–peer application connectivity is possible where no direct machine–level connectivity exists. *Peer–to–peer* is an architecture in which two or more nodes can communicate with each other directly, without the need for any intermediary devices. In a peer–to–peer system, a node can be both a client and a server.

TNS provides two key features to a TNS–based network product and, in turn, any application built using TNS:

- a single, common interface to all industry–standard protocols
- the ability to connect to applications in physically separate networks through one or more MultiProtocol Interchanges

TNS is the foundation component of all current and planned network products from Oracle. Today, TNS networks connect Oracle clients and servers through SQL*Net version 2. In the future, Oracle Corporation will provide additional TNS–based application connectivity tools.

## SQL*Net's Communication Role

In a distributed transaction, SQL*Net is responsible for sending information across various networks on behalf of a client application or database server. In such a configuration, there are commonly two types of computers acting as the client and server. Two–Task Common (see page 2 – 14) ensures that all differences between clients and servers, such as internal datatype representations or NLS character sets, are resolved, allowing the client and server to communicate transparently. SQL*Net relays all communication tasks to TNS through its common entry points. SQL*Net is unaffected by the specific communication mechanism used underneath TNS (for example, TCP / IP, DECnet, shared memory, and so on).

Communication between client and server proceeds in a stack–like fashion with corresponding levels communicating in a peer relationship. The logical exchange unit at each layer of the stack conveys the level of generalization employed at that level. The Oracle client and server exchange SQL statements and data rows. At the UPI / OPI (User / Oracle Program Interface) layers, these exchanges translate into series of calls to SQL routines such as logon, parse, execute, and fetch. The SQL*Net layer handles these calls as a series of Oracle send / receive messages, and TNS in turn processes the packets over the network. The network protocol, not provided by Oracle (typically provided with each particular platform by its vendor), guarantees a reliable means of communication between the two tasks.

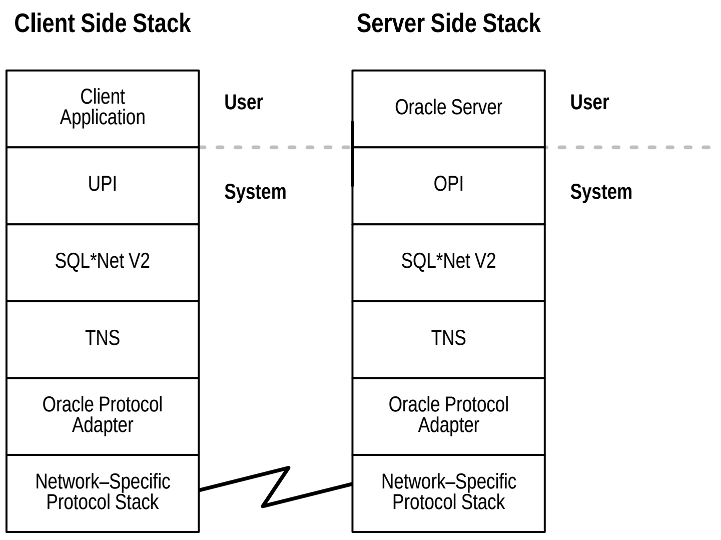


**Figure 2 – 5 Oracle Client–Server Components**

## SQL*Net in Distributed Processing

SQL*Net is responsible for enabling communications between the cooperating partners in a distributed transaction, either client–server or server–to–server. Specifically, SQL*Net enables clients and servers to connect to each other, send data such as SQL statements and data responses, initiate interrupts from the client or server, and disconnect when the session is complete. During the life of the connection, SQL*Net resolves all differences between the internal data representations and / or character sets of the computers being used.

When a client or server makes a connection request, SQL*Net receives the request. If more than one machine is involved, SQL*Net passes the request to the TNS, to be transmitted over the appropriate communications protocol to the appropriate server. On the server, SQL*Net receives the request from TNS and passes it to the database as a network message with one or more parameters (that is, a SQL statement).

Except for the initial connection, the local and remote applications generate the same requests whether they run on the same computer or are distributed across multiple computers. The initial connection differs only in that the application or user must specify the name of the remote database.

## Components Involved in Distributed Processing

Several software components complete a distributed transaction, whether it is a client–server or server–server transaction. Figure 2 – 5 shows the components of a client–server session. These components are described in the following sections.

## Client Side Interaction

The following paragraphs discuss the components of the client–server transaction process, beginning with the client application and concluding with the Oracle Server.

### Client Application

The client application provides all user–oriented activities, such as character or graphical user display, screen control, data presentation, application flow, and other application specifics. The application identifies any SQL database operations to send to the server database and passes them through the User Program Interface (UPI).

### User Program Interface (UPI)

The UPI code that contains all information required to initiate a SQL dialogue between the client and the server. It defines calls to the server to:

- parse SQL statements for syntax validation
- open a cursor for the SQL statement
- bind client application variables into the server shared memory
- describe the contents of the fields being returned based on the values in the server's data dictionary
- execute SQL statements within the cursor memory space
- fetch one or more rows of data into the client application
- close the cursor

The client application uses some combination of these calls to request activity within the server. Often, all UPI calls can be combined into a single message to the server, or they may be processed one at a time through multiple messages to the server, depending on the nature of the client application. Oracle products attempt to minimize the number of messages sent to the server by combining many UPI calls into a single message to the server. When a call is performed, control is passed to SQL*Net to establish the connection or transmit the request to the server.

### Two–Task Common

Two–Task Common provides character set and data type conversion between different character sets or formats between client and server. This layer is optimized to perform conversion only when required on a per connection basis.

At the time of initial connection, SQL*Net version 2 is responsible for evaluating differences in internal data and character set representations and determining whether conversions are required for the two computers to communicate.

### SQL*Net

The role of SQL*Net is to establish and maintain a connection between the client application and the server and exchange messages between them. The network listener receives connection requests for a particular database and passes control to the server.

### Transparent Network Substrate (TNS)

TNS receives requests from network applications, in this case SQL*Net, and settles all generic machine–level connectivity issues, such as:

- the location of the server or destination (open**,** close functions)
- whether one or more MultiProtocol Interchanges will be involved in the connection (open**,** close functions)
- how to handle interrupts between client and server based on the capabilities of each (send, receive functions)

The generic set of TNS functions (open, close, send, receive) passes control to an Oracle Protocol Adapter to make a protocol–specific call.

Additionally, TNS optionally provides encryption and sequenced cryptographic message digests to protect data in transit. See *Secure Network Services Administrator's Guide* for more information.

### Oracle Protocol Adapter

The Oracle Protocol Adapter is responsible for mapping TNS functionality to any industry–standard protocol used in the client–server connection. The adapters are responsible for mapping the equivalent functions between TNS and a specific protocol.

### Network–Specific Protocols

All Oracle software in the client–server connection process requires an existing network protocol stack to make the machine–level connection between the two machines. The network protocol is responsible only for getting the data from the client machine to the server machine, at which point the data is passed to the server–side Oracle Protocol Adapter.

## Server–Side Interaction

Going up the process stack on the server side is the reverse of what occurred on the way down the client side. See the right side of Figure 2 – 5.

The one operation unique to the server side is the act of receiving the initial connection. The server has a process (the network listener) that regularly checks for incoming connections and evaluates their destination.

The network listener is a process on a server that listens for connection requests for one or more databases on one or more protocols. It is discussed in "SQL*Net and the Network Listener" on page 2 – 20. Based on the Oracle Server ID (SID) specified, the connection is passed to the Oracle Server.

The components above SQL*Net, the OPI and the Oracle Server, are different from those on the client side.

Oracle Program Interface (OPI)

The OPI has a complementary function to that of the UPI. It is responsible for responding to each of the possible messages sent by the UPI. For example, a UPI request to fetch 25 rows would have an OPI response to return the 25 rows once they have been fetched.

Oracle Server

The Oracle Server side of the connection is responsible for receiving dialog requests from the client UPI code and resolving SQL statements on behalf of the client application. Once received, a request is processed and the resulting data is passed to the OPI for responses to be formatted and returned to the client application.

## Server–to–Server Interaction

When two servers are communicating to complete a distributed transaction, the process and dialogues are the same as in the client–server scenario, except that there is no client application. See Chapter 5, "Distributed Updates" for more information. The server has its own version of UPI, called NPI. The *NPI* interface can perform all of the functions that the UPI does for clients, allowing a coordinating server to construct SQL requests for additional servers. Figure 2 – 6 shows a server–to–server connection and all associated layers.

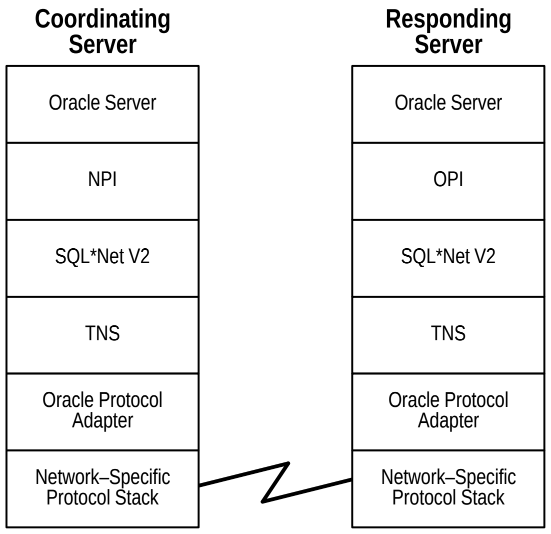


**Figure 2 – 6 Oracle Server–Server Components**

## SQL*Net Operations

SQL*Net provides functions, described in the following sections, that belong to the following classifications:

- connect operations
- data operations
- exception operations

All the functions work with tools and databases that use SQL*Net for distributed processing, although none of them are visible to the user.

**Note:** The information contained in the following summary is for the benefit of the network administrator, who needs to understand what role SQL*Net version 2 plays within
the network.

### Connect Operations

SQL*Net supports two basic connect operations:

- connect to server (open)
- disconnect from server (close)

## Connecting to Servers

The connect operation is initiated during any standard database login between the client application and the server, with information such as the client machine name and user name being passed to the remote machine. This information is required to support externally–identified logins.

A client application initiates a request for a connection to a remote database (or other network service) by providing a short name for its desired destination. That short name, called a *service name,* is mapped to a network address contained in a *connect descriptor* stored in the network configuration file TNSNAMES.ORA or in a database for use by Oracle Names. For more information on service names and connect descriptors, see "Global Naming Issues" on page 2 – 26. See also *Understanding SQL*Net.*

**Note:** If the network includes Oracle Names, the service names and associated connect descriptors are stored in a database that is accessed by the Names servers, and the TNSNAMES.ORA file is not needed. Similarly, if a native names adapter (such as NIS) is being used, this information will be stored in the corresponding native name service.

## Disconnecting from Servers

Requests to disconnect from the server can be initiated in the following ways:

- user–initiated disconnect
- additional connection request
- abnormal connection termination
- timer initiated disconnect

**User–Initiated Disconnect** A user can request a disconnection from the server when a client–server transaction completes. A server can also disconnect from a second server when all server–server data transfers have been completed, and no need for the link remains (the simplest case).

**Additional Connection Request** If a client application is connected to a server and requires access to another user account on the same server or on another server, most Oracle tools will first disconnect the application from the server to which it is currently connected. Once the disconnection is completed, a connection request to the new user account on the appropriate server is initiated.

**Abnormal Connection Termination** Occasionally, one of the components below SQL*Net will be disconnected or will abort communications and SQL*Net will not be immediately informed.

During the next SQL*Net data operation, the TNS module will recognize the failure and give SQL*Net a notice to clean up client and server operations, effectively disconnecting the current operation.

**Timer Initiated Disconnect** or Dead Connection Detection (SQL*Net release 2.1 and later only). Dead connection detection (Keep Alive or Dead Man's Handle) is a feature that allows SQL*Net to identify connections that have been left hanging by the abnormal termination of a client. On a connection with Dead Connection Detection enabled, a small probe packet is sent from server to client at a user–defined interval (usually several minutes). If the connection is invalid (usually due to the client process or machine being unreachable), the connection will be closed when an error is generated by the send operation, and the server process will exit.

This feature minimizes the waste of resources by connections that are no longer valid. It also automatically forces a database rollback of uncommitted transactions and locks held by the user of the broken connection.

## Data Operations

SQL*Net supports four sets of client–server data operations:

- send data synchronously
- receive data synchronously
- send data asynchronously
- receive data asynchronously

The concept of sending and receiving data between client and server on behalf of the UPI and OPI is relatively straightforward. A SQL dialogue request is forwarded from the UPI using a send request in SQL*Net. On the server side, SQL*Net processes a receive request and passes the data to the database. The opposite occurs in the return trip from the server.

All send and receive requests are synchronous. That is, when the client initiates a request, it waits for the server to respond with the answer. It can then issue an additional request.

SQL*Net version 2 adds the capability to send and receive data requests asynchronously. This capability was added to support the Oracle7 multi–threaded server, which requires asynchronous calls to service incoming requests from multiple clients.

## Exception Operations

SQL*Net supports three exception operations:

- initiate a break over the TNS connection
- reset a connection for synchronization after a break
- test the condition of the connection for incoming break

Of these three operations, only the initiation of a break can be controlled by the user. When the user presses the Interrupt key (Ctrl–c on some machines), the application calls this function. Additionally, the database can initiate a break to the client if an abnormal operation occurs, such as during an attempt to load a row of invalid data using SQL*Loader.

The other two exception operations are internal to some products using SQL*Net to resolve network timing issues. SQL*Net can initiate a test of the communication channel, for example, to see if new data has arrived. The reset function is used to resolve abnormal states, such as getting the connection back in synchronization after a break operation has occurred.

## SQL*Net and the Network Listener

TNS includes a protocol independent application listener that receives connections on behalf of any TNS application, over any underlying protocol. Referred to as a network listener, it runs as a single process or task and can service the needs of all TNS applications over all protocols available on a machine.

### Network Listener and Native Listeners

The network listener is available for all standard transport protocols supported by TNS. In addition, there are protocols that have application generic listeners or connection acceptance methods, such as DECnet and APPC / LU6.2, that may receive TNS connections.

OSDoc

**Additional Information:** For information on SQL*Net version 2 connections with a native connection acceptance method, see the Oracle operating system–specific documentation for that protocol and platform.

### SQL*Net and the Network Listener

SQL*Net version 2, as a TNS–based product, uses the network listener on a server to receive incoming connections from SQL*Net clients. The network listener listens for SQL*Net connections on a specific port or socket, which is defined in the ADDRESS portion of the connect descriptor.

### Prestarted Dedicated Server Processes

SQL*Net release 2.1 and later provides the option of automatically creating dedicated server processes. With this option, when the listener starts, it creates Oracle server processes which are then available to service incoming connection requests. These processes may last for the life of the listener, and they can be reused by subsequent connection requests.

**Note:** Prespawned dedicated servers requires SQL*Net release 2.1 or later, and requires Oracle7 Server release 7.1 or later.

Prestarted dedicated server processes reduce connect time by eliminating the need to create a dedicated server process for each new connection request as it comes to the listener. They also provide better use of allocated memory and system resources by recycling server processes for use by other connections without having to shut down and recreate a server. The use of prestarted dedicated server processes is particularly useful in systems where the Oracle7 Multi–Threaded Server is unsupported, or where the creation of a new server process is slow and resource–intensive.

Figure 2 – 7 shows the role of the network listener in a SQL*Net connection to a server connected to two communities.

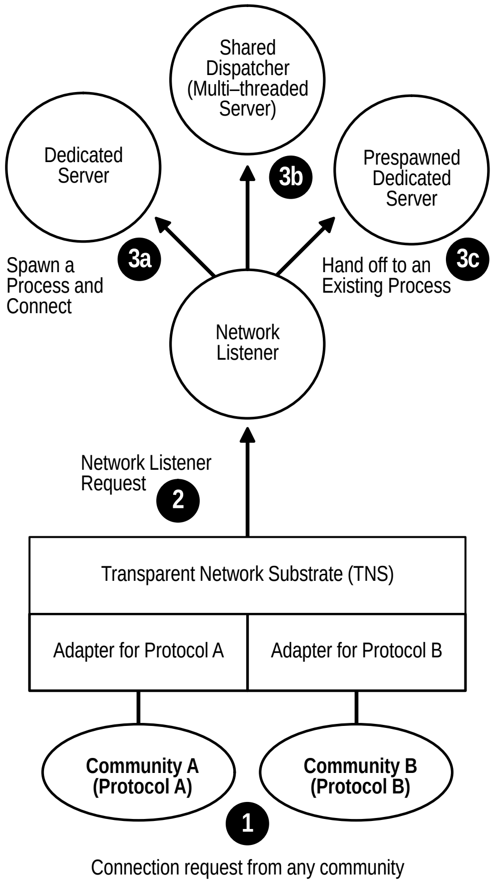

Shared Dispatcher (Multi–threaded Server)
Dedicated Server
3b
Prespawned Dedicated Server
Spawn a Process and Connect
3a
Hand off to an Existing Process
3c
Network Listener
Network Listener Request
2
Transparent Network Substrate (TNS)
Adapter for Protocol A
Adapter for Protocol B
Community A (Protocol A)
Community B (Protocol B)
1
Connection request from any community

**Figure 2 – 7 Network Listener in SQL*Net Connection**

The steps involved in establishing a connection (as shown in Figure 2 – 7) are:

**Step 1.** A connection request is made by any client in the TNS network and arrives through one of the communities to which the listener is attached.

**Step 2.** The network listener identifies that a connection request has arrived in one of its communities.

**Step 3.** **a.** The network listener spawns a dedicated server process and passes control of the incoming connection to it, or,
**b.** the address of a shared dispatcher process (multi–threaded server) is provided, and the incoming connection is directed to it, or,
**c.** the incoming connection is redirected to one of the prespawned dedicated server processes.

At the completion of a connection, the network listener continues to listen for additional incoming connections.

## How SQL*Net Establishes Connections to a Prespawned Dedicated Server

Prestarted (commonly referred to as “prespawned”) Oracle7 Servers are server processes that are pre–started by the Listener before any incoming connection request. They improve the time it takes to establish a connection on servers where the Multi–Threaded Server is not used or not supported on a given machine. Their use in a heavily loaded distributed system can be beneficial.

The following parameters must be specified for each SID to be prespawned and are located in their respective SID_DESC in the LISTENER.ORA file. They control how the server is spawned.

| | |
|---|---|
| PRESPAWN_MAX | The maximum number of prespawned servers the listener creates. This value should be a large number and at least the sum of the POOL_SIZE for each protocol. |

| | |
|---|---|
| POOL_SIZE | The number of unused prespawned server processes for the listener to maintain on the selected protocol. The number must be greater than zero, but no larger than the PRESPAWN_MAX value. Set this value to the average expected number of connections at any given time. |
| TIMEOUT | The time that an inactive server process should wait for the next connection before it shuts down. This parameter is used to prevent server processes from being immediately shut down after a client disconnects. For greatest efficiency, provide a short time value for this parameter. |

An additional feature of prespawned servers is the ability to set specific parameters for each SID. Thus, systems with heavy use can be tailored to accommodate the larger number of connection requests by setting PRESPAWNED_MAX and POOL_SIZE to large values. Similarly, when systems require mostly shared connections, the number of prestarted servers can be set to a low value.

Following is the sequence of events that occur when you are using prestarted servers to service client connection requests.

1. The listener is started and listens on the addresses pre–configured in LISTENER.ORA, created by the network administrator using Network Manager.
2. The listener then spawns a series of server processes until it reaches the POOL_SIZE for each SID defined in LISTENER.ORA.
3. Each spawned server process performs a wildcard listen and provides the Listener with the wildcard address that it is listening on. The listener initially marks all pre–started servers as idle.
4. The client calls the pre–configured well–known address of the listener.
5. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect. If not, the listener refuse the connection and then resumes at step 9.
6. The listener issues a redirect message to the client containing on e of the wildcard listen addresses of the pre–spawned servers. The listener then logs that server as active.

7. The client dissolves the connection to the listener and then establishes a connection to the pre–spawned server using the address provided in the redirect message.
8. The listener spawned another server to replace the active pre–spawned server (provided the PRESPAWN_MAX value is greater than the number of pre–spawned server processes either active or idle).
9. The listener continues listening for incoming connections.

The above sequence of events continues until the PRESPAWN_MAX is reached, at which point the listener will cease spawning new servers.

When clients disconnect, the prespawned server associated with the client is returned to the idle pool. If then waits the length of time defined in the TIMEOUT parameter to be assigned to another client. If no client is handed to the pre–spawned server before TIMEOUT expires, the pre–spawned server shuts itself down.

See the *Oracle Network Manager Administrator's Guide* for more information.

## How SQL*Net Establishes Connections to a Multi–Threaded Server

A multi–threaded server enables many clients to connect to the same server without the need for dedicated server processes for each client. Using the Multi–Threaded Server enables you to minimize the memory and processing resources needed on the server side as the number of connections to the database increases.

The sequence of events that occurs with the Oracle7 multi–threaded server can occur in two stages:

- The listener and the multi–threaded server start up.
- Clients connect to the Oracle7 multi–threaded server.

### What Happens When an MTS and Listener are Started

During initial startup of the Oracle7 multi–threaded server and the listener, the following sequence occurs:

1. The listener starts and listens for the addresses pre–configured in the LISTENER.ORA file. (The network administrator creates this file with Network Manager.)

2. The DBA starts the Oracle7 database. Dispatchers start according to the configuration parameters in the initialization parameter file. Dispatchers each use one particular protocol. There may be many, on assorted protocols. Each dispatcher performs a wildcard listen for the protocol assigned to it.

   **Note:** A *wildcard listen* is where the server process listens, but informs the underlying protocol stack (or operating system in the case of the IPC Protocol Adapter) that it has no preference as to what address it listens for other than specifying the protocol on which it wishes to perform the operation. As a result, many operating systems will choose a free listening address and automatically assign this to the requesting server process.

3. Each dispatcher calls the listener using the protocol it is assigned, at the address defined in the MTS_LISTEN_ADDRESS in the initialization parameter file.

   **Note:** If step 2 is performed before step 1, the dispatchers will be unable to contact the listener in step 3. If this occurs, each dispatcher loops and attempts to reconnect to the listener every 60 seconds. Meanwhile, incoming connection requests will be handled through other means (prespawned dedicated or newly–spawned dedicated server processes).

The listener and the Oracle7 multi–threaded server should be ready for incoming connections, at this point. You can check which dispatchers have registered with the Listener by typing

```
lsnrctl services listener_name
```

### How a Multi–Threaded Server Connection Request is Handled

The following is how a multi–threaded server connection request is handled:

1. The client calls the preconfigured well–known address of the listener.
2. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect (by checking the list of SIDs it listens for), at which point it continues with step 3. If not, the listener refuses the connection and then resumes at step 6.
3. The listener issues a redirect message back to the client containing the address of the least–called dispatcher that is listening on the protocol used by the client.
4. The client closes the connection to the listener and then establishes a new connection to the dispatcher, using the address provided by the Listener in the redirect message.

5. The listener and dispatcher perform a short handshake to update each other of the presence of a new connection. This is so that the listener can load balance connections between dispatchers running on the same protocol.

6. The listener resumes listening for incoming connections.

When an Oracle7 Server has been configured as a multi–threaded server, incoming connections are always routed to the dispatcher unless the connection request specifically requests a dedicated server (by having SERVER=DEDICATED in the CONNECT_DATA portion of the connect descriptor) or no dispatchers are available.

## Global Naming Issues

The following sections explain Oracle's naming scheme and how references to network objects are resolved within a distributed system.

### Global Database Names (Service Names)

Every database in an Oracle network has a *global database name*, more commonly referred to as a *service name*. The global database name uniquely identifies each database in the global network. A global database name typically consists of a database name (DB_NAME) and a hierarchical domain name (DB_DOMAIN).

For example, HR.US.ACME.COM is a global database name for the "HR" database, which is located in the US.ACME.COM domain. From the viewpoint of SQL*Net and Oracle Names, HR.US.ACME.COM is also the service name. The network domain component of a global database name must follow standard Internet conventions. Levels in domain names are separated by dots, and the order of domain names is from leaf to root, left to right.

For example, Figure 2 – 8 illustrates a representative hierarchical arrangement of databases throughout a network and how a global database name is formed.

**Note:** Do not confuse Oracle global database names with SQL*Net community names. A SQL*Net community is a group of machines and network services that communicate using the same protocol. A global database name consists of a database name and a domain name. Domains only exist for naming and administrative purposes, and have no functional relationship to community names.

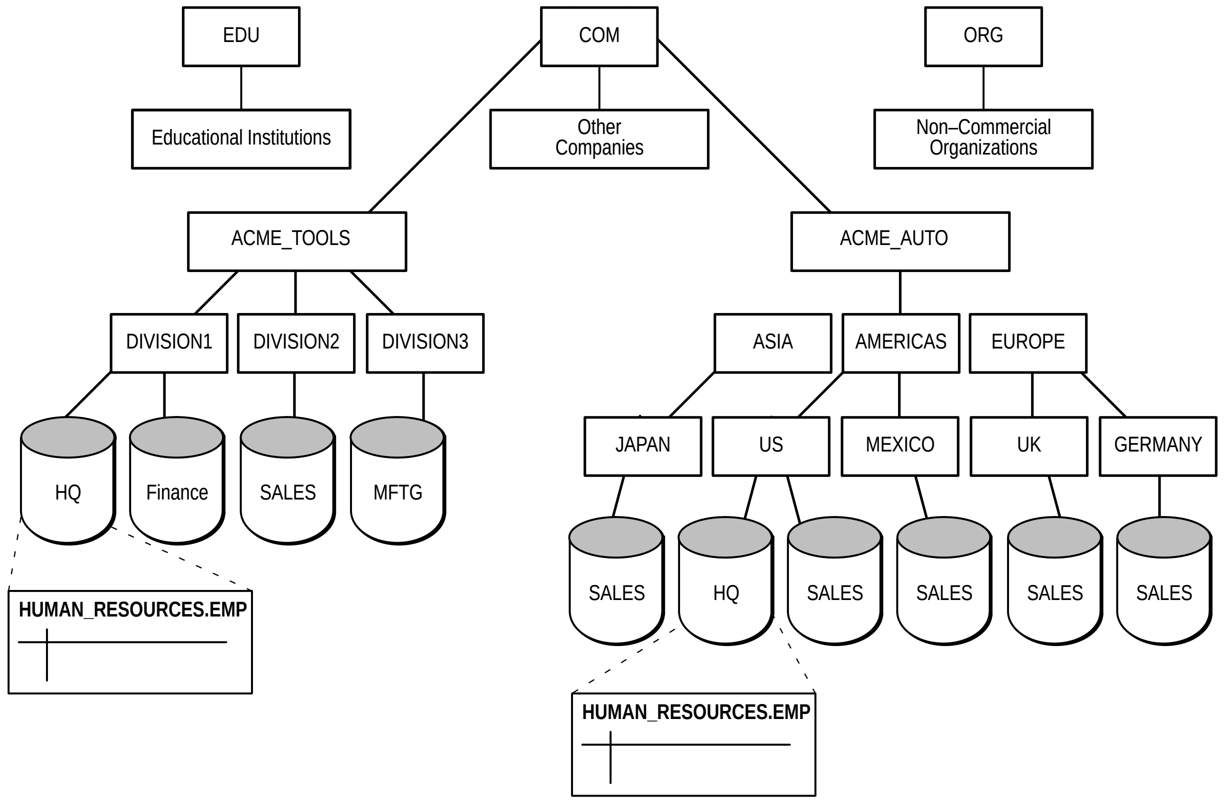


**Figure 2 – 8 Network Directories and Global Database Names**

Notice that throughout the network there are several databases with the same name (such as SALES). However, also notice that each database has a unique global database name because of its location within the network domain structure. From left to right, global database names are as follows:

```
hq.division1.acme_tools.com
finance.division1.acme_tools.com
sales.division2.acme_tools.com
mftg.division3.acme_tools.com
sales.japan.asia.acme_auto.com
hq.us.americas.acme_auto.com
sales.us.americas.acme_auto.com
sales.mexico.americas.acme_auto.com
sales.uk.europe.acme_auto.com
sales.germany.europe.acme_auto.com
```

Because each database has a unique global database name, each database and its objects can be uniquely identified with the objects' global object name. For example, notice that each HQ database contains a table named EMP. However, each EMP table can be uniquely identified with its global object name. In Figure 2 – 8, the global names for the two EMP tables are:

```
human_resources.emp@hq.us.americas.acme_auto.com
human_resources.emp@hq.division1.acme_tools.com
```

Each local data dictionary in an Oracle distributed system stores object names and names of containing schemas only, not complete global object names. However, because each database can have a unique name within a network, and because each object name is guaranteed to be unique within the scope of a single database; each object in a database in the distributed system has a unique global object name.

## Network Domains and Network Naming Services

Figure 2 – 8 illustrated a fictional network domain structure that follows standard Internet conventions. Network domains are similar to the file directories used by many operating systems (such as UNIX). However, unlike file systems, network domains may or may not correspond to any physical arrangement of databases and other structures in a network. Network domains might simply be name spaces. The availability and functionality of a network naming service dictate what is possible.

If Oracle Names servers are available, they can be configured into your network to perform name resolution. For example, Network Manager creates and drops network domains, controls access to network domains, creates and drops network objects (such as databases) within the network structure, and enforces the unique naming of objects within the network.

Oracle's architecture uses network naming services, such as Oracle Names, Network Information Services (NIS), and Domain Name System (DNS). Whether you are using a network naming service, such as Oracle Names or TNSNAMES.ORA name lookup files, to resolve names to addresses, you still need to follow global database naming conventions. Also keep in mind the following:

- All databases have global database names, whether or not you are using a network naming service such as Oracle Names.
- Database links (global, public, and private) facilitate remote connections and allow global name resolution.
- Global database link names are the same as the service name (or global database name). For example, HR.US.ACME.COM is the service name (global database name) and the global database link name.

- Global database links are created automatically when administrators configure Oracle Names servers into the network, and define databases in Network Manager. This eliminates the need to store and create public and private database links in each data dictionary.

You can define a database link so that the user accessing remote data connects to the remote database either with the local username and password or an explicitly–specified username and password. In other words, a database link can be defined so that all users of the link connect to the remote database with either a central, explicitly–specified remote account or an implied individual remote accounts.

### Local Object Names

In a distributed or non–distributed environment, Oracle guarantees that each database has a unique set of schemas. Within each schema, an object name is unique within its name space. Therefore, each schema object's name is guaranteed to be unique within the context of any given Oracle database, and the local Oracle node easily can resolve any references to objects within the local database. Each local data dictionary stores only the names of local objects (and synonyms), not remote object names.

## Database Links

Oracle uses *database links* to facilitate connections between the individual databases of a distributed system. A database link defines a *path* to a remote database by uniquely identifying and specifying the location of a remote database.

**Note:** Remember that a global database link is created automatically for each database defined in Network Manager. However, public and private database links are typically created by users or database administrators.

### Database Link Name same as Global Database Name

A database link defines a path to a remote database. The two components of a path are a remote account and a database string. Database links are essentially transparent to users of a distributed system, because the name of a database link is the same as the global name of the database to which the link points. For example, the following statement creates a database link in the local database. The database link named SALES.DIVISION3.ACME.COM describes a path to a remote database of the same name:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
USING 'sales.division3.acme.com'
```

At this point, any application or user connected to the local database can access data in the SALES database by using the global object name (SALES.DIVISION3.ACME.COM). The SALES.DIVISION3.ACME.COM database link implicitly facilitates the connection to the SALES database. For example, consider the following remote query that references the remote table SCOTT.EMP in the SALES database:

```
SELECT * FROM scott.emp@sales.division3.acme.com;
```

## National Language Support (NLS) and Database Links

When a user session connects to an instance, the values of NLS parameters used by the instance for that user session are defined by the value of the initialization parameter NLS_LANG for that session. This applies to direct and indirect connections.

If the values of the NLS parameters are changed during a session by an ALTER SESSION statement, the changes are automatically propagated to all instances to which the user session is connected, either directly or indirectly. For more information on National Language Support features, see the *Oracle7 Server SQL Reference.*

## Types of Database Links

Oracle uses several types of database links to resolve users' references to global object names:

| | |
|---|---|
| private database link | Created on behalf of a specific user. A private database link can be used when the owner of the link specifies a global object name in a SQL statement, or in the definition of the owner's views or procedures. |
| public database link | Created for the special user group PUBLIC. A public database link can be used when any user in the associated database specifies a global object name in a SQL statement or object definition. |
| global database link | Created and managed by a global naming service such as Oracle Names. A global database link can be used when any user of any database in the network specifies a global object name in a SQL statement or object definition. |

Public and private database links are stored in the data dictionary of a database. Global database links are not.

Each type of database link has advantages and disadvantages, as compared to the other types. For example, you can maintain tighter security with private than with public or global database links. The use of a private database link is at the discretion of the owner of the link.

#### Database Links in SQL Statements

The owner of a private database link can use the link in his / her own SQL statements and selectively allow other users to use the private link by creating views, procedures, or synonyms that reference the link in their definitions. Otherwise, there is no way to restrict the use of a public database link selectively (that is, any local user can connect to the remote database specified by the public database link).

### Database Links and Security

In a distributed system, application developers and individual users are often allowed to create private database links. However, you must account for the extra security responsibilities required in a distributed system. See page 6 – 11 for more information on security issues to consider when implementing a distributed system.

## Database Links and Connection Qualifiers

Connection qualifiers provide a way to have several database links of the same type (for example, public) that point to the same remote database, yet establish those connections using different communications pathways.

A *connection qualifier* is a method of aliasing a database link to a particular communication pathway (or instance in the case of the Oracle Parallel Server). The connection qualifier is an identifying string appended to the database link name It is preceded by an at sign (@) (for example, emp.scott@HQ.ACME.COM@DBMS1).

For example, you have a database that is connected to the HQ.ACME.COM database DBS2 by an ethernet link and by a slower modem link. You want to access the DBS2 by both communication links allowing higher priority applications to use the faster ethernet link. You could define the following database links:

```
CREATE PUBLIC DATABASE LINK hq.acme.com@ethernet
    USING ’ethernet_to_hq.acme.com’;
CREATE PUBLIC DATABASE LINK hq.acme.com@modem
    USING ’modem_to_hq.acme.com’;
```

Note that in the above examples, the connection qualifiers (@ethernet, @modem) are appended to the database link name. The connection qualifier does not necessarily specify how the connection is to be established; this information is specified by the USING clause.

Based on the connection qualifiers specified above, the following statement would use the ethernet connection to HQ.ACME.COM:

```
SELECT * FROM scott.emp@hq.acme.com@ethernet
```

Connection qualifiers can also be defined to use different instances at a node where the remote database is managed by the Oracle Parallel Server.

OSDoc

**Additional Information:** For more information about database links and connection qualifiers, see the *Oracle7 Server SQL Reference* and your operating system–specific SQL*Net documentation. Oracle Names can also be used to define database links (except those in a replicated environment). See the *Oracle Names Administrator's Guide* for more information.

## Oracle Names

Oracle Names is a distributed name service that resolves database service names and database links to network addresses, and makes them available to all clients in the network. When Oracle Names servers are used, it is no longer necessary to update every TNSNAMES.ORA file on every client whenever a change is made to an existing server or a new server is added to the network. Oracle Names is configured through Oracle Network Manager, so changes to an environment only need to be made at a single point for them to be available to all clients and servers.

The advantages of using Oracle Names servers are:

- Network names and addresses can be stored in a central location on a database server, instead of being stored in TNSNAMES.ORA name lookup files on each client and server. Changes only need to be made using one utility—Network Manager—by administrators, instead of having to update name lookup files on hundreds or thousands of clients.
- Oracle Names provides further location transparency by storing global database links, which eliminate the need to define and store them in local data dictionaries. Public and private database links will still be resolved properly. However, when using Oracle Names servers, they are no longer necessary.
- Multiple Oracle Names servers can be distributed throughout an administrative region for constant coverage. If one Names server should temporarily fail, another Names server in that region can answer name requests.

For more information on Oracle Names, see the *Oracle Names Administrator's Guide.*

## Automatic Creation of Global Database Links with Network Manager

When you define a network that includes Oracle Names, Network Manager automatically creates a global database link to every database server you define from every other database server in the network. These database links do not reside in the data dictionary, but in the network definition to which the Names servers refer. The default database links created do not initially include a CONNECT TO clause (that is, a username and password), so users reach the linked database using the same usernames and passwords as they use to reach the first database.

```
SQL> SELECT * FROM EMP@OHIO, DEPT@NY_FIN;
```

### Explicitly Defined Database Links

You can edit global database links to include a username and password using Network Manager. When you edit a database, you can specify a single default username and password for the database link. See the *Oracle Network Manager Administrator's Guide* for details on how to edit global database links.

### Connection Qualifiers

You can also define *connection qualifiers* to global database links through Network Manager. Connection qualifiers provide a way to create more than one link to a given database. These alternate links are a useful way to access different accounts on the same database with different sets of access privileges. The alternate link created by a connection qualifier must include a reference to a database by its global database name (or service name). See the *Oracle Network Manager Administrator's Guide* for details on how to create connection qualifiers using Network Manager.

## How SQL*Net Resolves Service Names

When a user types in a service name, SQL*Net resolves it to an address using a variety of mechanisms in the following order:

- First, SQL*net looks for the local TNSNAMES.ORA file for the user.
- SQL*Net then searches the global TNSNAMES.ORA file.
- SQL*Net then looks for the Oracle Names server address in the SQLNET.ORA file and contacts the Oracle Names server for resolution of the service name. Multiple Oracle Names server addresses can be stored in SQLNET.ORA. These addresses will be tried, in order, if the client is unable to connect to the first Oracle Names server.

If all attempts to resolve the name fail, Oracle issues the error message O*RA–12154 TNS: could not resolve database name.*

## How Oracle Names Resolves Service Names

Service names (also called global database names) are translated to addresses in SQL*Net using the following method:

- If the name is fully–qualified, in which case it contains the service name and the domain name, the given name is used. If not, SQL*Net will attach the domain suffix specified in the client profile. The domain suffix is stored in the SQLNET.ORA file under the entry NAMES.DEFAULT_DOMAIN.
- SQL*Net then looks for the name in the client's TNSNAMES.ORA file, the search path of which is described in the previous section "How SQL*Net Resolves Service Names." If the name is still not resolved to an address, SQL*Net will then contact Oracle Names to resolve the address. In general, the use of Oracle Names replaces TNSNAMES.ORA as the way to map names to addresses.

Consider the following example:

```
SQLPLUS scott/tiger@hr
```

will connect to the database HR.US.ACME.COM if the client profile contains a default domain of US.ACME.COM.

```
SQLPLUS scott/tiger@hr.us.acme.com
```

is fully qualified and properly identifies the database HR.US.ACME.COM.

## How Oracle Names Resolves Database Links

Similarly, for database links, the database looks at any defined private or public database link definitions and if not fully–qualified, the database domain (the GLOBAL_NAME minus the part preceding the first dot) is tacked on the database name of the link. If no USING clause is specified in the private or public database link definitions, and the database's client profile specifies one or more Oracle Names servers, these servers are called to resolve the database link name.

SQL*Net then receives either the database link's USING clause or the information returned by the Oracle Names server. If the USING clause contains a name, the name resolution process described above is then used to get the address. If the USING clause contains an address, the database link definition returned by Oracle Names is passed to SQL*Net, and name resolution is bypassed because an address has been directly provided.

**Note:** Though an address (SQL*Net connect descriptor) could conceivably be specified in the USING clause, a global database name is typically specified.

Consider the following example on the database MFG.US.ACME.COM:

A public database link HR@FIN exists and a user performs:

```
SELECT * FROM EMP@HR@FIN
```

The database will translate the database link name to HR.US.ACME.COM@FIN and call Oracle Names for link resolution because no USING clause was specified on the created link.

See *Understanding SQL*Net* and the *Oracle Names Administrator's Guide* for more information on service name and database link resolution.

### Using a SELECT Statement across a Database Link

When you issue a select of a table across a database link, you acquire a transaction lock and a slot in the transaction table for the rollback segment for the local database. The lock can be released only by a commit or rollback.

## Network Issues to Consider When Implementing a Distributed Database System

This section describes some things the network administrator must consider before deciding on the structure of a distributed system. Read this section before using the Oracle Network Manager to configure the network.

### Planning your Network

When planning your SQL*Net network, think about future needs as well as present requirements. Select a layout that is flexible and expandable. If you foresee your network growing, select computers that have the capacity to handle additional connections. When naming the components in your system, think about how your naming conventions can be extended to handle future components.

### Draw the Network Layout

It is a good idea to draw a picture of your network layout as you decide about its composition. Especially if your network includes multiple communities, Interchanges and Names servers, it is much easier to understand and modify if you have a diagram. Two types of diagrams are useful:

- physical diagrams
- logical diagrams

Physical diagrams show every component in a network, including the physical connections among them. A physical diagram can help show what pieces are required and demonstrate the connections between components.

Logical diagrams show the relationships between network components without going into detail about their physical placement. The figures throughout this book are good examples of the sort of graphical representation that is needed. In general, the more complex the network, the more necessary a visual mapping.

## Select Network Protocols to be Used

The first decision to make when designing a network is whether it will include only one protocol or more than one protocol.

As explained in "SQL*Net version 2 Architecture" on page 2 – 11, SQL*Net runs above TNS, which in turn runs over a transport level protocol, with an Oracle Protocol Adapter acting as an interface between TNS and the protocol of choice. The specific hardware below the transport layer is irrelevant to SQL*Net's functioning.

You may be able to choose a single transport level protocol that works well on all the components in your network. Protocol adapters are available for most of the major protocols on many platforms. If you have only one protocol in your network, as shown in Figure 2 – 9, then all the components are members of the same *community.*

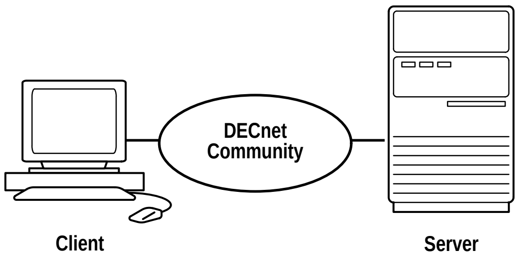


**Figure 2 – 9  A Single Community Network**

For reasons of necessity or efficiency, you may choose to have more than one protocol running in your network. You may do this by having multiple protocol adapters on the computers in your network, or, more efficiently, you may have an Interchange between the computers running one protocol and the computers running another.

Using SQL*Net and the MultiProtocol Interchange, individual computers can communicate across the protocols that are most compatible with their operating systems. For example, you can have personal computers running Novell's SPX/IPX connected to a VAX server that uses the DECnet protocol. If your network uses one or more Interchanges, as shown in Figure 2 – 10, it is a multicommunity network.

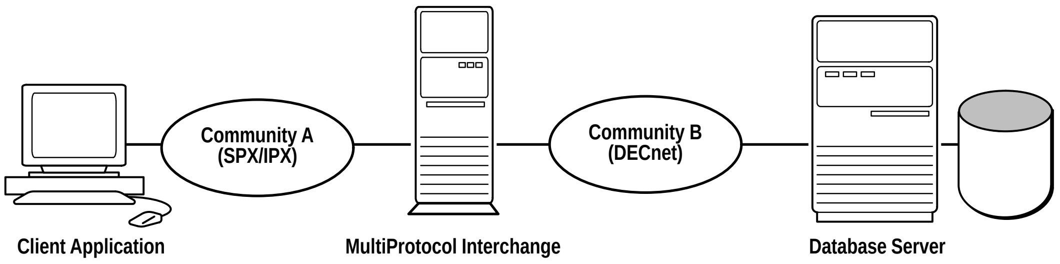


**Figure 2 – 10  A Multicommunity Network**

## Choose Nodes as Interchanges

If you decide on a multicommunity network, you must choose what nodes to use for Interchanges to connect the communities. Considerations include:

- What computers work well on all the protocols they connect?
- What computers have the capacity to handle the anticipated traffic?
- Should the computers chosen be dedicated to the Interchange service, or can they handle other applications as well?

For more information about Interchanges, see the *Oracle MultiProtocol Interchange Administrator's Guide.*

### Choose a Node to Run Network Manager

Select a location for Network Manager from which it is relatively easy to transfer configuration files to other network components. The Network Manager includes a utility, NetFetch, which helps you do this, but a SQL*Net network (either version 2 or release 2.1) must be up and running before it can be used.

For more information about using the Oracle Network Manager to create SQL*Net configuration files, see the *Oracle Network Manager Administrator's Guide.*

## Decide on the Structure of Network Administration

Most networks have one central point of administration, that is, one installation of the Oracle Network Manager. However, if you are using Oracle Names and your network is quite large or widely distributed geographically, you may choose to have several regions of network administration.

If your enterprise–wide network includes both the United States and Europe, you might want to have administrative decisions about the network made locally.

For example, it would be more efficient if a network administrator in Chicago had jurisdiction over the names and locations of US network services, while an administrator in Brussels was responsible for decisions about a European network.

For more information about centralized and decentralized administration, see the *Oracle Names Administrator's Guide.*

### Decide which Nodes will Run Oracle Names Servers

If you use Oracle Names to provide a centralized naming service for your network, you must decide what nodes should contain Names servers, which provide name and address information to enable connections throughout the network. Currently, you must have a name server in every community. In general, Oracle recommends that Names servers in a multicommunity network be placed on MultiProtocol Interchange nodes, thereby minimizing the number of Names servers required.

For more information about Oracle Names, see the *Oracle Names Administrator's Guide.*

### Decide on a Organizational Naming Standard

When you name entities such as nodes in a networked environment, you should ensure that object names are unique within the network. This can be a challenge if your organization is large, and network administration is not handled centrally.

You may be able to guarantee that all services in your building or jurisdiction have unique names, but that does not guarantee that the name is unique within the organization. Your goal should be to avoid the occurrence of duplicate names if multiple independent TNS communities are joined by installing an Interchange between them.

### Selecting Domain Names

A recommended network naming technique is to use hierarchical groups or domains in which each administrative unit is assigned to a unique domain based on the function it provides. Many of the examples in this guide feature the fictitious company, ACME Inc., which has segregated its naming domains as shown in Figure 2 – 11.

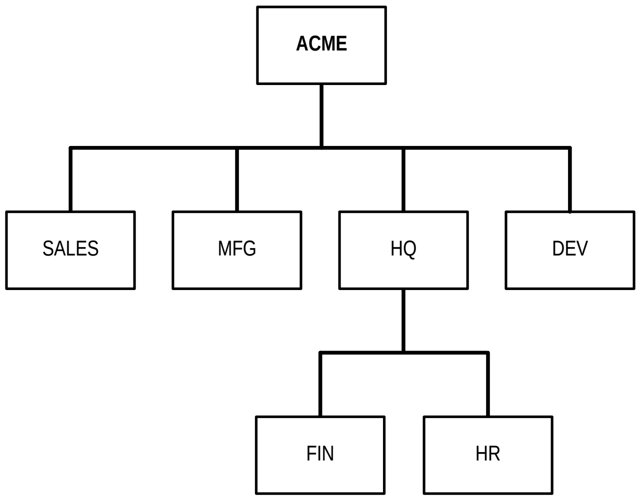


**Figure 2 – 11  Naming Domains at ACME**

In this figure, each of the boxes represents a separate domain. The domains are related hierarchically; that is, FIN and HR are the children of HQ, which is one of the children of ACME. A network object (such as an Interchange or a server) within a given domain has a unique name within that domain. This is generally manageable because one or at most a few people have authority to pass out names for that domain. The global name for that object includes the parent domains.

For example, consider the corporation shown in Figure 2 – 11. The sales organization (the SALES domain) could have a server named VAULT. The human resources group (the HR domain) could also have a server named VAULT. The global names of these servers would be unique. The finance group's server would have the global name

```
VAULT.SALES.ACME
```

while the global name of the human resources server would be

```
VAULT.HR.HQ.ACME
```

This structure is especially important if you are using a Oracle Names server to access any of the addresses or other information used within the network.

Names can go to the company level (the ACME stem) or can go to the inter–network level (for example, the ACME.COM stem) in support of inter–company communications such as mail or Electronic Data Interchange (EDI). If your organization belongs to the Internet, or you expect that it might join the Internet in the future, the domain names should include the appropriate stem (such as COM, GOV, or EDU).

If your organization already has global naming conventions, your network components should follow those conventions.

Network Manager automatically appends the default domain .WORLD to the name of all network components unless you provide alternative domain names. See the *Oracle Names Administrator's Guide* for more information.

## Default Domain

Every client and server has a default domain listed in its SQLNET.ORA file. The default domain is the domain to which most of the clients' connection requests are directed. The service names of databases in the default domain do not need to be fully qualified; that is, the domain name does not need to be included.

For example, if a client wanted to make a connection to a database in its default domain with the service name PROFITS.SALES.ACME.COM, it could do so using the following command:

```
% sqlplus scott/tiger@profits
```

However, if the PROFITS database were not in the client's default domain, the command would be:

```
% sqlplus scott/tiger@profits.sales.acme.com
```

**Note:** The default domain is not necessarily the same as the domain of the client itself. For example, clients in one domain may frequently access Oracle servers in another domain.

CHAPTER

3

# Network Administration Utilities

This chapter discusses some of SQL*Net's administrative capabilities and utilities. It also discusses some of the tasks you perform to configure and maintain your network.

Topics covered include:

- SQL*Net's administrative features
  - password encryption
  - client–side load balancing
  - connection route testing with ping
  - SNMP support
  - diagnostic logging and tracing capabilities
- Network Manager
- distributing configuration files
- testing your configuration

See page 4 – 21 for information on database administration utilities such as Server Manager, which lets a DBA administer and maintain a database.

## SQL*Net's Administration Utilities

SQL*Net has administrative, performance, and security related utilities, making it even easier to use in large environments. Some of these utilities are:

- password encryption
- client–side load balancing
- connection route testing with ping
- SNMP support
- diagnostic logging and tracing capabilities

### Encrypting Passwords with SQL*Net

The Listener Control Utility and the Oracle Names server provide for optional passwords to allow only authorized personnel to perform certain administration tasks.

Oracle Network Manager provides the option to encrypt these passwords within the configuration files for enhanced security. These passwords and SQL*Net logon passwords are also encrypted when passed over the network.

See the *Oracle Network Manager Administrator's Guide* for information on how to configure encrypted passwords for the listener and Names server control utilities. Also see *Understanding SQL*Net* and the *Oracle7 Server Administrator's Guide* for information on password encryption.

### Monitoring Network Services with SNMP Support

SQL*Net release 2.2 includes support for the Simple Network Management Protocol (SNMP) which allows the Oracle7 Server and the MultiProtocol Interchange, Oracle Names, and SQL*Net Listener to be monitored through SNMP–based platforms such as:

- HP's OpenView
- IBM's NetView / 6000
- Novell's NetWare Management
- SunSoft's SunNet Manager
- Digital's POLYCENTER

Support for these widely–used, SNMP–based management systems allows you to more easily integrate Oracle into your existing infrastructure. You have the freedom to choose the network management utility you want to use.

To monitor an Interchange, an Oracle7 database, SQL*Net listener, or Oracle Names server using Oracle SNMP support, you configure the server in Network Manager.

For example, if you want an Interchange to be monitored, you enter parameters into the SNMP Interchange Details window to control the Interchange subagent. Additionally, you can enter the name or other identifier of the person who administers this service.

See the *Oracle Network Manager Administrator's Guide* for more information.

### SQL*Net's Diagnostic Logging and Tracing Capabilities

SQL*Net version 2 provides tracing and logging abilities that enable you to record the behavior of each of the components of the network and to troubleshoot problems when they occur.

For detailed information about tracing and logging, see the *Oracle Network Products Troubleshooting Guide.*

### Connection Route Testing with PING

SQL*Net release 2.2 includes the TNSPING utility that allows clients to test connections to the servers. You can use TNSPING to test the status of the connection before actually logging on to the server.

This added functionality allows users to validate network connections before executing SQL statements. See *Understanding SQL*Net* for more information.

## Network Manager

Oracle Network Manager is a utility that assists you in creating the configuration files needed for Oracle networking products. It also provides graphical views of your network.

This section is intended only to provide an overview of Network Manager's capabilities. See the *Oracle Network Manager Administrator's Guide* for detailed information on how to configure a network using Oracle Network Manager.

Networking products require a number of configuration files. These configuration files, which have a very precise syntax, would be tedious to create manually. Network Manager provides a graphical user interface (GUI) through which administrators can create the configuration files quickly and accurately.

After creating a network definition using Network Manager, the configuration files are generated. Network Manager ensures that all resulting files are free from common syntax and semantic errors often made in manually created files. In fact, Oracle only supports files that are created with Network Manager (except for PROTOCOL.ORA and SQLNET.ORA files, which contain parameters that can be included only through manual editing).

Network Manager is similar to the utilities in Oracle's Cooperative Development Environment (CDE). If you are familiar with CDE, you will quickly be able to use Network Manager. Figure 3 – 1 shows Network Manager's interface running under Microsoft Windows.

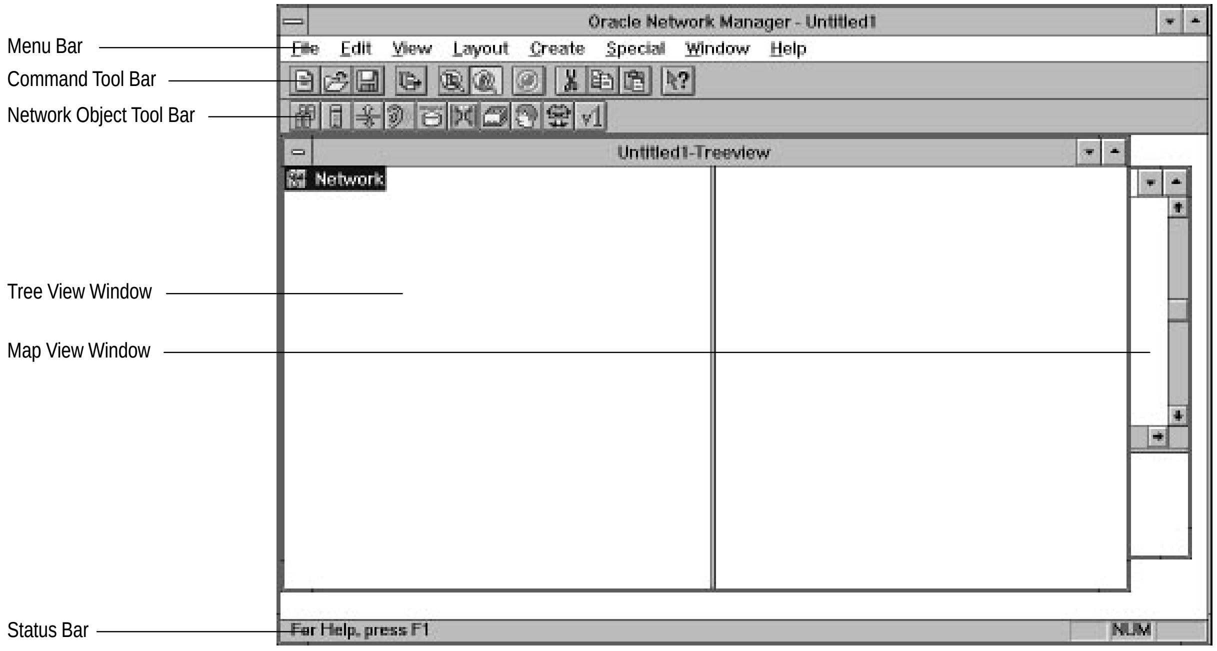


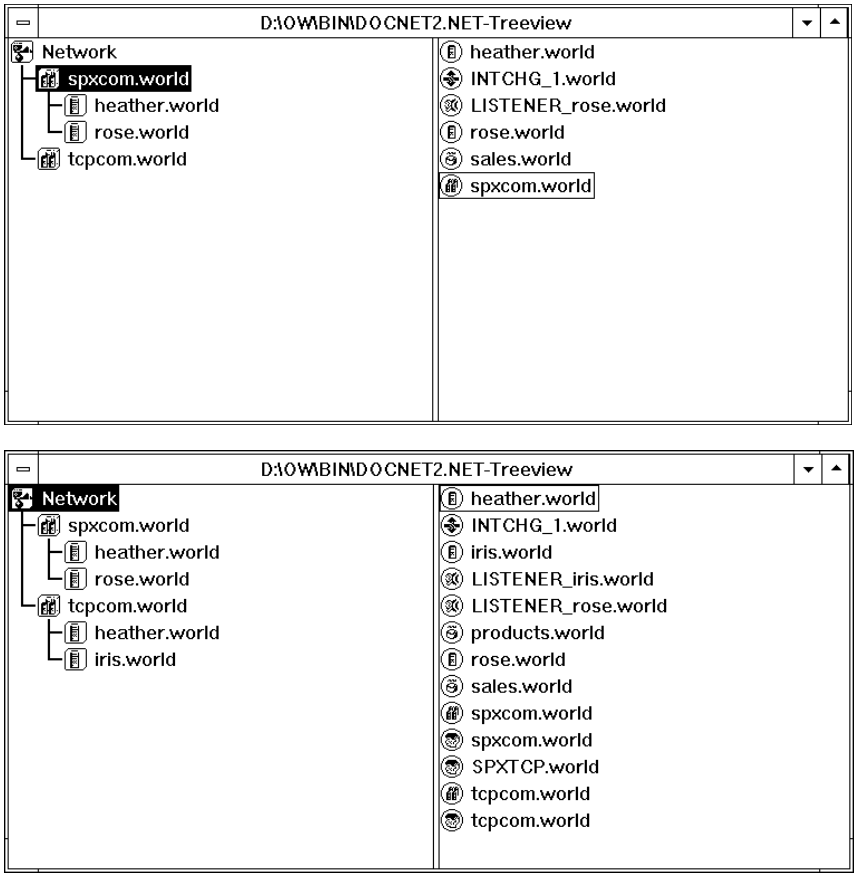
**Figure 3 – 1 Network Manager Interface**

## Viewing the Network

SQL*Net release 2.2 includes a new version of Network Manager (V3.0) which provides administrators two ways of visualizing a network. The network definition can be viewed in a "map" (topological) view or in a "tree" (hierarchical) view. See Figure 3 – 2 through  Figure 3 – 4 for sample illustrations of map and tree views of a network.

Network components, such as clients, servers, communities, etc. are treated as objects that can be selected, dragged and dropped, linked, among other operations within the network map.

### The Tree View



```
D:\OW\BIN\DOCNET2.NET-Treeview

Network                          heather.world
├─spxcom.world                   INTCHG_1.world
│  ├─heather.world               LISTENER_rose.world
│  └─rose.world                  rose.world
└─tcpcom.world                   sales.world
                                 spxcom.world
```

```
D:\OW\BIN\DOCNET2.NET-Treeview

Network                          heather.world
├─spxcom.world                   INTCHG_1.world
│  ├─heather.world               iris.world
│  └─rose.world                  LISTENER_iris.world
└─tcpcom.world                   LISTENER_rose.world
   ├─heather.world               products.world
   └─iris.world                  rose.world
                                 sales.world
                                 spxcom.world
                                 spxcom.world
                                 SPXTCP.world
                                 tcpcom.world
                                 tcpcom.world
```

**Figure 3 – 2  Two Tree Views of a Network**

The Tree View Window provides a tree, or hierarchical view of a network, indicating the relationships among the network services.

In Figure 3 – 2, the window on the left side of the screen displays domains or communities as well as their respective network components. The window on the right displays all network services belonging to the highlighted domain. This parent–child framework allows the administrator to clearly display network components and their relationships.

The administrator configures a network object either by selecting it from the toolbar and entering the appropriate information in its property sheet, or by dragging and dropping the selected toolbar object onto another predefined object. When a network object is dropped onto another one, Oracle Network Manager automatically updates it with the information from the object upon which is has been dropped.

For example, when a node is dropped onto a predefined community, the node is automatically linked to the community.

## The Map View

The topological or map view shows the network in a spatial layout connecting the network services as they relate to the user's actual network. It allows users to drag a client (or server) object and drop it in a desired location in the network and the configuration will be updated automatically.

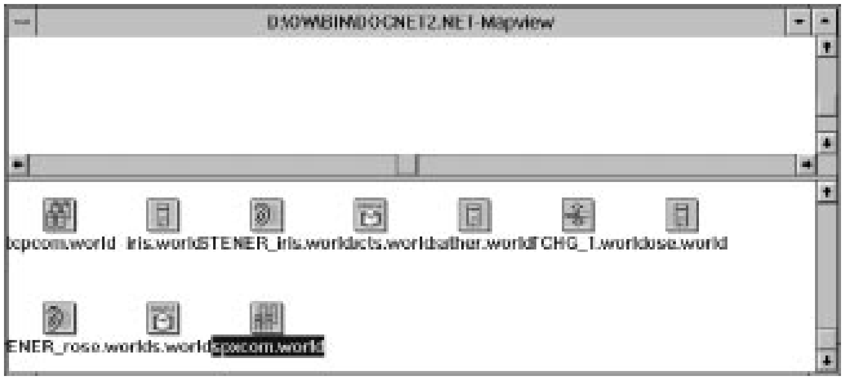


**Figure 3 – 3  Map View Window**

When you create a network object, an icon representing it appears in the bottom half of the Map View window. At any time you can select the icons in this window and move them to the upper part of the window.

Lines showing the objects' relationships will be displayed between them. Individual icons representing the network services will automatically connect to related nodes when placed on the map.

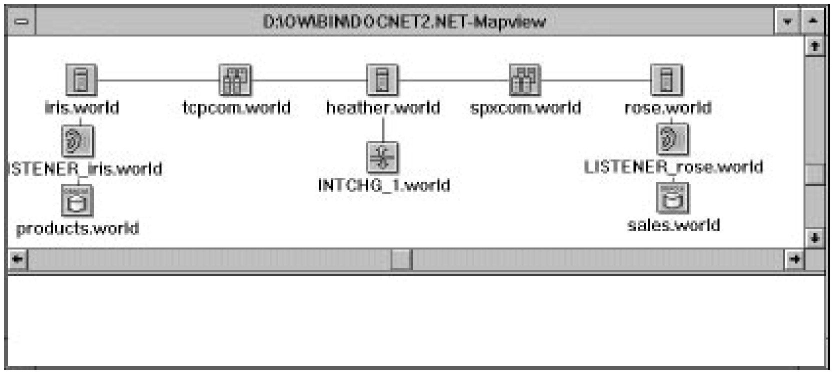


**Figure 3 – 4  Map View with Network Objects Arranged**

## Context–Sensitive Help

Network Manager V3.0 includes a context–sensitive help system, which allows you to display information about Network Manager while you are using it.

## Controlling Network Services from a Remote Site

You can use Network Manager to control and monitor network services, such as the MultiProtocol Interchange, Oracle Names server, and listener for an Oracle database from a remote site. For example, the administrator can set tracing levels in the network services and reload the Oracle Names server.

## Accessing Server Manager to Perform DBA Tasks

Provided Server Manager has been installed, you can access it from Network Manager V3.0. This allows a network administrator to perform certain DBA tasks such as database startup, shutdown, backup, and recovery while configuring the network. See page 4 – 23 for introductory information about Server Manager. For detailed information, see the *Oracle Server Manager User's Guide* and related documentation.

### Network Manager Migration Utilities

The Network Manager also includes a utility, NETCONV, to convert a SQL*Net version 2 network definition created by the version 2 configuration tool into a release 2.1 or 2.2 network definition that can be stored and edited by Network Manager.

Oracle Network Manager also makes moving from SQL*Net version 1 or SQL*Net version 2 easier. Network Manager can import a file of version 1 connect strings and aliases into a TNSNAMES.ORA file. They can then be recognized and used on operating systems (such as UNIX and VMS) that use TNSNAMES.ORA as their source of information about network destinations once SQL*Net version 2 is installed.

Network Manager provides an easy way to create a service name for a SQL*Net version 1 connect string that does not already have an alias mapped to it.

## Using Network Manager to Configure Your Network

You can configure the following network components, or network objects, in Network Manager:

- domains
- communities
- nodes
- databases
- SQL*Net listeners
- MultiProtocol Interchanges
- Oracle Names servers
- foreign regions
- local regions
- Oracle Transparent Gateways
- service name aliases
- SQL*Net version 1 connect strings

You will find complete instructions for using Network Manager in the *Oracle Network Manager Administrator's Guide*.

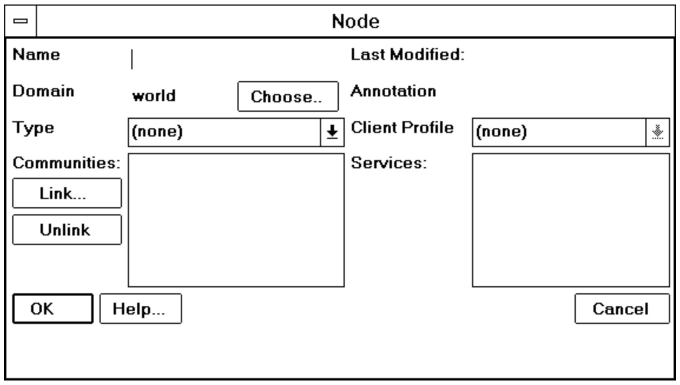
There are three major steps to configuring a network using Network Manager.

1. Create a network definition using Network Manager property sheets. This procedure is described in the *Oracle Network Manager Administrator's Guide.* Use the online help to get detailed information about using the property sheets as you use Network Manager.
2. Generate network component configuration files using the Generate command of Network Manager. This procedure is described in the *Oracle Network Manager Administrator's Guide.*
3. Pull the configuration files to the appropriate nodes on the network using NetFetch or FTP. This procedure is described in the *Oracle Network Manager Administrator's Guide.*

The three part process is illustrated in Figure 3 – 5.

1

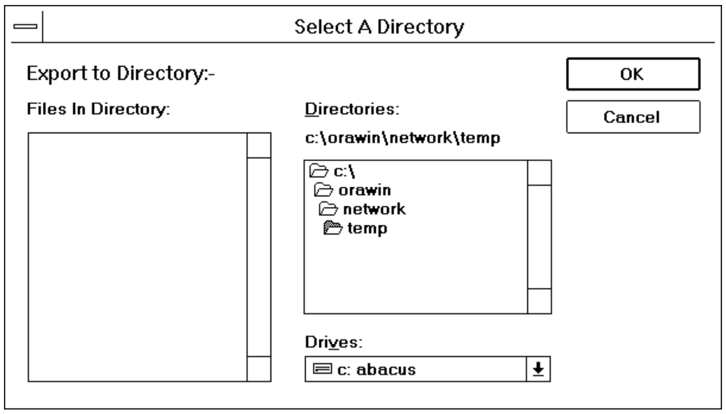
**Create Configuration Files**
(Network Manager)



Domain: world
Type: (none)
Client Profile: (none)

2

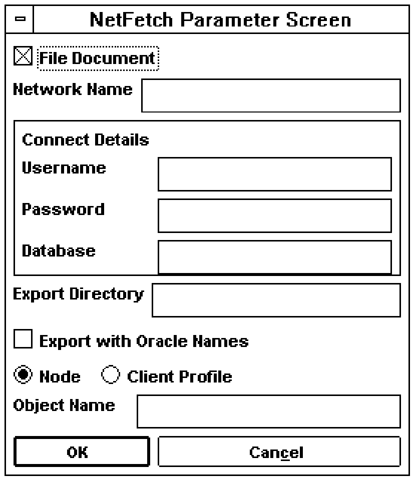
**Generate Configuration Files**
(Network Manager)



Directories: c:\orawin\network\temp
Drives: c: abacus

3

**Fetch to Node**
(Net Fetch or FTP)



[✓ File Document]
Export with Oracle Names
[✓ Node] Client Profile

**Figure 3 – 5 Configuration Process**

## Know Your Network

To use Network Manager effectively, you must have detailed information about the network at hand. This section describes the information you must have ready.

Network Manager knows the syntax of the configuration files, and it knows the default values for parameters in those files. However, it knows nothing about your network until you supply that information. In fact, supplying accurate information to the utility is your main task in using it.

### Names

Choose names for the following:

- all domains
- all communities
- all network listeners, and the computers (nodes) on which they run
- all databases, to act as database service names. Service names are short identifiers for their connect descriptors

  **Note:** With SQL*Net release 2.1 and the Oracle Server release 7.1 and later, the service names you provide must match precisely the unique global database names assigned by the database administrator. To achieve this, it may be necessary to change some of the service names you have been using.

  For example, if your previously defined SQL*Net version 2 network has service names that do not match the global database names, those service names must be changed. Similarly, if the network includes some databases that were named before you established your current domain names, their global database names and service names must reflect the current domain structure.
- all MultiProticol Interchange nodes (if any)
- all Names servers, if any, and the node(s) on which they run
- all client profiles (A *client profile*, or *client type* is a group of clients with the same communication requirements.)

Even if you have a single–protocol network (a single–community network), you must supply a name for that community.

### Addresses

Define addresses for the following:

- all servers
- all MultiProtocol Interchanges (if any)
- all Oracle Names servers (if any)

The addresses for these components consist of the names of the communities of which they are a part and any protocol–specific information.

## Protocol–Specific Information

Different protocols require different protocol–specific information. The following table summarizes the keywords for the protocols currently supported in a TNS network.

| **Protocol** | **Keywords** |
|---|---|
| AppleTalk | SERVICE |
| TCP/IP | HOST<br>PORT |
| DECnet | NODE<br>OBJECT |
| SPX/IPX | SERVICE |
| NetBIOS | NTBNAME |
| Named Pipes | PIPE<br>SERVER |
| Banyan | ITEM GROUP<br>ORGANIZATION |
| DCE | SERVER_PRINCIPAL<br>SERVICE<br>CELL_NAME |
| OSI4 | NSAP<br>TSEL<br>*or*<br>HOST<br>SERVICE |
| LU6.2/APPC | LU_NAME<br>LOCAL_LOOKUP<br>TP_NAME |
| ASYNC | PHONENUMBER<br>ASYNC_SERVER<br>LOCAL_LOOKUP |

**Table 3 – 1 Protocol–Specific Keywords**

OSDoc

**Additional Information:** Network Manager provides default values for many of these protocol–specific keywords. See your Oracle operating system–specific documentation for information on what values to supply for the protocol keywords.

Connect Data — Servers also require that you provide the system identifiers (SIDs) for their databases.

SQL*Net Version 1 Connect Strings — If your network includes both SQL*Net versions 1 and 2, have available the names of the files that hold the version 1 connect strings and their aliases. Know where in the file system they are stored.

LISTENER.ORA Parameter File — The LISTENER.ORA file includes a number of required and optional parameters that describe the listener. You should gather the parameter information and have it ready to use with Network Manager.

MultiProtocol Interchange Information — Network Manager creates configuration files specific to the MultiProtocol Interchanges in your network, if any, based on information you supply. The configuration files are described in the *Oracle MultiProtocol Interchange Administrator's Guide.*

Names Servers — If you are using Oracle Names, Network Manager creates a NAMES.ORA configuration file for each of the Names servers in your network. See the *Oracle Names Administrator's Guide* for information about this file.

Multiple Network Managers — If your network includes Oracle Names, you may want to provide further information about the naming structure of your network. You may want to include delegated administrative regions, so that widely separated parts of your network have some autonomy in their administration. See the *Oracle Names Administrator's Guide* for more information.

Database Links — If you are using Oracle Names, you may also want to include information about global database links. See "Entering Component Information" in the *Oracle Network Manager Administrator's Guide.*

## Using Network Manager with Oracle Names

If you are creating a new network, you may need to store the network definition in a file, create and distribute the configuration files, and start the network. Once the network is up, you can then store the network definition in the database used by Oracle Names.

## Distributing Configuration Files

On systems running Microsoft Windows 3.1, once the initial configuration has been generated through Network Manager and SQL*Net connections established, the administrator can easily distribute changes and updates to the network configuration through the included NETFETCH utility. See the *Oracle Network Manager Administrator's Guide* for details. Note that, non–Windows systems must distribute the configuration files manually

From the destination node, NETFETCH retrieves the appropriate network configuration from Network Manager.

**Suggestion:** NETFETCH can be run during Windows startup so that the destination node's configuration can be kept up to date automatically. .

The utility is intended to be used by either the DBA or network administrator, and is usually used in only one location. Choose a workstation from which it will be relatively easy to transfer files.

After you have created and distributed the configuration files, you can start and use the network.

## Testing your Configuration

After configuring the network using Network Manager and distributing the configuration files to the destination machines, each component can be started and tested.

The preferred sequence for testing the network is to:

- start and test each Listener
- start and test each MultiProtocol Interchange
- test each client type

In addition, if Oracle Names is running on your network, test Oracle Names. For further information on how to do so, see the *Oracle Names Administrator's Guide.*

### Start the Listener

From each listener's node, use the Listener Control Utility, LSNRCTL, to start each listener. In command line mode, the command is:

```
LSNRCTL START listener_name
```

LSNRCTL should display a status message indicating that it has started the listener process successfully. Check that all expected SIDs for that listener are listed in the services summary in the status message.

## Test the Listener

To test the listener, initiate a connection from a client in the same community as the listener to any active database controlled by that listener.

The simplest test uses SQL*Plus as follows:

```
SQLPLUS user/password@service_name
```

The *service_name* is found in the TNSNAMES.ORA file before each entry. For more information about testing from a client, see page 3 – 17.

If there are no clients in the same community as the listener, you must start an Interchange before testing the listener.

Repeat these steps for each listener in the network.

## Start the MultiProtocol Interchanges

Use the Interchange Control Utility, INTCTL, on the Interchange node to start an Interchange. For example, in command line mode the command is:

```
INTCTL START INTERCHANGE
```

or

```
INTCTL START INT
```

INTCTL should display a status message indicating that it has started the interchange successfully. For more information on the Interchange Control Utility, see the *Oracle MultiProtocol Interchange Administrator's Guide.*

## Test the Interchanges

To test the Interchange, initiate a connection through the Interchange from a client in one community to a database in another. On the client machine type:

```
SQLPLUS user/password@service_name
```

The *service_name* for the database in the other community is found in the TNSNAMES.ORA file. To ensure that the connection went through the Interchange, type:

```
INTCTL STATUS INTERCHANGE
```

The Connection Manager should indicate that there is one active connection through the Interchange. The STATUS command can be run on the Interchange machine, or, if the Interchange is listed in the TNSNAMES.ORA file, from any other node.

## Test Each Client Type to Make Sure you can Connect to Servers

Make sure that each client type is tested. It is not enough to test that the Interchange works. If there are several different client types in your network, initiate a connection to a server from each of them.

If a connection is unsuccessful, use logging and tracing, the diagnostic utilities, to find the cause of the problem. An error stack in the error log may point to the problem. If not, turn on tracing and repeat the operation. You can find information about error logging and tracing in the *Oracle Network Products Troubleshooting Guide.*

## Common Errors During Testing

If you are unsuccessful in bringing up a listener or Interchange, or fail to make a connection to a database, one of the following common errors may be the cause.

- The listener name in the LSNRCTL START command is invalid. Check for typos. Check the LISTENER.ORA file to ensure that the listener name you are using is valid.
- Files are in the wrong place. Both the listener and the Interchange will indicate that they cannot start because configuration files could not be found.

OSDoc

**Additional Information:** See your operating system–specific documentation to see that the LISTENER.ORA file has been placed correctly for the listener, and the INTCHG.ORA, TNSNAV.ORA, and TNSNET.ORA files are placed correctly on the Interchange. Be sure that the TNSNAMES.ORA file you access is the one created by Network Manager.

- A specified address is already in use. Another process may already be using the address listed in LISTENER.ORA. On some protocols such as TCP/IP, DECnet, and OSI, each network service on a node must use a unique port or socket. On other network protocols such as SPX/IPX or NetBIOS, each network service name must be unique for the entire network. Another network service may be using the same configuration. Contact your network administrator to evaluate whether the network address is available.
- When trying to connect to a database, you may get the message ORA–12203: "TNS: Unable to connect to destination." Use the LSNRCTL utility to start the listener on the server machine. See *Understanding SQL*Net* for further information.

- When trying to make a connection from a client, you may get the message ORA–12154:”TNS:Could not resolve service name.” The service name you requested is not listed in the TNSNAMES.ORA file, or the TNSNAMES.ORA file has been placed incorrectly. See *Understanding SQL*Net* for more information.
- When trying to connect to a database, you may get the message ORA–1034: “Oracle Not Available”. The database is not running on the server machine. A listener alone does not provide a database connection, the database instance must also be started.

Other common errors are listed in *Understanding SQL*Net.* All the error messages generated by SQL*Net, the MultiProtocol Interchange, Oracle Names, and Network Manager (and their underlying layers) can be found in the *Oracle Network Products Troubleshooting Guide.* This book also contains information about how to interpret log files and how to use the trace facility for trouble–shooting purposes.

## Initiating a SQL*Net Connection

There are a number of ways to initiate a connection with an Oracle server. Commonly used methods are:

- the operating–system command line
- a utility’s logon screen
- a 3GL program
- special commands within certain utilities

The specifics of use are slightly different in each case. Each of the general methods listed is briefly covered here. To identify the method used by a specific utility, see that utility’s user’s guide.

### Connecting from the Operating–System Command Line

The general form of connecting an application to a database server from the command line is:

```
tool username/password@service_name
```

where:

| | |
|---|---|
| *tool* | Specifies the command used to invoke a utility such as SQL*Plus, SQL*DBA, SQL*Forms, etc. |
| *username* | Specifies an Oracle username on the server. |

| | |
|---|---|
| *password* | Specifies the corresponding password on the server. |
| *service_name* | Specifies a service name entered in Oracle Names or the TNSNAMES.ORA file that identifies the connect descriptor for the desired server. If the server is in the client's default domain, the service name does not need to include the domain name. However, if the server is in another domain, the service name must include the domain. (The default domain is determined by a parameter in the client's SQLNET.ORA file. |

For example, in a network with only one domain, the default .WORLD domain, it is not necessary to include ".WORLD" in the service name.
For example:

```
% sqlplus scott/tiger@SERVERX
```

However, if the client's default domain were .EAST and the server's domain were .WEST, then the service name would have to include the domain. For example,

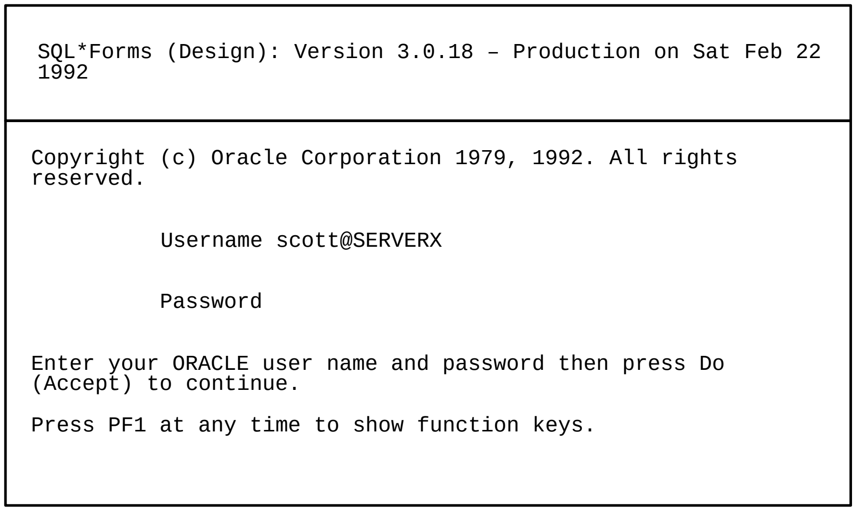
```
% sqlplus scott/tiger@SERVERX.WEST
```

**Note:** To prevent a password from displaying during a logon, you can omit the password parameter on the command line. You will then be prompted to enter your password. It will not be displayed as you enter it

Most Oracle utilities can use the operating–system command line to connect. Others provide alternatives.

### Connecting from the Utility Logon Screen

Some utilities provide a logon screen as an alternative form of logon. A user can log on to a database server just as easily by identifying both the username and service name in the username field of the utility logon screen, and typing the password as usual in the password field. Figure 3 – 6 shows a SQL*Forms logon screen where the user SCOTT is connecting to the server SERVERX with a password of TIGER. Notice the password cannot be seen, a standard feature of Oracle utilities' logon screens.



```
SQL*Forms (Design): Version 3.0.18 – Production on Sat Feb 22
1992
```

```
Copyright (c) Oracle Corporation 1979, 1992. All rights
reserved.

          Username scott@SERVERX

          Password

Enter your ORACLE user name and password then press Do
(Accept) to continue.

Press PF1 at any time to show function keys.
```

**Figure 3 – 6  Connection from Logon Screen**

## Connecting from a 3GL Application

In applications written using a 3GL, the program must establish a connection to a server using the following syntax:

```
EXEC SQL CONNECT :username IDENTIFIED BY :password
```

In this connection request, the *:username* and :*password* are 3GL variables that can be set within the program either statically or by prompting the user. When connecting to a database server, the value of the :*username* variable is in the form:

```
username@service_name
```

which is the same as in the logon screen above. The :*password* variable contains the password for the database account being connected to.

## Connecting Using Special Commands within Utilities

Some Oracle utilities have internal commands for database connection, once the utility has been started, that allow an alternative username to be specified without leaving the utility. Both SQL*Plus and SQL*DBA allow the CONNECT command using the following syntax:

```
SQL> CONNECT username/password@service_name
```

For example:

```
SQL> CONNECT SCOTT/TIGER@SERVERX
```

This is very similar to the operating–system command–line method, except that it is entered in response to the utility’s prompt instead of the operating–system prompt.

Other Oracle utilities use slightly different methods specific to their function or interface. For example, Oracle CDE utilities use logon buttons and a pop–up window with the username, password, and remote database ID field. For more information on connecting to Oracle with a specific utility, see that utility's user guide.

**CHAPTER**

*4*

# Database Administration

This chapter discusses issues of concern to the database administrator (DBA) implementing or maintaining databases in a distributed system. Topics covered include:

- issues to consider before implementing a distributed system
- setting up public and private database links
- Oracle database management utilities

The database administrator must also be responsible for implementing distributed update recovery management (see Chapter 5) and basic or advanced replication features. See *Oracle7 Server Distributed Systems, Volume II.*

## Implementing Databases in a Distributed System

There are a number of factors the DBA should consider before implementing databases in a distributed system. Many of the planning decisions will require the cooperation and transfer of information between the network administrator and the DBA.

OSDoc

There are also operating system–specific issues the DBA must consider when planning to distribute databases on certain platforms. See your Oracle operating system–specific documentation.

**Attention:** Some of the issues and tasks listed in this section are specific to the SunOS platform, and are described here as a sample list of things to consider when implementing a distributed system.

### Purpose of the Database

The purpose and size of the database will determine how you plan and structure the database.

For example, an online transaction processing (OLTP) database, such as a bank automated teller machine (ATM) has a high volume of transactions. Therefore when planning an OLTP database, you may need to distribute I/O across multiple disks and controllers. You usually need to split the logical database design.

A decision support database, such as an inventory system, has a relatively low number of database updates (measured in transactions per hour). Also, users tend to make few queries and may look at results of these queries for many minutes at a time. Thus a decision support database has less need to distribute I/O across multiple disks and controllers.

#### Size of the Database

Consider the size of a machine relative to the database that will run on it. For example, keep in mind that a machine with more physical memory and more processors can support a larger database. A large database (typically over one Gb) is more likely to need to split I/O and the logical database design. It is more likely to need larger initialization file parameter values.

## Physical and Logical Layout of the Database

You need to determine several aspects of database physical layout. For example, you need to estimate the number of disks and controllers required for optimal performance from your Oracle7 system. Many smaller disks tend to give better performance than a few larger disks.

When planning the logical layout of your database, consider how tables should be split up among tablespaces. A larger number of smaller tablespaces tend to be more flexible than a few catchall tablespaces.

Keep in mind that different sites have different needs, and decisions should be based on a particular site's needs for a specific application.

## File Locations and Initialization Parameters

The network or database administrator must determine locations for database files. For example, for Oracle on a SunOS platform, it is recommended that database files be stored according to the Oracle Optimal Flexible Architecture (OFA). OFA is a specification for configuring Oracle systems at sites demanding high performance with low maintenance under continually evolving requirements. The OFA makes configuration recommendations regarding aspects of your operating system, such as mount points, login home directories, and user profiles. It also makes configuration recommendations regarding Oracle software and administrative files, and database files.

OSDoc

The initialization file, INIT.ORA, contains certain default parameter values, which the DBA may need to increase for optimal performance. Many of theses values are operating system–specific and depend on multiple factors affecting the entire distributed system. See your Oracle operating system–specific documentation for information about your specific requirements.

### Backup Strategies

The purpose of the database and how frequently it is used determine which backup methods are chosen, as well as the frequency of backups. The DBA must develop a plan to produce and store archive tapes for each database server in the distributed system, or his region of responsibility, if responsibility for parts of the distributed system is divided between several DBAs.

## Memory Requirements

When designing your database, consider the following:

- the larger the number of different applications running on the server, the less physical memory is available for Oracle.
- a server with all clients running remotely can (and should) have a larger SGA than one with only local clients.

OSDoc

For more information about these issues, see your operating system–specific documentation.

### Relinking Product Executables

Most Oracle products provide relinkable executables. Relinking lets you regenerate a program from its component parts. Relinking also lets you add options to the Oracle7 Server, such as the distributed option, PL/SQL, and Oracle (SQL*Net) Protocol Adapters.

Relinking relies on operating–system facilities that must be installed and usable before you can perform relinking successfully. For example, on the SunOS system, the commands, **make** and **ar** must be present, and the system libraries must be available.

Most sites must relink during installation. For example, you *must* relink when installing the distributed option or any Oracle (SQL*Net) protocol adapters.

## Views and Location Transparency

Local views can provide location transparency for local and remote tables in a distributed system.

For example, assume that table EMP is stored in a local database. Another table, DEPT, is stored in a remote database. To make the location of, and relationship between, these tables transparent to users of the system, a view named COMPANY can be created in the local database that joins the data of the local and remote servers:

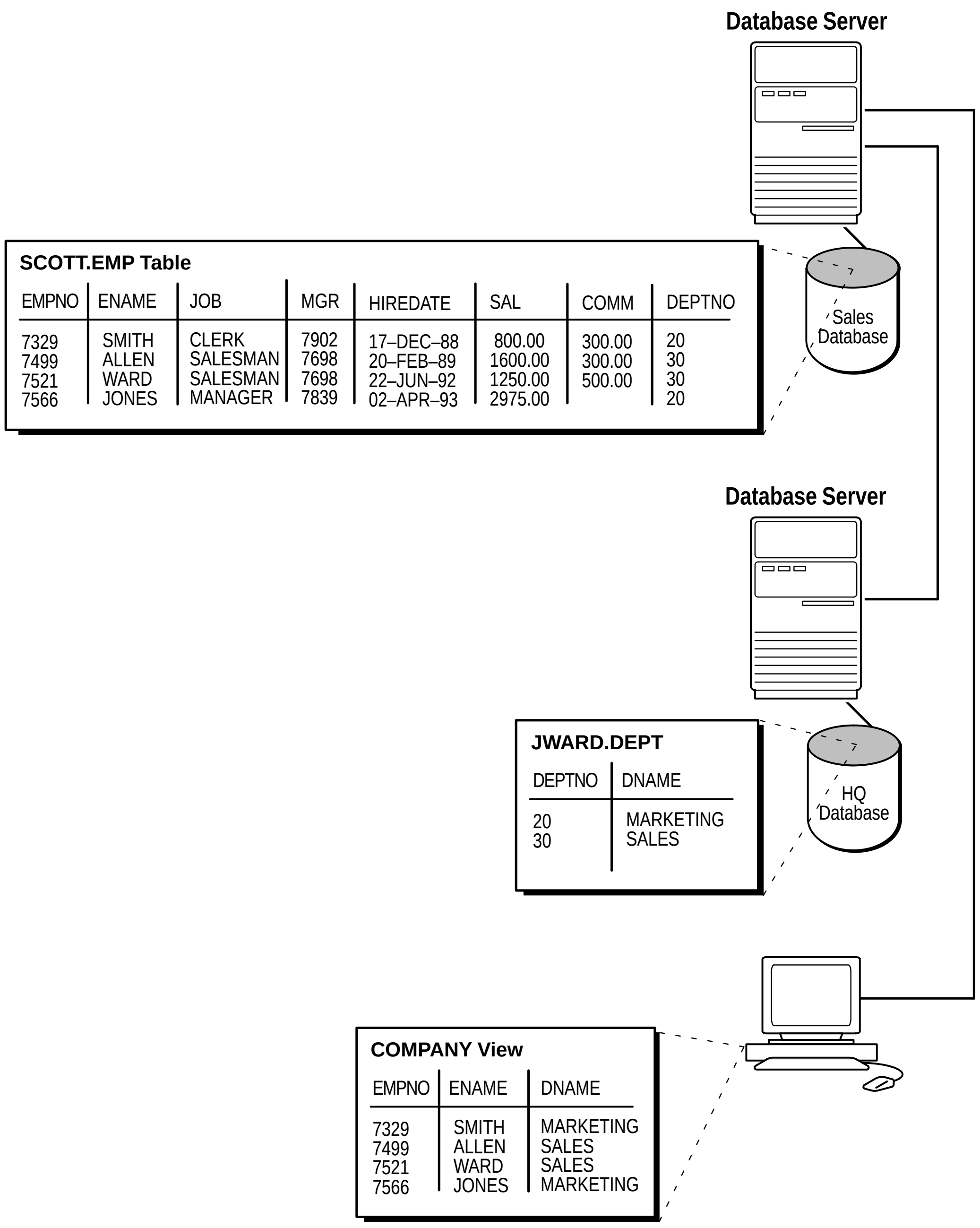
```
CREATE VIEW company AS
   SELECT empno, ename, dname
   FROM scott.emp a, jward.dept@hq.acme.com b
   WHERE a.deptno = b.deptno;
```

When users access this view, they do not know, or need to know, where the data is physically stored, or if data from more than one table is being accessed. Thus, it is easier for them to get required information.
For example:

```
SELECT * FROM company;
```

provides data from both the local and remote database table.

Figure 4 – 1 illustrates this example of location transparency.



**SCOTT.EMP Table**

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|
| 7329 | SMITH | CLERK | 7902 | 17–DEC–88 | 800.00 | 300.00 | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 20–FEB–89 | 1600.00 | 300.00 | 30 |
| 7521 | WARD | SALESMAN | 7698 | 22–JUN–92 | 1250.00 | 500.00 | 30 |
| 7566 | JONES | MANAGER | 7839 | 02–APR–93 | 2975.00 |  | 20 |

**JWARD.DEPT**

| DEPTNO | DNAME |
|---|---|
| 20 | MARKETING |
| 30 | SALES |

**COMPANY View**

| EMPNO | ENAME | DNAME |
|---|---|---|
| 7329 | SMITH | MARKETING |
| 7499 | ALLEN | SALES |
| 7521 | WARD | SALES |
| 7566 | JONES | MARKETING |

**Figure 4 – 1 Views and Location Transparency**

## Synonyms

Synonyms are very useful in both distributed and non–distributed environments because they hide the identity of the underlying object, including its location in a distributed system. If the underlying object must be renamed or be moved, only the synonym needs to be redefined; applications based on the synonym continue to function without modification. Synonyms can also simplify SQL statements for users in a distributed system.

Database synonyms are a standard SQL feature that provide alternate names for database objects and, optionally, their locations. A synonym can be created for any table, view, snapshot, sequence, procedure, function, or package. All synonyms are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym can allow single–word access to remote data, isolating the specific object name and the location from users of the synonym. The syntax to create a synonym is:

```
CREATE [PUBLIC] SYNONYM_name
FOR [schema.]object_name[@database_link_name]
```

where:

| | |
|---|---|
| [PUBLIC] | Specifies that this synonym is available to all users. Omitting this parameter makes a synonym private, and usable only by the creator. Public synonyms can be created only by a user with CREATE PUBLIC SYNONYM system privilege. |
| *synonym_name* | Specifies the alternate object name to be referenced by users and applications. |
| *schema* | Specifies the schema of the object specified in object_name. Omitting this parameter uses the creator's schema as the schema of the object. |
| *object_name* | Specifies either a table, view, sequence, or other name as appropriate. |
| *database_link_name* | Specifies the database link which identifies the remote username in which the object specified in object_name is located. |

A synonym must be a uniquely named object for its schema. If a schema contains a database object and a public synonym exists with the same name, Oracle always finds the database object when the user that owns the schema references that name.

Because a synonym is a reference to the actual object, the security domain of the object is used when the synonym is accessed. For example, a user that has access to a synonym for a specific table must also have privileges on that table to access the data in it. If the user attempts to access the synonym, but does not have privileges on the table it identifies, an error occurs indicating that the table or view does not exist.

## A Simple Example

Assume that in every database in a distributed system, a public synonym is defined for the SCOTT.EMP table stored in the HQ database:

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.acme.com;
```

An employee management application can be designed without regard to where the application is used because the location of the EMP table is hidden by the public synonyms. SQL statements in the application access the table SCOTT.EMP@HQ.ACME.COM by referencing the public synonym EMP.

Furthermore, if the EMP table is moved from the HQ database to the HR database, only the public synonyms need to be changed on the nodes of the system. The employee management application continues to function properly on all nodes.

## A More Complex Example

Figure 4 – 2 shows two servers, OHIO and NY_FIN, in which a database link from OHIO to NY_FIN and the synonym FOR_SALE provide an alternate object name for use in OHIO to reference the OPEN table in NY_FIN. The database link and the synonym are created as follows:

```
CREATE PUBLIC DATABASE LINK NY_FIN
CONNECT TO REAL_ESTATE IDENTIFIED BY NOPASS;
USING 'NY_FIN'
CREATE PUBLIC SYNONYM FOR_SALE
FOR OPEN@NY_FIN;
```

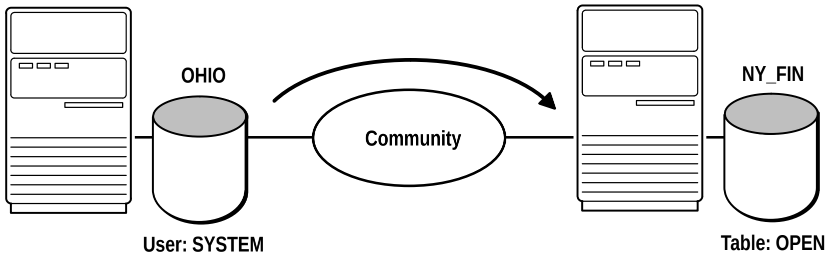


**Figure 4 – 2 Using Synonyms for Alternate Object Names**

The table OPEN on NY_FIN could be accessed from OHIO using the SQL statement:

```
SELECT * FROM FOR_SALE;
```

Using this database link, the user is logging on to NY_FIN as user REAL_ESTATE. Notice that this public synonym was created by the DBA on behalf of the REAL_ESTATE username. Without such a prefix, a table that does not exist in the database link user's schema would return an error, because it would be looking for the OPEN table owned by the REAL_ESTATE user.

## Relocating a Table

If the OPEN table in the example above were to be moved from one database server to another, only the synonym or the database link would need to be changed to identify the new location. The applications would continue to reference the same object name, although they would be connecting to a new location to access the data in that table. Figure 4 – 3 shows the most common method of redefining the location of a table to retain location transparency. The command would be:

```
CREATE PUBLIC DATABASE LINK NY_TAX
CONNECT TO REAL_ESTATE IDENTIFIED BY NOPASS;
DROP PUBLIC SYNONYM FOR_SALE;
CREATE PUBLIC SYNONYM FOR_SALE
FOR OPEN@NY_TAX;
```

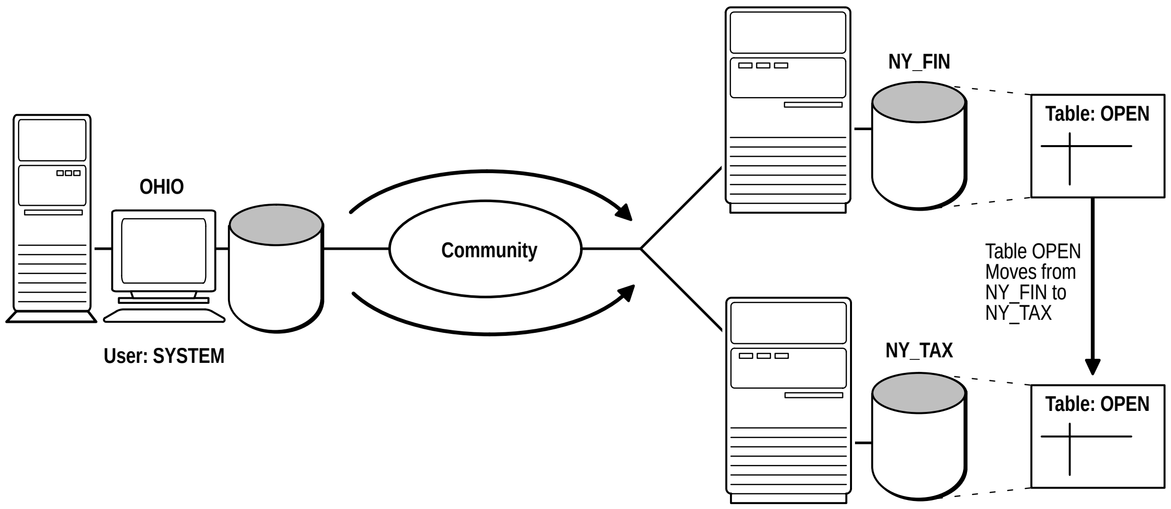


**Figure 4 – 3 Redefining Table Location to Retain Location Transparency**

To relocate the table, a second database link was created called NY_TAX that connected to a new database with the service name NY_TAX, and the synonym was recreated to reference the NY_TAX database link instead of the NY_FIN database link. Other tables that were accessed through the NY_FIN database link to NY_FIN would continue to function properly.

## Replication Transparency

The ability to insure reliable data replication is an extremely important (and potentially complex) factor in a distributed system. Data replication means that any given data object can have several stored representatives at several different sites and that, if each representative is potentially updatable, there must be a mechanism for insuring that all representatives reflect the changes.

Oracle7 provides two mechanisms for accomplishing transparent table replication in a distributed system. The basic replication that comes with the distributed option provides *asynchronous* table replication through snapshots (changes to an updatable master table are replicated on the read–only or updatable snapshot tables only at timed intervals). Alternatively the advanced replication option allows you to implement *synchronous* table replication (changes to any table are applied to all replicated copies immediately) through the Oracle Symmetric Replication Facility. In either case, table replication is transparent to users making changes to replicated tables.

See *Oracle7 Server Distributed Systems, Volume II* for detailed information about all forms of replication.

## Procedures and Location Transparency

Location transparency is also provided by PL/SQL program units called *procedures* that contain SQL statements that reference remote data. For example, consider the procedure created by the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
    DELETE FROM jward.emp@hq.acme.com
      WHERE empno = enum;
END;
```

When a user or application calls the FIRE_EMP procedure, it is not apparent that a remote table is being modified.

A second layer of location transparency is possible if the statements in a procedure indirectly reference remote data using local procedures, views, or synonyms. For example, the following statement defines a local synonym:

```
CREATE SYNONYM emp FOR jward.emp@hq.acme.com;
```

Consequently, the FIRE_EMP procedure can be defined with the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
   DELETE FROM emp WHERE empno = enum;
END;
```

If the table JWARD.EMP@HQ is renamed or moved, only the local synonym that references the table needs to be modified. None of the procedures and applications that call the procedure require modification.

## Query and Update Transparency

Oracle allows the following standard DML statements to reference remote data:

- SELECT (queries)
- INSERT
- UPDATE
- DELETE
- SELECT...FOR UPDATE
- LOCK TABLE

A query, including joins, aggregates, subqueries, and SELECT ... FOR UPDATE, can reference any number of local and remote tables and views. For example, the following query joins information from two remote tables:

```
SELECT empno, ename, dname
  FROM scott.emp@sales.acme.com a,
    jward.dept@hq.acme.com b
  WHERE a.deptno = b.deptno;
```

UPDATE, INSERT, DELETE, and LOCK TABLE statements can reference both local and remote tables. No programming is necessary to update remote data. For example, the following statement inserts new rows into the remote table SCOTT.EMP in the SALES database by selecting rows from the JWARD.EMP table in the local database:

```
INSERT INTO scott.emp@sales.division3.acme.com
  SELECT * FROM jward.emp;
```

### Restrictions

Within a single SQL statement, all referenced LONG and LONG RAW columns, sequences, updated tables, and locked tables must be located at the same node. Oracle does not allow remote Data Definition Language (DDL) statements (for example, CREATE, ALTER, and DROP). Also, the LIST CHAINED ROWS clause of an ANALYZE statement cannot reference remote tables.

## Transaction Transparency and Distributed Transaction Management

Transactions in Oracle, single–site or distributed, generally are terminated with a COMMIT or ROLLBACK statement. SAVEPOINT and ROLLBACK TO SAVEPOINT statements are also supported by the Oracle distributed architecture.

If a transaction is a single–site transaction, it simply commits or rolls back. If a transaction is distributed, Oracle's distributed transaction management mechanism is automatically used to commit it. Oracle's recovery management mechanism guarantees that the nodes referenced in a distributed transaction either all commit or all roll back the transaction, even if a network failure occurs while the transaction is being committed. For detailed information about Oracle's distributed transaction management mechanism, see page 5 – 4.

## Failure Resolution Transparency and the RECO Background Process

Network or computer hardware failure is always possible. Oracle's distributed system architecture guarantees that if a network or system failure occurs during the commit of a distributed transaction, the transaction is automatically and transparently resolved globally. When the network or system is restored, either all nodes commit, or all roll back.

The RECO background process automatically recovers *pending* or *in–doubt* distributed transactions (transactions not committed or rolled back when a network failure occurs during a two–phase commit process). When a network failure is corrected, the RECO processes of the involved database servers automatically resolve the outcome of any pending distributed transactions. No work is required of the database administrator to resolve pending transactions, although the database administrator may force resolution of in–doubt transactions if he cannot wait for the network failure to be corrected. For information about failure resolution of in–doubt distributed transactions in an Oracle distributed system, see the *Oracle7 Server Concepts* manual.

## Performance Transparency

All of Oracle's underlying performance optimizations are transparent in a distributed system, including shared SQL, cost–based SQL statement optimization, Oracle's array interface (for fetch and insert), and PL/SQL (for reduced network traffic).

## Object Resolution and Location Transparency

When defining views, synonyms, and procedures that reference remote objects, consider the following issues so that your views, synonyms, and procedures always access the intended objects with the intended privileges:

- Unless all remote objects are explicitly qualified with their schema names, the accessed object is dependent on the connection established via the specified database link. For example, if you want a local synonym's remote base table to be SCOTT.EMP, explicitly specify schema SCOTT in the synonym definition:

  ```
  CREATE SYNONYM emp FOR scott.emp@hq.acme.com;
  ```

- Database links defined with a CONNECT TO clause always connect to the remote database as the user specified in the CONNECT TO clause. Views, synonyms, and procedures that use this type of database link are always subject to the privileges and schema resolution of the specified remote user.

  Alternatively, when database links without a CONNECT TO clause are used in the definition of a view, synonym, or procedure, the remote connection is dependent on the local user.

Views and procedures should typically use database links with explicit accounts to ensure that they can function with the proper remote authorizations.

## Schema Object Names and Data Access

Oracle allows schema objects (for example, tables, views, and procedures) throughout a distributed system to be referenced in SQL statements using global object names. In Oracle, a schema object's global name consists of the name of the schema that contains the object, the object name, followed by an "at" sign (@), and a database name. For example, the following query selects information from the table named SCOTT.EMP in the SALES database:

```
SELECT * FROM scott.emp@sales.division3.acme.com;
```

Oracle does not check, nor enforce, unique global object names when an object is created. Oracle does not store complete global object names in the distributed data dictionaries. However, Oracle does guarantee that an object name is unique within its own local database. Additionally, a distributed system can be configured so that each database within the system has a unique database name, thereby providing unique global object names.

## Balancing Location Transparency and Security

The choice of using a view, synonym, or procedure for location transparency determines the degree to which the local and remote administrators are responsible for object security. The following example statements and sections outline the issues to consider when choosing among the options for location transparency.

**Statement issued at remote database:**

```
GRANT SELECT, DELETE ON scott.emp TO user1;
```

**Statements issued at local database:**

```
CREATE DATABASE LINK hr.acme.com
   CONNECT TO user1 IDENTIFIED BY password
   USING 'db_string';
CREATE VIEW admin.emp_view AS
   SELECT * FROM scott.emp@hr.acme.com;
CREATE PROCEDURE admin.fire_emp (enum NUMBER) AS
BEGIN
   DELETE FROM scott.emp@hr.acme.com
      WHERE empno = enum;
END;
CREATE SYNONYM admin.emp_syn FOR scott.emp@hr.acme.com;
```

## Privilege Management With Views

Assume a local view (ADMIN.EMP_VIEW) references a remote table or view. The owner of the local view can grant only the object privileges on his view that have been granted the remote user referenced in the database link. This is similar to privilege management for views that reference local data. Therefore, local privilege management is possible when views are used for location transparency. For example, the user ADMIN can successfully grant the SELECT and DELETE privileges, but not the INSERT and UPDATE privileges for EMP_VIEW.

Views are a good choice for location transparency if unlimited local object privilege management is a requirement. For example, assume the local security administrator needs to selectively grant object privileges for several remote tables. The remote administrator can create a powerful user account that is granted many privileges for many remote tables. Then, the local administrator can create a private database link that connects to the powerful remote account and, in the same schema, create views to "mirror" the remote tables. The local administrator controls local privilege management for the remote tables by granting privileges on the local views. Also note that in this example, many users can use a private database link.

## Privilege Management With Procedures

Assume a local procedure includes a statement that references a remote table or view (see example on previous page). The owner of the local procedure can grant the EXECUTE privilege to any user, thereby giving that user the ability to execute the procedure and access remote data.

In general, procedures aid in security. Users who call a procedure can only perform the controlled operations of the procedure. Privileges for objects referenced within a procedure do not need to be explicitly granted to the calling users. Much like views, procedures are a good choice for location transparency if unlimited local privilege management is a requirement.

For example, assume the local security administrator needs to selectively allow users to query and update several remote tables. The remote administrator can create a powerful user account that grants many object privileges. Then, the local administrator can create a private database link that connects to the powerful remote account and, in the same schema, create procedures to query and modify the remote tables, as desired. The local administrator can control how local users can access the remote tables by selectively granting the EXECUTE privilege for the local procedures, thus controlling local privilege management for remote objects.

### Privilege Management With Synonyms

Assume a local synonym is an alias for a remote object. The owner of the local synonym *cannot* grant any object privileges on the synonym to any other local user. This behavior is different from privilege management for synonyms that are aliases for local tables or views; in the case where a synonym is an alias for a remote object, local privileges for the synonym cannot be granted because this would amount to granting privileges for the remote object, which is not allowed. Therefore, no local privilege management can be performed when synonyms are used for location transparency; security for the base object is controlled entirely at the remote node. For example, the user ADMIN cannot grant any object privileges for the EMP_SYN synonym.

Unlike a database link referenced in a view or procedure definition, a database link referenced in a synonym is resolved by first looking for a private link owned by the schema in effect at the time the reference to the synonym is parsed. Therefore, to ensure the desired object resolution, it is especially important to specify the underlying object's schema in the definition of a synonym.

## Public and Private Database Links

Public and private database links are typically created by DBAs or individual users. Global database links are typically created by a network administrator. This section discusses some topics related to public and private database links, such as creating, viewing, and dropping database links from the data dictionary. For general information on database links, see page 2 – 29.

## Manually Creating Database Links

The DBA and application user typically create public and private database links manually. Global database links are created automatically for every database defined in Network Manager by a network administrator or DBA.

This section discusses how public and private links are created. For information on how global database links are defined, see “Creating Global Database Links in Network Manager” on page 2 – 33.

The syntax for creating public and private links is:

```
CREATE [PUBLIC] DATABASE LINK linkname
[CONNECT TO username IDENTIFIED BY password]
USING 'service_name'
```

In this syntax:

| | |
|---|---|
| [PUBLIC] | Specifies a database link available to all users with the CREATE SESSION privilege. If the PUBLIC option is omitted, a private link available only to the creator is created. Note that creating a public database link requires CREATE PUBLIC DATABASE LINK privilege. |
| *linkname* | Specifies the name of the database link. If the remote server is in the local server’s domain, the link name does not need to include the domain name. However, if the server is in another domain, the link name must include the domain. (The domain is determined by DB_DOMAIN in the initialization parameter file). |
| CONNECT TO | Optionally specifies a single username and password for all users of the database link to share. If the clause is omitted, the Oracle username and password of the user account using the database link will connect to the remote database server. |
| *username* | Specifies a valid Oracle username on the remote database server. |
| *password* | Specifies the corresponding password of the username on the remote database server. |

| | |
|---|---|
| *service_name* | Specifies the service name defined in the TNSNAMES.ORA file or stored in Oracle Names associated with the connect descriptor for the desired database. If the remote server is in the local server's default domain, the service name does not need to include the domain name. However, if the server is in another domain, the service name must include the domain. (The default domain is determined by a parameter in the server's SQLNET.ORA file. |

Before Oracle7, a database administrator could specify any linkname for a database link. However, with Oracle7 and later releases, a database link must have the same name as the global database name of the database. Remember that the service name is also the same as the global database name. Therefore, the linkname and service name are now the same. Although this may seem to make the USING clause redundant, it is still a necessary part of the syntax.

For example, the command for creating a public database link to a database that has the global database name ORCHID.HQ.ACME is as follows:

```
CREATE PUBLIC DATABASE LINK ORCHID.HQ.ACME
CONNECT TO scott IDENTIFIED BY tiger
USING 'ORCHID.HQ.ACME'
```

The following statement is the complete CREATE DATABASE LINK statement shown earlier. This example illustrates the creation of the SALES database link and the complete path that is specified for the link:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
    CONNECT TO guest IDENTIFIED BY password
    USING 'dbstring';
```

When a database link is created, a *complete path* (the remote account and the database string), a *partial path* (just the remote account or just the database string), or no path can be specified.

When a SQL statement references a global object name in the SALES database, the local Oracle node finds the corresponding SALES database link in the local database and attempts to establish a session in the remote database for the user GUEST/PASSWORD. The database string specified in the SALES database link definition (which is operating system and network dependent) is used to facilitate the remote connection.

Typically, it is the responsibility of each database administrator or application administrator to create the necessary database links to databases throughout the network. Database links are an implementation detail that should be completely transparent to applications and end–users of a database. It should appear to applications and users that a remote table is accessed by specifying the table's global object name, not by referencing an available database link. In fact, administrators should create location transparency for remote objects using views, synonyms, or procedures, so that applications do not explicitly reference remote data. Then, if the remote object is moved, only the synonym needs to be altered, not all applications.

## Public Database Links with a Default Connection

Figure 4 – 4 shows a public database link created by the DBA user SYSTEM using the service name NY_FIN.HQ.ACME. The link is created by entering:

```
CREATE PUBLIC DATABASE LINK NY_FIN.HQ.ACME
USING 'NY_FIN.HQ.ACME'
```

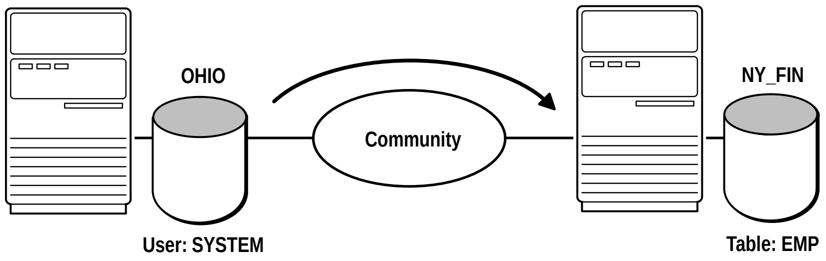


**Figure 4 – 4 Public Database Link with Default Connection**

Users connected to OHIO.SALES.ACME can use the NY_FIN.HQ.ACME database link to connect to NY_FIN.HQ.ACME with the same username and password they have on OHIO.SALES.ACME. To access the table on NY_FIN.HQ.ACME called EMP, any user could issue the SQL query:

```
SQL> SELECT * FROM EMP@NY_FIN.HQ.ACME;
```

**Note:** If the target database were in the source database's default domain, the user would not need to include the domain in the link name or service name, or in the SELECT command.

This query would initiate a connection from OHIO to NY_FIN with the current username and password to log onto NY_FIN. The query would then be processed on NY_FIN. The data available to the current user from the table EMP would be returned to OHIO. Each user creates a separate connection to the server. Subsequent queries to that database link by that user would not require an additional logon.

## Public Database Links with a Specific Connection

Figure 4 – 5 shows the database link created by the user SYSTEM with the service name NY_FIN:

```
CREATE PUBLIC DATABASE LINK NY_FIN
CONNECT TO FINPUBLIC IDENTIFIED BY NOPASS
USING 'NY_FIN'
```

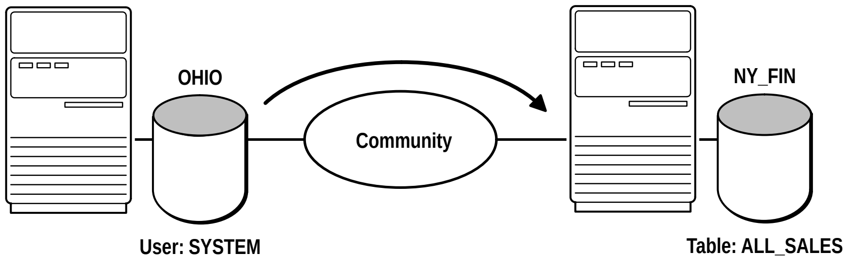


**Figure 4 – 5 Public Database Link with Specific Connection**

Any user connected to OHIO can use the NY_FIN database link to connect to NY_FIN with the common username / password of FINPUBLIC / NOPASS. To access the table in the FINPUBLIC account of NY_FIN called ALL_SALES, any user could issue the SQL query:

```
SQL> SELECT * FROM ALL_SALES@NY_FIN;
```

This query initiates a connection from OHIO to NY_FIN to the common account FINPUBLIC. The query is processed on NY_FIN, and data from the table ALL_SALES are returned to OHIO.

Each user creates a separate connection to the common account on the server. Subsequent queries to that database link by that user would not require an additional logon.

## Connection Qualifiers

You can also define *connection qualifiers* to database links. Connection qualifiers provide a way to create more than one link to a given database. Alternate links are a useful way to access different accounts on the same database with different sets of access privileges. The alternate link created by a connection qualifier must include a reference to a database by its global database name (or service name).

A connection qualifier contains a qualifier name and, optionally, a username and password. To create a connection qualifier, use a statement similar to the following:

```
CREATE PUBLIC DATABASE LINK NY_FIN@PROFITS
CONNECT TO ACCOUNTS IDENTIFIED BY TAXES
USING 'NY_FIN'
```

To use the connection qualifier, you append the qualifier name to the service name of the database you want to access.

For example, the following SQL queries use three database links to the same database, using different connection qualifiers:

```
SELECT * FROM EMP@NY_FIN;
SELECT * FROM SCHEDULE@NY_FIN@PROFITS;
SELECT * FROM EMPSALARIES@NY_FIN@FIN;
```

In this example @PROFITS and @FIN are connection qualifiers.

## Dropping a Database Link

You can drop a database link just as you can drop a table or view. The command syntax is:

```
DROP DATABASE LINK linkname;
```

For example, to drop the database link NY_FIN, the command would be:

```
DROP DATABASE LINK NY_FIN;
```

## Examining Available Database Links

The data dictionary of each database stores the definitions of all the database links in that database. The USER / ALL / DBA_DB_LINKS data dictionary views show the database links that have been defined.

For example, assume that the local database's global name is MKTG.ACME.COM. Also assume that the following CREATE DATABASE LINK statements have been issued by the same user:

```
CREATE DATABASE LINK hq.acme.com
   CONNECT TO guest IDENTIFIED BY password

CREATE DATABASE LINK sales USING 'dbstring';
```

The following query lists all of the private database links contained in the schema associated with the current user issuing the query:

```
SELECT db_link, username, host
   FROM user_db_links;
```

For example, if the user that owns the previously created database links (HQ and SALES) issues the query above, results similar to those below are returned:

```
DB_LINK            USERNAME  HOST
-----------------  --------  ----------
HQ.ACME.COM        GUEST
SALES.ACME.COM               dbstring
```

Notice that the USERNAME and HOST fields can be null if database link definitions do not indicate complete paths to the remote database.

### Finding Available Database Links

Any user can query the data dictionary to determine what database links are available to that user. For information on viewing the data dictionary, see *Oracle7 Server Concepts* or the *Oracle7 Server Administrator's Guide.*

### Limiting the Number of Active Database Links

You can limit the number of connections from a user process to remote databases with the parameter OPEN_LINKS. This parameter controls the number of remote connections that a single user process can use concurrently within a single SQL statement. To improve application performance, increase the value of this parameter if users need to access more databases at the same time. This allows the user to access all the required remote data without waiting for the local instance to close and open connections.

## Database Links and Oracle Transparent Gateways

An Oracle Transparent Gateway provides access to non–Oracle data and services, such as cooperative server connectivity to IBM DB2 through a transparent gateway.

Once a gateway is installed, Oracle7 client applications can access the non–oracle data as if it were data in Oracle tables. To do so, a system administrator creates database links and local synonyms at each integrating server for the gateway server. Database links and synonyms provide location transparency. They are created on the Oracle7 Servers that integrate the transparent gateway into the Oracle7 cooperative server environment.

## Tools for Managing and Monitoring the Database

The database administrator for a distributed system must deal with added levels of complexity not faced by administrators of dedicated host systems.

Beyond the typical database administration duties, a database administrator for a distributed environment will need to deal with some administration duties specific to distributed systems. The administrator may need to coordinate with a number of other administrators including the network administrator to properly coordinate changes made to the system.

Oracle provides several utilities to aid the database administrator in maintaining and monitoring database performance.

## Third–Party Vendor Support

There are currently more than 60 companies producing more than 150 products that help manage Oracle databases and networks providing a truly open environment.

## SNMP Support

Besides its network administration capabilities, Oracle Simple Network Management Protocol (SNMP) support allows an Oracle server to be located and queried by any SNMP–based network management system. SNMP is the accepted standard underlying many popular network management systems such as:

- HP's OpenView
- Digital's POLYCENTER Manager on NetView
- IBM's NetView / 6000
- Novell's NetWare Management System
- SunSoft's SunNet Manager

These graphical systems display various views of a network, allowing administrators to zoom in to provide more detail about an individual service or device. Oracle SNMP Support allows DBAs to:

- monitor current status of Oracle servers
- receive alerts when exceptional events occur
- integrate database, network, and system management procedures more closely
- spot and react instantly to potential problems

DBAs administering multiple databases no longer must repeat basic tasks for every database instance. For example, logging sequentially into multiple machines to check the status of each Oracle database. After DBAs configure SNMP support in Network Manager, they can use it to monitor current activity for all Oracle databases on a network and request more detail when desired. DBAs can use SNMP support to verify normal activity and spot abnormal situations faster, allowing more time for other, less automatic tasks.

SNMP support aligns database managements tasks with those of system or network managers. For example, DBAs can be alerted if a database file runs out of space in the middle of the night.

DBAs can monitor a number of variables about Oracle servers; every variable is defined in a Management Information Base (MIB). By monitoring key variables, such as the current number of transactions and the amount of space allocated and used, DBAs can spot potential problems more readily.

Most systems support the ability to call another program, such as Oracle Server Manager, to allow the DBA to respond to an event, such as an abnormal shutdown or out–of–control query.

Currently, MIBs are available for:

- Oracle Network Listener
- Oracle MultiProtocol Interchange
- Oracle Names
- Oracle Media Server

**Note:** Oracle SNMP Support is not intended to replace Oracle Server Manager which offers a different set of functions for managing and controlling individual Oracle servers and applications.

For information on configuring Oracle SNMP Support for Oracle networking products (Oracle Server and Listener, Oracle MultiProtocol Interchange, and Oracle Names), see the *Oracle Network Manager Administrator's Guide.* For information on using the Oracle SNMP Support feature to develop third–party SNMP–based management applications, see the *Oracle SNMP Support Reference Guide.*

## Server Manager

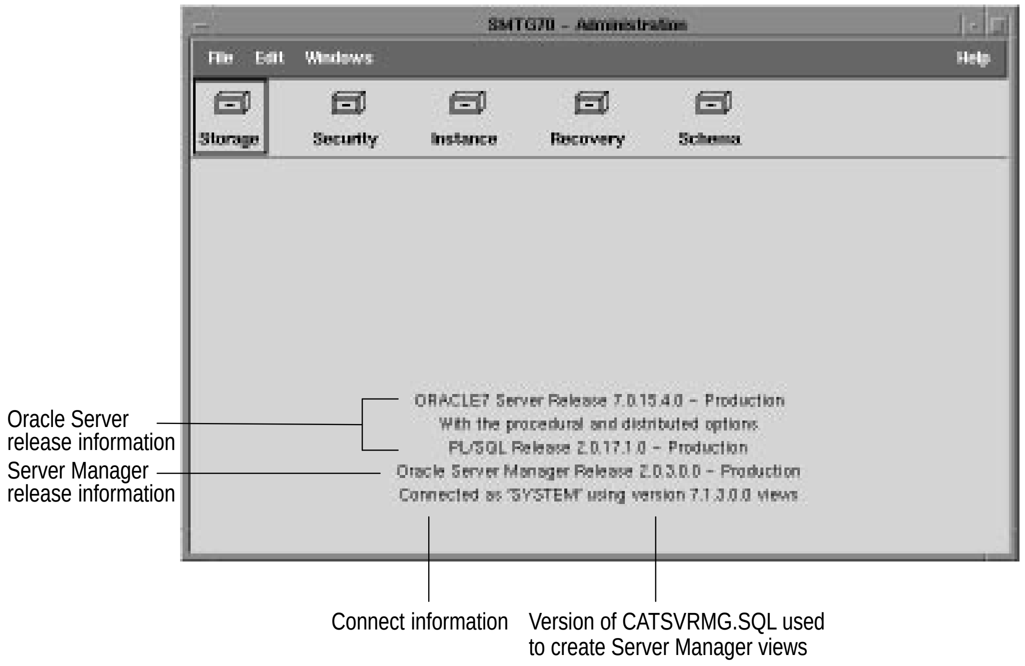
Server Manager is Oracle's database administration tool. The graphical component of Server Manager (Server Manager / GUI) allows you to perform database administration tasks with the convenience of a graphical user interface (GUI). The line mode component of Server Manager provides a line–mode interface.

☞ **Attention:** Server Manager replaces SQL*DBA after release 7.2.

Server Manager provides administrative functionality via an easy–to–use interface. You can use Server Manager to:

- Perform traditional administrative tasks, such as database startup, shutdown, backup, and recovery. Rather than manually entering the SQL commands to perform these tasks, you can use Server Manager's graphical interface to execute the commands quickly and conveniently by pointing and clicking with the mouse.
- Concurrently perform multiple tasks. Because you can open multiple windows simultaneously in Server Manager, you can perform multiple administrative and non–administrative tasks concurrently.

- Administer multiple databases. You can use Server Manager to administer a single database or to simultaneously administer multiple databases.
- Centralize database administration tasks. You can administer both local and remote databases running on any Oracle platform in any location worldwide. In addition, these Oracle platforms can be connected by any network protocol(s) supported by SQL*Net and the MultiProtocol Interchange.
- Dynamically execute SQL, PL/SQL, and Server Manager commands. You can use Server Manager to enter, edit, and execute statements. Server Manager also maintains a history of statements executed. Thus, you can re–execute statements without retyping them, a particularly useful feature if you need to execute lengthy statements repeatedly.
- Perform administrative tasks using Server Manager's line–mode interface when a graphical user interface is unavailable or undesirable.


SMTG70 – Administration
File
Edit
Windows
Help
Storage
Security
Instance
Recovery
Schema
Oracle Server release information
Server Manager release information
Connect information
Version of CATSVRMG.SQL used to create Server Manager views

```
ORACLE7 Server Release 7.0.15.4.0 – Production
With the procedural and distributed options
PL/SQL Release 2.0.17.1.0 – Production
Oracle Server Manager Release 2.0.3.0.0 – Production
Connected as 'SYSTEM' using version 7.1.3.0.0 views
```

**Figure 4 – 6  Administration Window Version Banner**

Portability

Server Manager / GUI is available for multiple GUI environments, yet adopts the native look and feel of the platform on which it is running. So Server Manager running on Motif looks like a Motif application, and Server Manager running on Windows looks like a Windows application.

Supported Oracle Server Releases

You can use Server Manager to administer any database running Oracle7 release 7.0 or later. You can also simultaneously administer different databases running different releases of Oracle7.

Server Manager in Line Mode

For those environments that do not support a graphical user interface, or for those times when a command line interface is desirable, Server Manager in line mode provides a conversational line mode. In line mode, you can explicitly execute commands on a command line.

You may want to use Server Manager in line mode when a graphical device is unavailable (such as when dialing–in from a non–GUI terminal), or when performing unattended operations (such as when running nightly batch jobs or batch scripts that do not require user intervention).

## Connecting Between Oracle Server Versions

In administering a distributed processing network, you must be aware of the version of Oracle software running at networked sites:

- Applications using database links originating from Oracle version 6 databases can query Oracle version 6 databases, as well as Oracle7 Server databases. When querying an Oracle7 Server database, Oracle version 6 can fetch both fixed– and variable–length strings. However, for fixed–length strings, Oracle version 6 uses Oracle comparison semantics, but portions of the query evaluated in the Oracle7 database use blank–padded comparison semantics.

  **Note:** SQL*Net version 2 cannot be used to connect to Oracle version 6 servers, only Oracle7 Servers. For information on issues to consider when migrating from an earlier version of SQL*Net, and coexistence of SQL*Net version 1 and 2, see *Understanding SQL*Net*.

- Applications using database links originating from Oracle7 Server databases can query any number of Oracle version 6 databases per statement. Applications using database links originating from Oracle7 Server databases can update a single Oracle version 6 database per statement, but no other nodes can be updated in the same single–site transaction.

  In both queries and updates, features unique to Oracle7 Server databases only work with Oracle version 6 data after it is retrieved.

  For example, if the database link sends the entire SQL statement to the Oracle version 6 database, the oracle version 6 database is incapable of reopening the same database link to return the results.

  If a distributed transaction updates data at only a single node, the transaction is committed without a two–phase commit because a coordinated commit is not required. An Oracle7 Server database can update (committing using two–phase commit) any number of Oracle7 Server databases within the same transaction.

  Attempting to update an Oracle version 6 database and either an Oracle7 database or a different Oracle version 6 database results in an error message ("ORA–02047: cannot join distributed transaction in progress").

CHAPTER

5

# Distributed Updates

This chapter describes how Oracle7 maintains the integrity of distributed update transactions. Topics include:

- updates in a distributed environment
- replication
- concurrency control
- transaction recovery management
- the prepare / commit phases
- controlling coordination of the prepare / commit phases
- failures that can affect prepare / commit
- how Oracle7 transparently recovers in–doubt distributed transactions

## Updates in a Distributed Environment

Distributed updates add several levels of complexity to a distributed system. The fact that multiple users are sharing and accessing data that exists at many sites rather that at a single site, adds the following considerations when these users attempt to update that data:

- replication (update propagation)
- concurrency control
- transaction recovery management

### Replication

In a distributed system, a data object can be represented at many sites. Making sure that updates to any representative at any of the sites are propagated to all other sites is the responsibility of Oracle7's replication mechanisms.

The Oracle Server provides several methods for data replication.

- read–only snapshots
- symmetric replication facility
  - updatable snapshots
  - N–way master replication

A *snapshot* is a full copy of a table, or a subset of a table, that reflects a recent state of a *master table* (the table on the node that you designate as the master node). A snapshot is defined by a distributed query that references one or more master tables, views, or, with certain limitations, other snapshots.

Read–only snapshots can be used for queries only and are the simplest form of replication. They are typically used for systems in which many sites need to query data that are updated only by one site..

Updatable snapshots can reflect local updates and therefore improve response time by avoiding network traffic. However, there must be a mechanism that ensures that the local updates are not lost when the snapshot is refreshed from the master table.

Oracle's symmetric replication facility provides that mechanism. It allows multiple copies of data to be maintained at different sites in a distributed system.

See *Oracle7 Server Distributed Systems, Volume II* for information about the concepts behind replication and how to implement replication in your distributed system.

## Concurrency Control

In a distributed system, there is the strong potential for more than one user to be concurrently executing transactions that update the same data and produce incorrect results. Oracle provides locking mechanisms to handle multiuser access to the same data.

See Chapter 10, "Data Concurrency", of the *Oracle7 Server Concepts* manual for more information.

In a distributed system, there is another level of complexity. A situation called global deadlock can occur. For example, transaction T1 has a lock on object A in San Francisco and requests a lock on object B in Dallas, and transaction T2 has a lock on object B in Dallas and requests a lock on object A in San Francisco. This series of events would cause both transactions to go into wait states.

In Oracle distributed transactions, local deadlocks are detected by analyzing a "waits for" graph, and global deadlocks are detected by a time–out. Once detected, non–distributed and distributed deadlocks are handled by the database and application in the same way.

## Distributed Transaction Management

All participants (nodes) in a distributed transaction should be unanimous as to the action to take on that transaction. That is, they should either all commit or rollback.

Oracle uses a prepare / commit mechanism to ensure that all participants in the distributed transaction are "on the same track".

Briefly, a prepare / commit mechanism works this way (more detailed instructions for implementing prepare / commit are in following sections):

- **Phase 1**

  In a prepare / commit system, the initiating node is called the *global coordinator.* When this site receives a commit request from an update transaction, it asks all participants in the distributed system to prepare (to promise to commit or rollback the transaction, even if there is a failure). The participating sites will reply that they are either prepared to commit or not.

- **Phase 2**

  If the global coordinator receives a "prepared to commit" message from all the participating sites, it broadcasts a commit command to all sites. If even *one* site has replied that it is *not* prepared to commit, the global coordinator will abort the transaction and send out a rollback (or undo) command.

## The Distributed Transaction Management Mechanism

Oracle7 *automatically* controls and monitors the commit or rollback of a distributed update transaction and maintains the integrity of the *global database* (the collection of databases participating in the transaction) using a transaction recovery management mechanism known as prepare/commit). This mechanism is completely transparent. Its use requires no programming on the part of the user or application developer.

The information in this section explains how this mechanism works, why it is used, and how you can take advantage of its features to better design and configure a distributed system.

### Coordinating Distributed Updates

By definition, changes made by all SQL statements in a transaction are either committed or rolled back as a unit. The commit of a non–distributed transaction (one that contains SQL statements that modify data only at a local database) is simple — all changes are either committed or rolled back as a unit in the non–distributed system.

However, the commit or rollback of a distributed transaction must be coordinated over a network so that the participating nodes either all commit or all roll back the transaction, even if a network failure or a system failure of any number of nodes occur during the process. The prepare/commit mechanism guarantees that the nodes participating in a distributed transaction either all commit or all roll back the transaction, thus maintaining the integrity of the global database.

### When Is Transaction Management Needed?

The transaction management mechanism is used only when a distributed update changes the contents of two or more databases in the distributed system, or there is a remote procedure call (RPC) that references a remote object using its global object name. When a node is read–only, Oracle7 automatically notes this, and the node need not participate in the ensuing prepare/commit phases.

All implicit database changes performed by integrity constraints, remote procedure calls, and triggers are also protected by Oracle7's distributed transaction management mechanism.

## The Prepare and Commit Phases

The processing of a distributed update has two distinct phases:

| | |
|---|---|
| prepare phase | The *global coordinator* (initiating node) asks participants to *prepare* (to promise to commit or rollback the transaction, even if there is a failure). |
| commit phase | If all participants respond to the coordinator that they are prepared, the coordinator asks all nodes to *commit* the transaction. If any participants cannot prepare, the coordinator asks all nodes to roll back the transaction. |

When a user commits a distributed transaction with a COMMIT statement, both phases are performed automatically. The following sections describe each phase in further detail.

### Prepare Phase

The first phase in committing a distributed transaction is the prepare phase in which the commit of the transaction is not actually carried out. Instead, all nodes referenced in a distributed transaction (except one, known as the commit point site, described on page 5 – 10) are told to prepare (to commit). By preparing, a node records enough information so that it can subsequently either commit or abort the transaction (in which case, a rollback will be performed), regardless of intervening failures.

When a node responds to its requestor that it has prepared, the prepared node has made a promise to be able to either commit or roll back the transaction later and not to make a unilateral decision on whether to commit or roll back the transaction.

**Note:** Queries that start after a node has prepared cannot access the associated locked data until all phases are complete (an insignificant amount of time unless a failure occurs).

When a node is told to prepare, it can respond with one of three responses:

| | |
|---|---|
| prepared | Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared. |
| read–only | No data on the node has been, or can be, modified (only queried), so no prepare is necessary. |
| abort | The node cannot successfully prepare. |

## Prepare Phase Actions by Nodes

To complete the prepare phase, each node performs the following actions:

- The node requests its descendants (nodes subsequently referenced) to prepare.
- The node checks to see if the transaction changes data on that node or any of its descendants. If there is no update, the node skips the next steps and replies with a read–only message (see below).
- The node allocates all resources it needs to commit the transaction if data is updated.
- The node flushes any entries corresponding to changes made by that transaction to its local redo log.
- The node guarantees that locks held for that transaction are able to survive a failure.
- The node responds to the node that referenced it in the distributed transaction with a prepared message *or*, if its prepare or the prepare of one of its descendents was unsuccessful, with an abort message (see below).

These actions guarantee that the transaction can subsequently commit or roll back on that node. The prepared nodes then wait until a COMMIT or ROLLBACK is sent. Once the node(s) are prepared, the transaction is said to be *in–doubt*.

## Read–only Response

When a node is asked to prepare and the SQL statements affecting the database do not change that node's data, the node responds to the node that referenced it with a read–only message. These nodes do not participate in the second phase (the commit phase). For more information about read–only distributed transactions, see "Read–Only Distributed Transactions" on page 5 – 20

## Unsuccessful Prepare

When a node cannot successfully prepare, it performs the following actions:

- That node releases any resources currently held by the transaction and rolls back the local portion of the transaction.
- The node responds to the node that referenced it in the distributed transaction with an *abort message*.

These actions then propagate to the other nodes involved in the distributed transaction to roll back the transaction and guarantee the integrity of the data in the global database.

Again, this enforces the primary rule of a distributed transaction. All nodes involved in the transaction either all commit or all roll back the transaction at the same logical time.

### Commit Phase

The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, *all* nodes referenced in the distributed transaction have guaranteed that they have the necessary resources to commit the transaction. That is, they are all prepared.

Therefore, the commit phase consists of the following steps:

1. The global coordinator send a message to all nodes telling them to commit the transaction.
2. At each node, Oracle7 commits the local portion of the distributed transaction (releasing locks) and records an additional redo entry in the local redo log, indicating that the transaction has committed.

When the commit phase is complete, the data on all nodes of the distributed system are consistent with one another.

A variety of failure cases, caused by network or system failures, are possible during both the prepare phase and the commit phase. For a description of failure situations and how Oracle7 resolves intervening failures during prepare / commit, see "Troubleshooting Distributed Update Problems" on page 5 – 22.

## The Session Tree

As the statements in a distributed transaction are issued, Oracle7 defines a *session tree* of all nodes participating in the transaction. A session tree is a hierarchical model that describes the relationships between sessions and their roles. All nodes participating in the session tree of a distributed transaction assume one or more roles:

- a client
- a database server
- a global coordinator
- a local coordinator
- the commit point site (see page 5 – 10)

The role a node plays in a distributed transaction is determined by:

- whether the transaction is local or remote
- the *commit point strength* of that node (see page 5 – 11)

- whether all requested data is available at a node, or whether other nodes need to be referenced to complete the transaction
- whether the node is read–only

Figure 5 – 1 below illustrates a simple session tree.

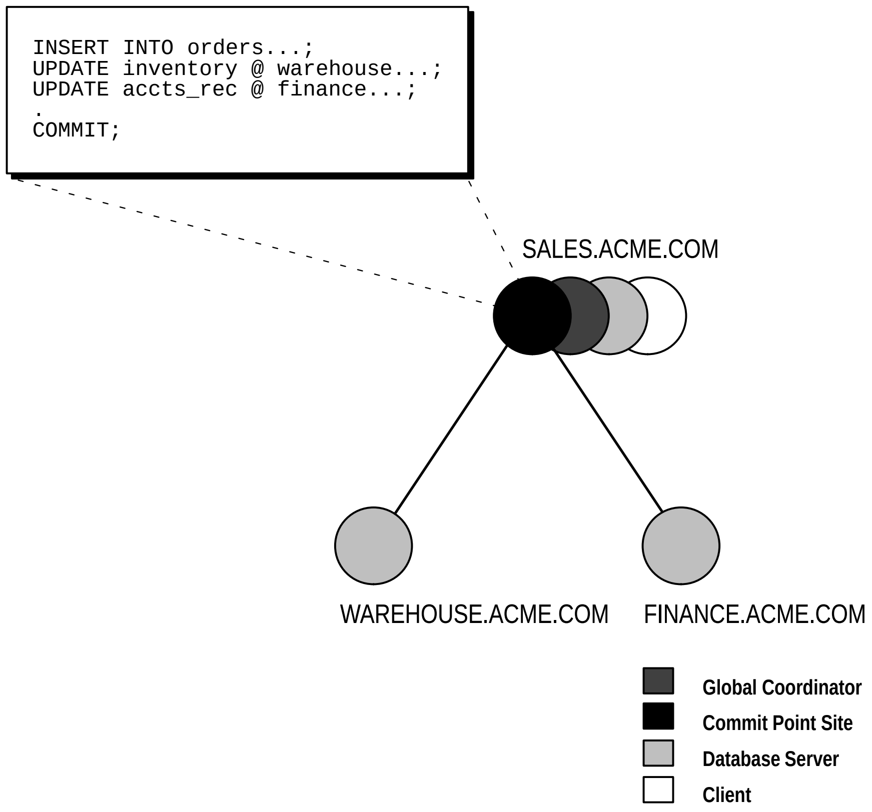


**Figure 5 – 1 An example of a Simple Session Tree**

## Clients

A node acts as a client when it references information from another node's database. The referenced node is a *database server*. In the above example, the node SALES.ACME.COM is a client of the nodes (database servers) that serve the WAREHOUSE and FINANCE databases.

## Servers and Database Servers

A server is a node that is directly referenced in a distributed transaction or is requested to participate in a transaction because another node requires data from its database. A node supporting a database is also called a database server.

In Figure 5 – 1, an application at the node holding the SALES database initiates a distributed transaction which accesses data from the nodes holing the WAREHOUSE and FINANCE databases. Therefore, SALES.ACME.COM has the role of client node, and WAREHOUSE and FINANCE are both database servers.

In this example, SALES is a database server *and* a client because the application is also requesting an update of the SALES database.

## Local Coordinators

A node that must reference data on other nodes to complete its part in the distributed transaction is called a *local coordinator*. In Figure 5 – 1, SALES.ACME.COM, although it happens to be the global coordinator, is also considered a local coordinator because it coordinates the nodes it directly references: WAREHOUSE.ACME.COM and FINANCE.ACME.COM.

A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:

- receiving and relaying transaction status information to and from those nodes
- passing queries to those node
- receiving queries from those nodes and passing them on to other nodes
- returning the results of queries to the nodes that initiated them

## The Global Coordinator

The node where the distributed transaction originates (to which the database application issuing the distributed transaction is directly connected) is called the global coordinator. This node becomes the parent or root of the session tree. The global coordinator performs the following operations during a distributed transaction:

- All of the distributed transaction's SQL statements, remote procedure calls, etc. are sent by the global coordinator to the directly referenced nodes, thus forming the session tree.

  For example, in Figure 5 – 1, the transaction issued at the node SALES.ACME.COM references information from the database servers WAREHOUSE.ACME.COM and FINANCE.ACME.COM. Therefore, SALES.ACME.COM is the global coordinator of this distributed transaction.

- The global coordinator instructs all directly referenced nodes other than the commit point site (see below) to prepare the transaction.
- If all nodes prepare successfully, the global coordinator instructs the commit point site to initiate the global commit of the transaction.
- If there is one or more abort messages, the global coordinator instructs all nodes to initiate a global rollback of the transaction.

For more information about the global coordinator's role, see "A Simple Example" on page 5 – 10.

## The Commit Point Site

The job of the commit point site is to initiate a commit or roll back as instructed by the global coordinator. The system administrator always designates one node to be the *commit point site* in the session tree by assigning all nodes a commit point strength (see below). The node selected as commit point site should be that node that stores the most critical data (the data most widely used)

The commit point site is distinct from all other nodes involved in a distributed transaction with respect to the following two issues:

- The commit point site never enters the prepared state. This is potentially advantageous because if the commit point site stores the most critical data, this data never remains in–doubt, even if a failure situation occurs. (In failure situations, failed nodes remain in a prepared state, holding necessary locks on data until in–doubt transactions are resolved.)
- In effect, the outcome of a distributed transaction at the commit point site determines whether the transaction at all nodes is committed or rolled back. The global coordinator ensures that all nodes complete the transaction the same way that the commit point site does.

A distributed transaction is considered to be committed once all nodes are prepared and the transaction has been committed at the commit point site (even though some participating nodes may still be only in the prepared state and the transaction not yet actually committed).

The commit point site's redo log is updated as soon as the distributed transaction is committed at that node. Likewise, a distributed transaction is considered *not* committed if it has not been committed at the commit point site.

## Commit Point Strength

Every node acting as a database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines what role it plays in the prepare / commit phases. Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction.

This value is specified using the initialization parameter COMMIT_POINT_STRENGTH (see page 5 – 13).

The commit point site is determined at the beginning of the prepare phase. The commit point site is selected only from the nodes participating in the transaction. Once it has been determined, the global coordinator sends prepare messages to all participating nodes.

Of the nodes directly referenced by the global coordinator, the node with the highest commit point strength is selected. Then, the initially–selected node determines if any of its servers (other nodes that it has to obtain information from for this transaction) has a higher commit point strength.

Either the node with the highest commit point strength directly referenced in the transaction, or one of its servers with a higher commit point strength becomes the commit point site. Figure 5 – 2 shows in a sample session tree the commit point strengths of each node (in parentheses) and shows the node chosen as the commit point site.

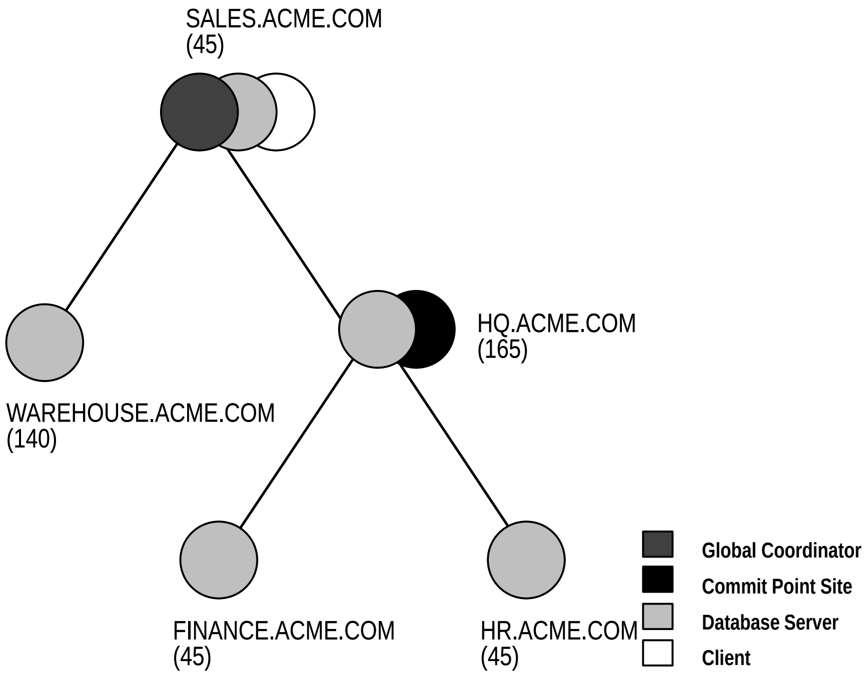


**Figure 5 – 2 Commit Point Strengths and Determination of the Commit Point Site**

The following conditions apply when determining the commit point site:

- A read–only node (a node which will not change its local data for the transaction) cannot be designated as the commit point site.
- If multiple nodes directly referenced by the global coordinator have the same commit point strength, Oracle7 will designate one of these nodes as the commit point site.
- If a distributed transaction ends with a rollback, the prepare and commit phases are not needed, consequently a commit point site is never determined. Instead, the global coordinator sends a ROLLBACK statement to all nodes and ends the processing of the distributed transaction.

The commit point strength only determines the commit point site in a distributed transaction. Because the commit point site stores information about the status of the transaction, the commit point site should not be a node that is frequently unreliable or unavailable in case other nodes need information about the transaction's status.

As Figure 5 – 2 illustrates, the commit point site and the global coordinator can be different nodes of the session tree.

The commit point strengths of each nodes is communicated to the coordinator(s) when the initial connections are made. The coordinator(s) retain the commit point strengths of each node they are in direct communication with so that commit point sites can be efficiently selected during prepare / commits. Therefore, it is not necessary for the commit point strength to be exchanged between a coordinator and a node each time a commit occurs.

## Specifying the Commit Point Strength of an Instance

Specify a commit point strength for each node that insures that the most critical server will be "non–blocking" if a failure occurs during a prepare / commit phase.

A node's commit point strength should relate to the estimated number of collisions that can result from data locked by in–doubt transactions. For example, mainframe–based database servers will probably have higher commit point strengths than minicomputer–based servers. In turn, minicomputer–based database servers will probably have higher commit point strengths than PC–based database servers. To determine each node's commit point strength, it will be necessary for all administrators of the distributed system to communicate and establish the appropriate values.

A node's commit point strength is set by the initialization parameter COMMIT_POINT_STRENGTH. The range of values is any integer from 0 to 255. For example, to set the commit point strength of a database to 200, include the following line in that database's parameter file:

```
COMMIT_POINT_STRENGTH=200
```

OSDoc

**Additional Information:** See your Oracle operating system–specific documentation for the default value.

## A Case Study

This case study illustrates:

- the definition of a session tree
- how a commit point site is determined
- when prepare messages are sent
- when a transaction actually commits
- what information is stored locally about the transaction

### The Scenario

A company that has separate Oracle7 servers, SALES.ACME.COM and WAREHOUSE.ACME.COM. As sales records are inserted into the SALES database, associated records are being updated at the WAREHOUSE database.

### The Process

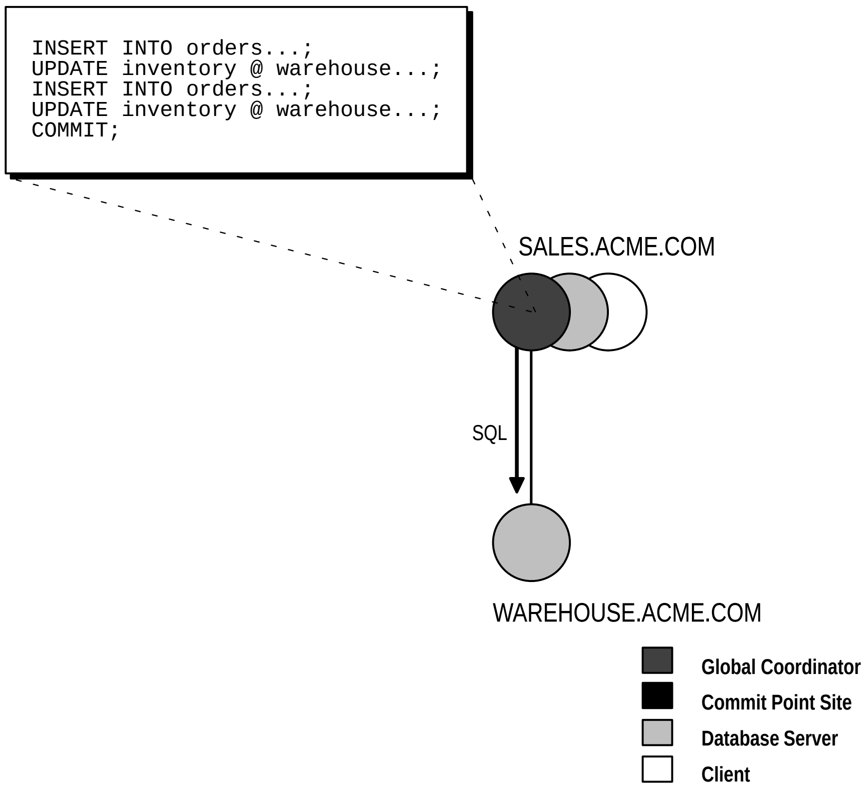
The following steps are carried out during a distributed update transaction that enters a sales order:

1. An application issues SQL statements.

   At the Sales department, a salesperson uses a database application to enter, then commit a sales order. The application issues a number of SQL statements to enter the order into the SALES database and update the inventory in the WAREHOUSE database.

   These SQL statements are all part of a single distributed transaction, guaranteeing that all issued SQL statements succeed or fail as a unit. This prevents the possibility of an order being placed but, inventory is not updated to reflect the order. In effect, the transaction guarantees the consistency of data in the global database.

   As each of the SQL statements in the transaction executes, the session tree is defined, as shown in Figure 5 – 3.



```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
COMMIT;
```

**Figure 5 – 3 Defining the Session Tree**

Note the following:

- An order entry application running with the SALES database initiates the transaction. Therefore, SALES.ACME.COM is the global coordinator for the distributed transaction.
- The order entry application inserts a new sales record into the SALES database and updates the inventory at the warehouse. Therefore, the nodes SALES.ACME.COM and WAREHOUSE.ACME.COM are both database servers. Furthermore, because SALES.ACME.COM updates the inventory, it is a client of WAREHOUSE.ACME.COM.

This completes the definition of the session tree for this distributed transaction.

Remember that each node in the tree has acquired the necessary data locks to execute the SQL statements that reference local data. These locks remain even after the SQL statements have been executed until the prepare / commit phases are completed.

2. The application issues a COMMIT statement.

   The final statement in the transaction that enters the sales order is now issued — a COMMIT statement which begins the prepare/commit phases starting with the prepare phase.

3. The global coordinator determines the commit point site.

   The commit point site is determined immediately following the COMMIT statement. SALES.ACME.COM, the global coordinator, is determined to be the commit point site, as shown in Figure 5 – 4.

   See "Specifying the Commit Point Strength" on page 5 – 13 for more information about how the commit point site is determined.

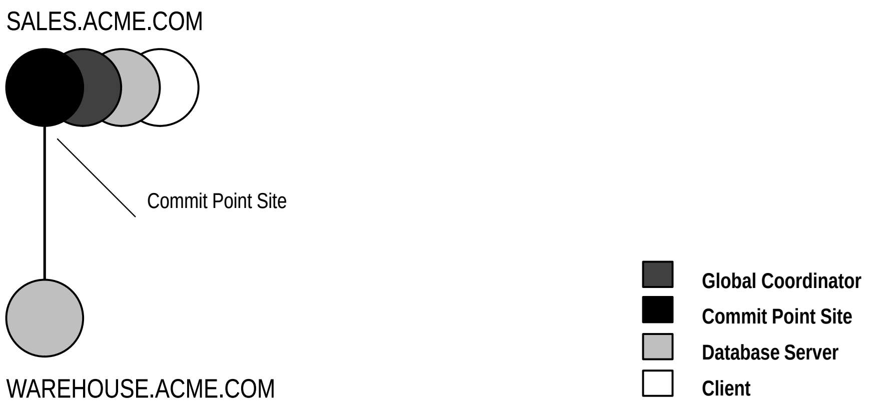


**Figure 5 – 4 Determining the Commit Point Site**

4. The global coordinator sends the Prepare message.

   After the commit point site is determined, the global coordinator sends the prepare message to all directly referenced nodes of the session tree, *excluding* the commit point site. In this example, WAREHOUSE.ACME.COM is the only node asked to prepare.

   WAREHOUSE.ACME.COM tries to prepare. If a node can guarantee that it can commit the locally dependent part of the transaction and can record the commit information in its local redo log, the node can successfully prepare.

In this example, only WAREHOUSE.ACME.COM receives a prepare message because SALES.ACME.COM is the commit point site (which does not prepare). WAREHOUSE.ACME.COM responds to SALES.ACME.COM with a prepared message.

As each node prepares, it sends a message back to the node that asked it to prepare. Depending on the responses, two things can happen:

- If *any* of the nodes asked to prepare respond with an abort message to the global coordinator, the global coordinator then tells all nodes to roll back the transaction, and the process is completed.
- If *all* nodes asked to prepare respond with a prepared or a read–only message to the global coordinator. That is, they have successfully prepared, the global coordinator asks the commit point site to commit the transaction.

Continuing with the example, Figure 5 – 5 illustrates the parts of Step 4.

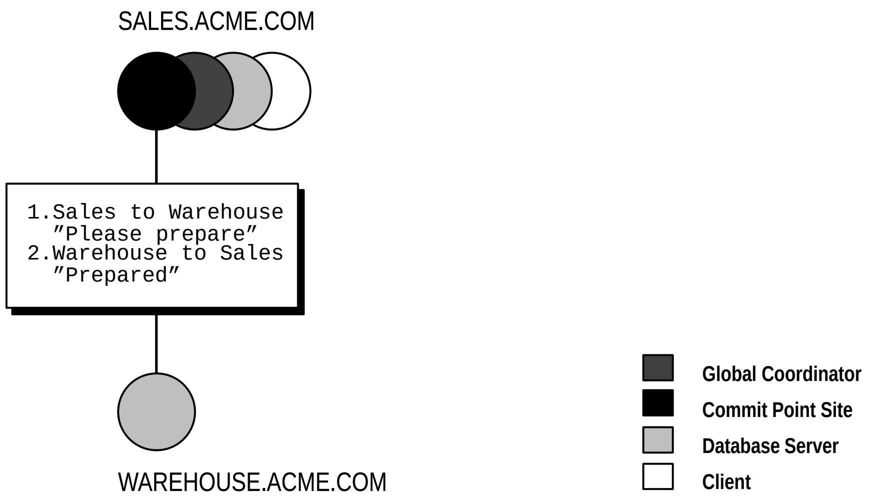


**Figure 5 – 5 Sending and Acknowledging the PREPARE Message**

5. The commit point site commits.

   SALES.ACME.COM, receiving acknowledgement that WAREHOUSE.ACME.COM is prepared, instructs the commit point site (itself, in this example) to commit the transaction. The commit point site now commits the transaction locally and records this fact in its local redo log.

Even if WAREHOUSE.ACME.COM has not committed yet, the outcome of this transaction is determined, that is, the transaction *will* be committed at all nodes even if the node's ability to commit is delayed.

6. The commit point site informs the global coordinator of the commit.

   The commit point site now tells the global coordinator that the transaction has committed. In this case, where the commit point site and global coordinator are the same node, no operation is required. The commit point site remembers it has committed the transaction until the global coordinator confirms that the transaction has been committed on all other nodes involved in the distributed transaction.

   After the global coordinator has been informed of the commit at the commit point site, it tells all other directly referenced nodes to commit. In turn, any local coordinators instruct their servers to commit, and so on. Each node, including the global coordinator, commits the transaction and records appropriate redo log entries locally. As each node commits, the resource locks that were being held locally for that transaction are released.

   Figure 5 – 6 illustrates Step 6 in this example. SALES.ACME.COM, both the commit point site and the global coordinator, has already committed the transaction locally. SALES now instructs WAREHOUSE.ACME.COM to commit the transaction.

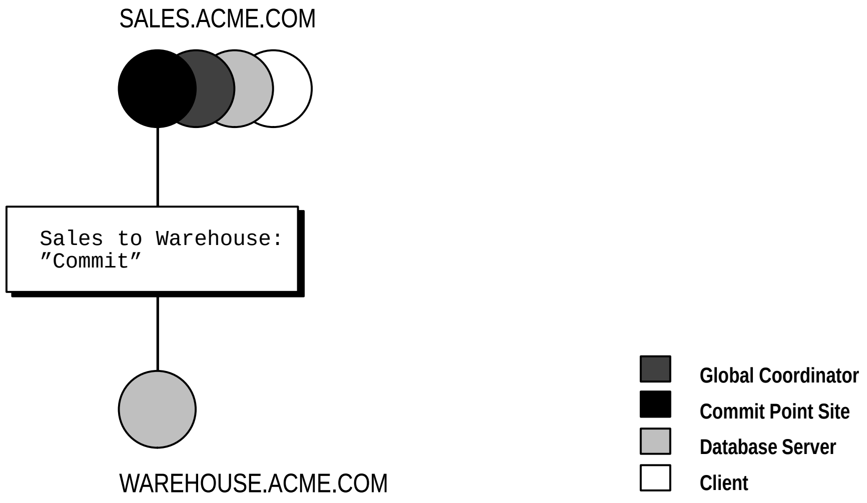


**Figure 5 – 6 The Global Coordinator and Other Servers Commit the Transaction**

7. The global coordinator and commit point site complete the commit.

   After all referenced nodes and the global coordinator have committed the transaction, the global coordinator informs the commit point site.

   The commit point site, which has been waiting for this message, erases the status information about this distributed transaction and informs the global coordinator that it is finished. In other words, the commit point site forgets about committing the distributed transaction. This is acceptable because all nodes involved in the prepare / commit phases have committed the transaction successfully, and they will never have to determine its status in the future.

   After the commit point site informs the global coordinator that it has forgotten about the transaction, the global coordinator finalizes the transaction by forgetting about the transaction itself.

   This completes the COMMIT phase and thus completes the distributed update transaction.

All of the steps described above are accomplished automatically and in a fraction of a second.

## Coordination of System Change Numbers

Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction. In a distributed system, the SCNs of communicating nodes are coordinated when:

- A connection occurs using the path described by one or more database links.
- A distributed SQL statement executes (the execute phase completes).
- A distributed transaction commits.

Among other benefits, the coordination of SCNs among the nodes of a distributed system allows global distributed read–consistency at both the statement and transaction level. If necessary, global distributed time–based recovery can also be completed.

During the prepare phase, Oracle7 determines the highest SCN at all nodes involved in the transaction. The transaction then commits with the high SCN at the commit point site. The commit SCN is then sent to all prepared nodes with the commit decision.

## Read–Only Distributed Transactions

There are three cases in which all or part of a distributed transaction is read–only:

- A distributed transaction can be partially read–only if:
  - only queries are issued at one or more nodes
  - updates do not modify any records
  - updates rolled back due to violations of integrity constraints or triggers being fired

  In each of these cases, the read–only nodes recognize this fact when they are asked to prepare. They respond to their respective local coordinators with a read–only message. By doing this, the commit phase completes faster because Oracle eliminates the read–only nodes from subsequent processing.

- The distributed transaction can be completely read–only (no data changed at any node) and the transaction is *not* started with the SET TRANSACTION READ ONLY statement.

  In this case, all nodes recognize that they are read–only during the prepare phase, and no commit phase is required. However, the global coordinator, not knowing whether all nodes are read–only, must still perform the operations involved in the prepare phase.
- The distributed transaction can be completely read–only (all queries at all nodes) and the transaction *is* started with a SET TRANSACTION READ ONLY statement. In this case, only queries are allowed in the transaction, and the global coordinator does not have to undertake a prepare/commit. Updates by other transactions do not degrade global transaction–level read consistency, because it is automatically guaranteed by coordination of SCNs at each node of the distributed system.

## Limiting the Number of Distributed Transactions Per Node

The initialization parameter DISTRIBUTED_TRANSACTIONS controls the number of possible distributed transactions in which a given instance can concurrently participate, both as a client and a server. If this limit is reached and a subsequent user tries to issue a SQL statement referencing a remote database, the statement is rolled back and the following error message is returned:

```
ORA–2042: too many global transactions
```

For example, assume that DISTRIBUTED_TRANSACTIONS is set to 10 for a given instance. In this case, a maximum of ten sessions can concurrently be processing a distributed transaction. If an eleventh session attempts to issue a DML statement requiring distributed access, an error message is returned to the session, and the statement is rolled back.

The database administrator should consider increasing the value of the initialization parameter DISTRIBUTED_TRANSACTIONS when an instance regularly participates in numerous distributed transactions and the above error message is frequently returned as a result of the current limit. Increasing the limit allows more users to concurrently issue distributed transactions.

If the DISTRIBUTED_TRANSACTIONS initialization parameter is set to zero, no distributed SQL statements can be issued in any session.

Also, the RECO background process is not started at startup of the local instance. In–doubt distributed transactions that may be present (from a previous network or system failure) cannot be automatically resolved by Oracle7.

Therefore, only set this initialization parameter to zero to prevent distributed transactions when a new instance is started, and when it is certain that no in–doubt distributed transactions remained after the last instance shut down.

**Additional Information:** See the *Oracle7 Server Reference* for more information.

## Troubleshooting Distributed Update Problems

A network or system failure can cause the following types of problems:

- A prepare/commit being processed when a failure occurs may not be completed at all nodes of the session tree.
- If a failure persists (for example, if the network is down for a long time), the data exclusively locked by in–doubt transactions is unavailable to statements of other transactions.

The following sections describe these situations.

### Failures that Interrupt Prepare/Commit

The user program that commits a distributed transaction is informed of a problem by one of the following error messages:

```
ORA–02050: transaction ID rolled back,
           some remote dbs may be in–doubt
ORA–02051: transaction ID committed,
           some remote dbs may be in–doubt
ORA–02054: transaction ID in–doubt
```

A robust application should save information about a transaction if it receives any of the above errors. This information can be used later if manual distributed transaction recovery is desired.

**Note:** The failure cases that prompt these error messages are beyond the scope of this book and are unnecessary to administer the system.

No action is required by the administrator of any node that has one or more in–doubt distributed transactions due to a network or system failure. The automatic recovery features of Oracle7 transparently complete any in–doubt transaction so that the same outcome occurs on all nodes of a session tree (that is, all commit or all roll back) once the network or system failure is resolved.

However, in extended outages, the administrator may wish to force the commit or rollback of a transaction to release any locked data. Applications must account for such possibilities.

## Failures that Prevent Access of Data

When a user issues a SQL statement, Oracle7 attempts to lock the required resources to successfully execute the statement. However, if the requested data is currently being held by statements of other uncommitted transactions and continues to remained locked for an excessive amount of time, a time–out occurs. Consider the following two scenarios.

### Transaction Time–Out

A DML SQL statement that requires locks on a remote database may be blocked from doing so if another transaction (distributed or non–distributed) currently own locks on the requested data. If these locks continue to block the requesting SQL statement, a time–out occurs, the statement is rolled back, and the following error message is returned to the user:

```
ORA–02049: time–out: distributed transaction waiting for lock
```

Because no data has been modified, no actions are necessary as a result of the time–out. Applications should proceed as if a deadlock has been encountered. The user who executed the statement can try to re–execute the statement later. If the lock persists, the user should contact an administrator to report the problem.

The timeout interval in the above situation can be controlled with the initialization parameter DISTRIBUTED_LOCK_TIMEOUT. This interval is in seconds. For example, to set the time–out interval for an instance to 30 seconds, include the following line in the associated parameter file:

```
DISTRIBUTED_LOCK_TIMEOUT=30
```

With the above time–out interval, the time–out errors discussed in the previous section occur if a transaction cannot proceed after 30 seconds of waiting for unavailable resources.

**Additional Information:** For more information about initialization parameters and editing parameter files, see the *Oracle7 Server Reference.*

### Lock From In–Doubt Transaction

A query or DML statement that requires locks on a local database may be blocked from doing so indefinitely due to the locked resources of an in–doubt distributed transaction. In this case, the following error message is immediately returned to the user:

```
ORA–01591: lock held by in-doubt distributed transaction ID
```

In this case, the SQL statement is rolled back immediately. The user who executed the statement can try to re–execute the statement later. If the lock persists, the user should contact an administrator to report the problem, *including* the ID of the in–doubt distributed transaction.

The chances of the above situations occurring are very rare, considering the low probability of failures during the critical portions of the prepare/commit phases. Even if such a failure occurs and assuming quick recovery from a network or system failure, problems are automatically resolved without manual intervention. Thus problems usually resolve before they can be detected by users or database administrators.

## Manually Overriding In–Doubt Transactions

A database administrator can manually force the COMMIT or ROLLBACK of a local in–doubt distributed transaction. However, a specific in–doubt transaction should be manually overridden *only* when the following situations exist:

- The in–doubt transaction locks data that is required by other transactions. This happens if users complain that the ORA–01591 error message interferes with their transactions.
- An in–doubt transaction prevents the extents of a rollback segment to be used by other transactions. The first portion of an in–doubt distributed transaction's local transaction ID corresponds to the ID of the rollback segment, as listed by the data dictionary views DBA_2PC_PENDING and DBA_ROLLBACK_SEGS.
- The failure that did not allow the prepare/commit phases to complete will not be corrected in an acceptable time period. Examples of such cases might include a telecommunication network that has been damaged or a damaged database that needs a substantial amount of time to complete recovery.

Normally, a decision to locally force an in–doubt distributed transaction should be made in consultation with administrators at other locations. A wrong decision can lead to database inconsistencies which can be difficult to trace and that you must manually correct.

If the conditions above do not apply, *always* allow the automatic recovery features of Oracle7 to complete the transaction. However, if any of the above criteria are met, the administrator should consider a local override of the in–doubt transaction. If a decision is made to locally force the transaction to complete, the database administrator should analyze available information with the following goals in mind:

- Try to find a node that has either committed or rolled back the transaction. If you can find a node that has already resolved the transaction, you can follow the action taken at that node.
- See if any information is given in the TRAN_COMMENT column of DBA_2PC_PENDING for the distributed transaction. Comments are included in the COMMENT parameter of the COMMIT command. For example, an in–doubt distributed transaction's comment might indicate the origin of the transaction and what type of transaction it is:

  ```
  COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
  ```

- See if any information is given in the ADVICE column of DBA_2PC_PENDING for the distributed transaction. An application can prescribe advice about whether to force the commit or force the rollback of separate parts of a distributed transaction with the ADVISE parameter of the SQL command ALTER SESSION.

  The advice sent during the prepare phase to each node is the advice in effect at the time the most recent DML statement executed at that database in the current transaction.

  For example, consider a distributed transaction that moves an employee record from the EMP table at one node to the EMP table at another node. The transaction could protect the record (even when administrators independently force the in–doubt transaction at each node) by including the following sequence of SQL statements:

  ```
  ALTER SESSION ADVISE COMMIT;
  INSERT INTO emp@hq ... ;    /*advice to commit at HQ */

  ALTER SESSION ADVISE ROLLBACK;
  DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/

  ALTER SESSION ADVISE NOTHING;
  ```

If you manually force the in–doubt transaction, the worst that can happen is that each node has a copy of the employee record being moved; the record cannot disappear.

## Manual Override Example

The following example shows a failure during the commit of a distributed transaction and how to go about gaining information before manually forcing the commit or rollback of the local portion of an in–doubt distributed transaction. Figure 5 – 7 illustrates the example.

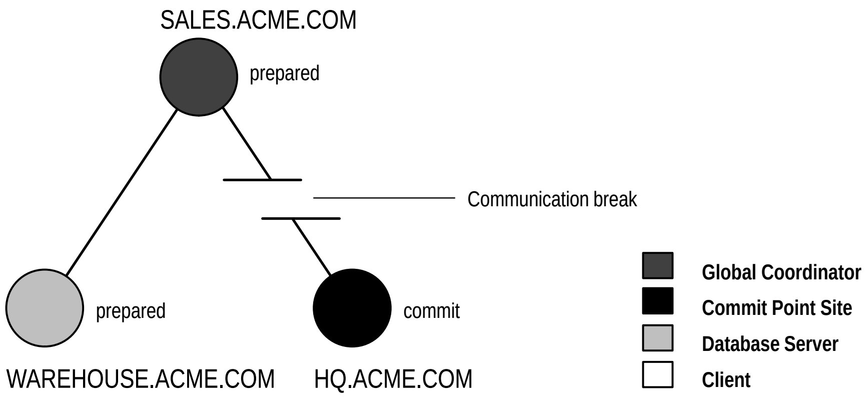


**Figure 5 – 7 An Example of an in–Doubt Distributed Transaction**

In this failure case, the prepare phase completed. However, during the commit phase, the commit point site's commit message (the message telling the global coordinator that the transaction was committed at the commit point site) never made it back to the global coordinator, even though the commit point site committed the transaction.

You are the WAREHOUSE database administrator. The inventory data locked because of the in–doubt transaction is critical to other transactions. However, the data cannot be accessed because the locks must be held until the in–doubt transaction either commits or rolls back. Furthermore, you understand that the communication link between sales and headquarters cannot be resolved immediately. Therefore, you decide to manually force the local portion of the in–doubt transaction using the following steps:

1. Record user feedback.
2. Query the local DBA_2PC_PENDING view to obtain the global transaction ID and get other information about the in–doubt transaction.
3. Query the local DBA_2PC_NEIGHBORS view to begin tracing the session tree so that you can find a node that resolved the in–doubt transaction.

4. Check the mixed outcome flag after normal communication is re–established.

The following sections explain each step in detail for this example.

## Step 1: Record User Feedback

The users of the local database system that conflict with the locks of the in–doubt transaction get the following error message:

```
ORA–01591: lock held by in–doubt distributed transaction 1.21.17
```

Here, 1.21.17 is the local transaction ID of the in–doubt distributed transaction in this example. The local database administrator should request and record this ID number from the users that report problems to identify in–doubt transactions that should be forced.

## Step 2: Query DBA_2PC_PENDING

Query the local DBA_2PC_PENDING (see also page 5 – 32) to gain information about the in–doubt transaction:

```
SELECT * FROM sys.dba_2pc_pending
   WHERE local_tran_id = ’1.21.17’;
```

For example, when the previous query is issued at WAREHOUSE, the following information is returned.

```
Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         SALES.ACME.COM.55d1c563.1.93.29
STATE                  prepared
MIXED                  no
ADVICE
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31–MAY–91
FORCE_TIME
RETRY_TIME             31–MAY–91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   system1
DB_USER                SWILLIAMS
COMMIT#
```

**Figure 5 – 8 Results of Querying DBA_2PC_PENDING**

The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction. It is of the form:

```
global_database_name.hhhhhhhh.local_transaction_id
```

Here, *global_database_name* is the database name of the global coordinator (where the transaction originates), *hhhhhhhh* is an internal database ID at the global coordinator (8 hexadecimal digits), and *local_tran_id* is the corresponding local transaction ID assigned on the global coordinator. Therefore, the last portion of the global transaction ID and the local transaction ID match at the global coordinator. In the example, you can tell that WAREHOUSE is not the global coordinator because these numbers do not match.

The transaction on this node is in a prepared state. Therefore, WAREHOUSE awaits its coordinator to send either a commit or a rollback message.

The transaction's comment or advice may include information about this transaction. If so, use this comment to your advantage. In this example, the origin (the sales order entry application) and transaction type is in the transaction's comment. This information may reveal something that would help you decide whether to commit or rollback the local portion of the transaction.

If useful comments do not accompany an in–doubt transaction, you must complete some extra administrative work to trace the session tree and find a node that has resolved the transaction.

### Step 3: Query DBA_2PC_NEIGHBORS

The purpose of this step is to climb the session tree so that you find coordinators, eventually reaching the global coordinator. Along the way, you might find a coordinator that has resolved the transaction. If not, you can eventually work your way to the commit point site, which will always have resolved the in–doubt transaction.

The DBA_2PC_NEIGHBORS view provides information about connections associated with an in–doubt transaction. Information for each connection is different, based on whether the connection is inbound or outbound:

- If the connection is inbound, your node is subordinate (a server of) another node. In this case, the DATABASE column lists the name of the client database that connected to your node, and the DBUSER_OWNER column lists the local account for the database link connection that corresponds to the in–doubt transaction.
- If the connection is outbound, your node is a client of other servers. In this case, the DATABASE column lists the name of the database link that connects to the remote node. The DBUSER_OWNER column lists the owner of the database link for the in–doubt transaction.

Additionally, the INTERFACE column tells whether the local node or a subordinate node is the commit point site.

To trace the session tree, you can query the local DBA_2PC_NEIGHBORS view. In this case, you query this view on the WAREHOUSE database.

```
SELECT * FROM sys.dba_2pc_neighbors
   WHERE local_tran_id = ’1.21.17’
   ORDER BY sess#, in_out;

Column Name             Value
----------------------  --------------------------------------
LOCAL_TRAN_ID           1.21.17
IN_OUT                  in
DATABASE                SALES.ACME.COM
DBUSER_OWNER            SWILLIAMS
INTERFACE               N
DBID                    000003F4
SESS#                   1
BRANCH                  0100
```

The columns of particular interest in this view are the IN_OUT, DATABASE, DBUSER_OWNER, and INTERFACE columns. In this example, the IN_OUT column reveals that the WAREHOUSE database is a server for the SALES database, as specified in the DATABASE column. The connection to WAREHOUSE was established through a database link from the SWILLIAMS account, as shown by the DB_OWNER column, and WAREHOUSE, nor any of its descendants, was the commit point site, as shown by the INTERFACE column.

At this point, you can contact the administrator at the located nodes and ask them to repeat Steps 2 and 3, using the global transaction ID.

**Note:** If you can directly connect to these nodes with another network, you can repeat Steps 2 and 3 yourself.

For example, the following results are returned when Steps 2 and 3 are performed at SALES and HQ, respectively.

## Manually Checking the Status of Pending Transactions at SALES.ACME.COM

```
SELECT * FROM sys.dba_2pc_pending
   WHERE global_tran_id = 'SALES.ACME.COM.55d1c563.1.93.29';

Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.93.29
GLOBAL_TRAN_ID         SALES.ACME.COM.55d1c563.1.93.29
STATE                  prepared
MIXED                  no
ADVICE
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31–MAY–91
FORCE_TIME
RETRY_TIME             31–MAY–91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   system1
DB_USER                SWILLIAMS
COMMIT#

SELECT * FROM dba_2pc_neighbors
   WHERE global_tran_id = 'SALES.ACME.COM.55d1c563.1.93.29'
   ORDER BY sess#, in_out;
```

At SALES, there are three rows for this transaction (one for the connection to WAREHOUSE, one for the connection to HQ, and one for the connection established by the user). Information corresponding to the rows for the SALES and HQ connections is listed below:

```
Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.93.29
IN_OUT                 OUT
DATABASE               WAREHOUSE.ACME.COM
DBUSER_OWNER           SWILLIAMS
INTERFACE              N
DBID                   55d1c563
SESS#                  1
BRANCH                 1

LOCAL_TRAN_ID          1.93.29
IN_OUT                 OUT
DATABASE               HQ.ACME.COM
DBUSER_OWNER           ALLEN
INTERFACE              C
DBID                   00000390
SESS#                  1
BRANCH                 1
```

The information from the previous query reveals several facts:

- SALES is the global coordinator because the local transaction ID and global transaction ID match. Also, notice that two outbound connections are established from this node, but no inbound links (this node is not a server of another node).
- HQ or one of its servers (none in this example) is the commit point site.

**Manually Checking the Status of Pending Transactions at HQ.ACME.COM:**

```
SELECT * FROM dba_2pc_pending
   WHERE global_tran_id = ’SALES.ACME.COM.55d1c563.1.93.29’;
Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.45.13
GLOBAL_TRAN_ID         SALES.ACME.COM.55d1c563.1.93.29
STATE                  COMMIT
MIXED                  NO
ACTION
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31–MAY–91
FORCE_TIME
RETRY_TIME             31–MAY–91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   SYSTEM1
DB_USER                SWILLIAMS
COMMIT#                129314
```

At this point, you have found a node that resolved the transaction. It has been committed. Therefore, you can force the in–doubt transaction to commit at your local database (see the following section for information on manually committing or rolling back in–doubt transactions). It is a good idea to contact any other administrators you know that could also benefit from your investigation.

## Step 4: Check for Mixed Outcome

After you manually force a transaction to commit or roll back, the corresponding row in the pending transaction table remains. The STATE of the transaction is changed to forced commit or forced abort, depending on how you forced the transaction.

Furthermore, once connections between the instances resume, RECO checks the global outcome of the transaction. The MIXED column is changed to yes and the row for the transaction is not deleted if you forced the transaction the wrong way.

If you ever see a transaction forced the wrong way, you should be aware that some global data inconsistency may exist. Eventually, you can purge unnecessary rows from the pending transaction table.

## The Pending Transaction Table (DBA_2PC_PENDING)

Every Oracle7 database has a *pending transaction table* which is a special table that stores information about distributed transactions as they proceed through the prepare/commit phases. You can query a database's pending transaction table by referencing the DBA_2PC_PENDING data dictionary view.

Each transaction with an entry in the pending transaction table is classified in one of the following categories (as indicated in DBA_2PC_PENDING.STATE):

| | |
|---|---|
| collecting | This category normally applies only to the global coordinator or local coordinators. The node is currently collecting information from other database servers before it can decide whether it can prepare. |
| prepared | The node has prepared and may or may not have acknowledged this to its local coordinator with a prepared message. However, no commit message has been received. The node remains prepared, holding any local resource locks necessary for the transaction to commit. |
| committed | The node (any type) has committed the transaction, but other nodes involved in the transaction may not have done the same. That is, the transaction is still pending at one or more nodes. |
| forced commit | A pending transaction can be forced to commit at the discretion of a database administrator. This entry occurs if a transaction is manually committed at a local node by a database administrator. |
| forced abort (rollback) | A pending transaction can be forced to roll back at the discretion of a database administrator. This entry occurs if this transaction is manually rolled back at a local node by a database administrator. |

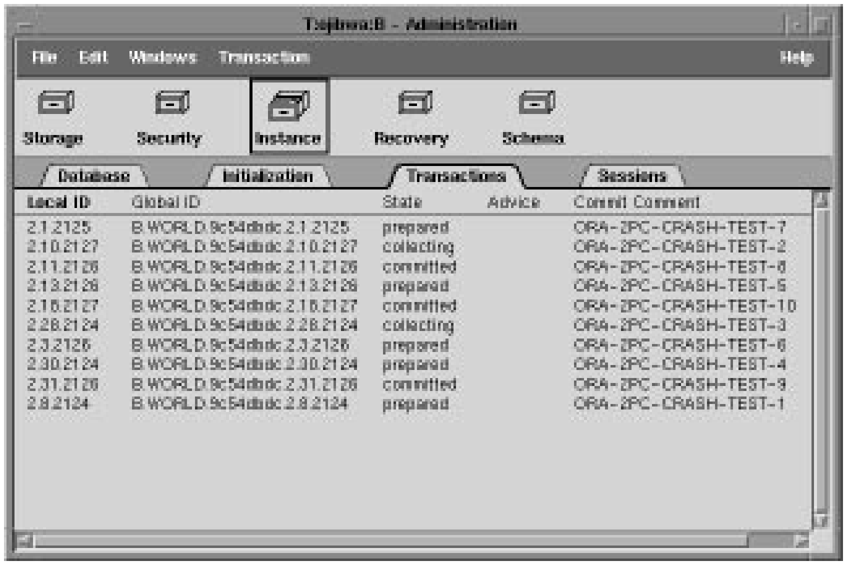
Also of particular interest in the pending transaction table is the mixed outcome flag (as indicated in DBA_2PC_PENDING.MIXED). The database administrator can make the wrong choice if a pending transaction is forced to commit or roll back (for example, the local administrator rolls back the transaction, but the other nodes commit it). Incorrect decisions are detected automatically, and the damage flag for the corresponding pending transaction's record is set (MIXED=yes).

The RECO (Recoverer) background process uses the information in the pending transaction table to finalize the status of in–doubt transactions. The information in the pending transaction table can also be used by a database administrator, who decides to manually override the automatic recovery procedures for pending distributed transactions.

All transactions automatically resolved by RECO are automatically removed from the pending transaction table. Additionally, all information about in–doubt transactions correctly resolved by an administrator (as checked when RECO reestablishes communication) are automatically removed from the pending transaction table. However, all rows resolved by an administrator that result in a mixed outcome across nodes remain in the pending transaction table of all involved nodes until they are manually deleted.

## Manually Committing In–Doubt Transactions

The local database administrator has two ways to manually force an in–doubt transaction to commit. The DBA can use the Server Manager Transaction Object List option Force Commit or the SQL command COMMIT with the FORCE option and a text string, indicating either the local or global transaction ID of the in–doubt transaction to commit. Figure 5 – 9 shows the Server Manager Transaction Object List.



| Local ID | Global ID | State | Advice | Commit Comment |
|---|---|---|---|---|
| 2.1.2125 | B.WORLD.9c54dbdc.2.1.2125 | prepared | | ORA-2PC-CRASH-TEST-7 |
| 2.10.2127 | B.WORLD.9c54dbdc.2.10.2127 | collecting | | ORA-2PC-CRASH-TEST-2 |
| 2.11.2126 | B.WORLD.9c54dbdc.2.11.2126 | committed | | ORA-2PC-CRASH-TEST-8 |
| 2.13.2126 | B.WORLD.9c54dbdc.2.13.2126 | prepared | | ORA-2PC-CRASH-TEST-5 |
| 2.16.2127 | B.WORLD.9c54dbdc.2.16.2127 | committed | | ORA-2PC-CRASH-TEST-10 |
| 2.28.2124 | B.WORLD.9c54dbdc.2.28.2124 | collecting | | ORA-2PC-CRASH-TEST-3 |
| 2.3.2126 | B.WORLD.9c54dbdc.2.3.2126 | prepared | | ORA-2PC-CRASH-TEST-6 |
| 2.30.2124 | B.WORLD.9c54dbdc.2.30.2124 | prepared | | ORA-2PC-CRASH-TEST-4 |
| 2.31.2126 | B.WORLD.9c54dbdc.2.31.2126 | committed | | ORA-2PC-CRASH-TEST-9 |
| 2.8.2124 | B.WORLD.9c54dbdc.2.8.2124 | prepared | | ORA-2PC-CRASH-TEST-1 |

**Figure 5 – 9 Transaction Object List**

## Forcing a Commit or Rollback in Server Manager

To commit an in–doubt transaction, select the transaction from the Transaction Object List and choose Force Commit from the Transaction menu.

To roll back an in–doubt transaction, select the transaction from the Transaction Object List and choose Force Rollback from the Transaction menu.

☞ **Attention:** You cannot roll back an in–doubt transaction to a savepoint.

## Manually Committing or Rolling Back In–Doubt Transactions

The following SQL statement is the command equivalent of the action taken in Figure 5 – 9 to commit an in–doubt transaction.

```
COMMIT FORCE 'transaction_name';
```

To manually rollback an in–doubt transaction, use the SQL command ROLLBACK with the FORCE option and a text string, indicating either the local or global transaction ID of the in–doubt transaction to rollback. For example, to rollback the in–doubt transaction with the local transaction ID of 2.9.4, use the following statement:

```
ROLLBACK FORCE '2.9.4';
```

☞ **Attention:** You cannot roll back an in–doubt transaction to a savepoint.

**Privileges Required to Manually Commit or Rollback In–Doubt Transactions**

To manually force the commit or rollback of an in–doubt transaction issued by yourself, you must have been granted the FORCE TRANSACTION system privilege. To force the commit or rollback of another user's distributed transaction, you must have the FORCE ANY TRANSACTION system privilege. Both privileges can be obtained either explicitly or via a role.

**Note:** Forcing the commit or rollback of an in–doubt distributed transaction does not affect the status of the operator's current transaction.

**Forcing Rollback / Commit on the Local Pending Transaction Table**

In all examples, the transaction is committed or rolled back on the local node, and the local pending transaction table records a value of forced commit or forced abort for the STATE column of this transaction's row.

**Specifying the SCN**

Optionally, you can specify the SCN for the transaction when forcing a transaction to commit. This feature allows you to commit an in–doubt transaction with the SCN assigned when it was committed at other nodes. Thus you maintain the synchronized commit time of the distributed transaction even if there is a failure. Specify an SCN only when you can determine the SCN of the same transaction already committed at another node.

For example, assume you want to manually commit a transaction with the global transaction ID *global_id*. First, query the DBA_2PC_PENDING view of a remote database also involved with the transaction in question. Note the SCN used for the commit of the transaction at that node. Specify the SCN (a decimal number) when committing the transaction at the local node. For example, if the SCN were 829381993, you would use the command:

```
COMMIT FORCE 'global_id', 829381993;
```

## Changing Connection Hold Time

If a distributed transaction fails, the connection from the local site to the remote site may not close immediately. Instead, it remains open in case communication can be restored quickly, without having to re–establish the connection. You can set the length of time that the connection remains open with the database parameter DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME.

A high value minimizes the cost of reconnecting after failures, but causes the local database to consume more resources. In contrast, a lower value minimizes the cost of resources kept locked during a failure, but increases the cost of reconnecting after failures. The default value of the parameter is 200 seconds. See the *Oracle7 Server Reference* for more information.

## Setting a Limit on Distributed Transactions

OSDoc

The database parameter DISTRIBUTED_TRANSACTIONS sets a maximum on the number of distributed transactions in which a database can participate. You should increase the value of this parameter if your database is part of many distributed transactions. The default value is operating system–specific.

In contrast, if your site is experiencing an abnormally high number of network failures, you can temporarily decrease the value of this parameter. Doing so limits the number of in–doubt transactions in which your site takes part, and thereby limits the amount of locked data at your site, and the number of in–doubt transactions you might have to resolve.

For more information on this parameter, see the *Oracle7 Server Reference.*

## Testing Distributed Transaction Recovery Features

If you like, you can force the failure of a distributed transaction to observe RECO, automatically resolving the local portion of the transaction. Alternatively, you might be interested in forcing a distributed transaction to fail so that you can practice manually resolving in–doubt distributed transactions and observing the results.

The following sections describes the features available and the steps necessary to perform such operations.

### Forcing a Distributed Transaction to Fail

Comments can be included in the COMMENT parameter of the COMMIT statement. To intentionally induce a failure during the prepare/commit phases of a distributed transaction, include the following comment in the COMMENT parameter:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

where *n* is one of the following integers:

| *n* | Effect |
|---|---|
| 1 | Crash commit point site after collect |
| 2 | Crash non–commit point site after collect |
| 3 | Crash before prepare (non–commit point site) |
| 4 | Crash after prepare (non–commit point site) |
| 5 | Crash commit point site before commit |
| 6 | Crash commit point site after commit |
| 7 | Crash non–commit point site before commit |
| 8 | Crash non–commit point site after commit |
| 9 | Crash commit point site before forget |
| 10 | Crash non–commit point site before forget |

**Table 5 – 1 Failure Values for the Parameter COMMENT**

For example, the following statement returns the following messages if the local commit point strength is greater than the remote commit point strength and both nodes are updated:

```
COMMIT COMMENT 'ORA–2PC–CRASH–TEST–7';

ORA–02054: transaction #.##.## in–doubt
ORA–02059: ORA–CRASH–TEST–n in commit comment
```

At this point, the in–doubt distributed transaction appears in the DBA_2PC_PENDING view. If enabled, RECO automatically resolves the transaction rather quickly.

### Privileges Required to Induce Prepare/Commit Phase Failures

You can induce prepare/commit phase failures via the previous comments only if the local and remote sessions have the FORCE ANY TRANSACTION system privilege. Otherwise, an error is returned if you attempt to issue a COMMIT statement with a crash comment.

## The Recoverer (RECO) Background Process

The RECO background process of an Oracle7 instance automatically resolves failures involving distributed transactions. At exponentially growing time intervals, the RECO background process of a node attempts to recover the local portion of an in–doubt distributed transaction.

RECO can use an existing connection or establish a new connection to other nodes involved in the failed transaction. When a connection is established, RECO automatically resolves all in–doubt transactions. Rows corresponding to any resolved in–doubt transactions are automatically removed from each database's pending transaction table.

## Disabling and Enabling RECO

The recoverer background process, RECO, can be enabled and disabled using the ALTER SYSTEM command with the ENABLE / DISABLE DISTRIBUTED RECOVERY options, respectively. For example, you might want to temporarily disable RECO to force the failure of a prepare / commit and manually resolve the in–doubt transaction. The following statement disables RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Alternatively, the following statement enables RECO so that in–doubt transactions are automatically resolved:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

**Note:** Single–process instances (for example, a PC running MS–DOS) have no separate background processes, and therefore no RECO process. Therefore, when a single–process instance that participates in a distributed system is started, distributed recovery must be manually enabled using the statement above.

OSDoc

**Additional Information:** See your Oracle operating system–specific documentation for more information about distributed transaction recovery for single–process instances.

CHAPTER

# 6 Security Issues

Networked systems are at greater risk than closed systems and require even greater attention to security issues. This chapter covers database security issues in a distributed system. The following topics are covered:

- network security
  - terminology
  - security methods
  - Oracle Secure Network Services
  - security through authentication methods
- database security
  - user access
  - privileges and roles
  - DBA roles

For more detailed information on how network security is implemented using authentication services such as Kerberos (V5), see *Secure Network Services Administrator's Guide*, *Understanding SQL*Net* and the *Oracle7 Server Administrator's Guide.* See the *Oracle Network Manager Administrator's Guide* for information on configuring Secure Network Services' encryption and authentication services.

## Some Terms Related to Network Security

Below are brief descriptions of some terms related to network security.

| | |
|---|---|
| **External authentication** | The process by which a client is identified and authenticated by something other than an Oracle username and password. For example, a user can be authenticated by the operating system or by a network authentication service such as Cybersafe Challenger or Kerberos. |
| **Operating–system authorized login** or **OPS$ login** | A term formerly used to refer to the process by which a user's identity is authenticated by the operating system (same as *operating system authentication*). The client passes its operating system user name to the server. The server then verifies that the user has an operating–system account by calling the operating system. |
| **Proxy logins** | Used by the server to impersonate the client in performing an operation for the client. Proxy logins are performed for database links, or server–to–server connections. Data is only as secure as the underlying protocol can provide. Security will vary depending on the underlying protocol in use, and network topology. |
| **Privileges and roles** | They control the access a user has to database functions and the schema objects within the database. |

For more information, see *Understanding SQL*Net,* and the *Oracle7 Server Administrator's Guide.*

## Introduction to Network Security

The network security provided in Secure Network Services version 2.0 and Oracle7 Server release 7.2 and higher extended the Oracle7 Server's current operating system authentication mechanisms to include network authentication services. Like the Oracle Protocol Adapters, authentication adapters are designed for a specific service, such as Kerberos, Cybersafe Challenger and SecureID Smart Cards. A site has a choice of which ones to link into their SQL*Net configuration.

> **Note:** Authentication adapters are a part of Secure Network Services version 2.0. Availability of specific adapters will be tied to the release of Secure Network Services, which is scheduled for phased introduction throughout the lifetime of SQL*Net release 2.2.

The major features related to network security in Oracle7 Server release 7.2 and Secure Network Services version 2.0 are:

- **Network encryption and message digests.** Secure Network Services allows for encryption of all network traffic, and for sequenced, cryptographic message digests to provide data integrity and defend against replay attacks.
- **Secure external authentication.** For connections made using authentication services, the database will trust the authentication service for authorization.
- **Network roles.** The database recognizes roles granted by network services, similar to the way operating system roles are recognized (where supported by the underlying authentication system).
- **Network SYSDBA/SYSOPER privilege.** Remote internal connections can be authorized by the network service instead of by the Oracle7 password file utility as in release 7.1.

> **Note:** The following feature will be supported in a future release of Secure Network Services.

- **Secure database links.** Any user who is authenticated by a network service that supports proxy logins will be authenticated using that service when calling a remote database using an anonymous database link.

New security features in release 7.2 are described in more detail below. See *Understanding SQL*Net* and *Secure Network Services Administrator's Guide* for more information on network security in release 7.2.

## Security During Connection Establishment

In addition to authenticating the identity of the user when a connection to a database is established, SQL*Net also provides the ability to accept or refuse connections on the basis of the system from which a client application is connecting. Using a pre–configured table, SQL*Net validates a connection request and determines whether the source system has the right to establish connections to the server.

This feature, called "valid node", can be administered with inclusive or exclusive authority, allowing you to explicitly list the systems from which you will and will not allow valid connections.

Valid node must be configured in PROTOCOL.ORA manually—it cannot be configured with Network Manager. Refer *Understanding SQL*Net* for information on the valid node feature.

☞ **Attention:** Use the valid node feature with caution, as it only provides the degree of assurance that the underlying protocol provides about who is on the other side of the connection.

## Password Protection and Encryption

All SQL*Net components can be protected from startup and shutdown using a password mechanism. Password protection provides additional security beyond normal connection establishment security and prevents malicious tampering with the network infrastructure. In addition, SQL*Net protects all passwords used in client applications during login by encrypting them when they are sent across the network. Password encryption, unique among database vendors, prevents passwords from being compromised by network tracing tools. For more information about password security, see the *Oracle7 Server Administrator's Guide*.

## Oracle Secure Network Services

Secure Network Services, an optional product, allows customers to protect the valuable data travelling across their networks from eavesdropping and data modification by unauthorized third parties. System architects are free to build sophisticated, distributed networks of Oracle7 databases without having to worry about the security of their data as it fans out across the globe. Even connections to third–party data repositories can be encrypted with Secure Network Services.

### Network Authentication

Unlike many competing products, the Oracle client library encrypts login passwords before they are transmitted across the network. This makes Oracle client–server applications more secure than most traditional host–based systems that used insecure terminal emulation protocols such as TELNET for PC–to–server communication. In addition, Oracle's sophisticated network authentication architecture allows for the installation of Oracle Authentication Adapters that enable Oracle to take advantage of sophisticated network security systems such as Kerberos, Cybersafe Challenger and SecureID Smart Cards, and biometric identification devices.

### Secure Network Services Ensures Tamper–Proof Data

To detect modification or replay of data during transmission, the optional Secure Network Services can generate a cryptographically secure message digest and include it with each SQL*Net packet sent across the network. Upon reception at the destination, an integrity check is immediately performed on each packet.

This makes it virtually impossible for an intruder to alter data or commands without detection, and ensures that any attempt to do so is immediately reported to the user and written to the system log files.

### Secure Network Services Provides High–Speed Global Data Encryption

To protect data from unauthorized viewing, the optional Secure Network Services includes an encryption module that uses the RSA Data Security RC4™ encryption algorithm. Using a secret, randomly–generated 40–bit key for every SQL*Net session, all user network traffic is fully safeguarded, including all data values, SQL statements, and stored procedure calls and results.

Use of the encryption module can be requested or required by either the client, the server, or both for guaranteed data protection. Oracle's highly optimized implementation provides a high degree of security for an almost imperceptible performance penalty. Since Secure Network Services Version 1.0 meets the new U.S. government export guidelines for encryption products, it is available in all but a few countries, allowing most companies to safeguard their entire worldwide operations with this software. Version 1.1 also includes DES encryption for domestic users, with crypto–algorithm negotiation between client and server at connect time.

### Secure Network Services Provides Cross–Protocol Data Security

Secure Network Services is fully supported by the Oracle MultiProtocol Interchange, making secure data transfer a reality across network protocol boundaries. Clients using LAN protocols, such as NetWare (SPX/IPX), can now securely share data with large servers using different network protocols like LU6.2, TCP/IP, or DECnet. To eliminate potential weak points in the network infrastructure and to maximize performance, Interchanges pass encrypted data from protocol to protocol without the cost and exposure of decryption and re–encryption.

## Transparent Gateway Security

Functionally, gateway security is identical to that of an Oracle7 Server, as described in *Oracle7 Server Concepts*. For SQL–based transparent gateways, Oracle7 database security is mapped to the native data dictionary of the data source. For file–based transparent gateways, the Transparent Gateway Utility is used to administer gateway users, roles, privileges, tables, and views.

## Authentication Services Provide Enhanced Security

In Secure Network Services version 2.0, it will be possible to use network authentication services to authenticate connections to the database. An authentication service is usually part of a network operating system (NOS) that overlays several machines. The purpose of authentication services is to provide enhanced security in a distributed environment with network authentication.

Network administration of machines can be centralized by creating a group of network users that have the same identity and privileges, verified by authentication services.

### Secure External Authentication

Users need to use a slash (/) to indicate the lack of a username when requesting external authentication. If an authentication adapter is available (installed and linked into the SQL*Net configuration), then the server will use it to find the user’s network identity. Alternatively, leave the username and password fields in the pop–up login box of an application (such as SQL*Plus) empty.

```
SVRMGR> CONNECT /@ny
Connected.
```

Following are some important points related to secure external authentication:

- The connection fails and an error is returned if the user does not have a valid externally–authenticated database account.
- It is recommended that you do not set REMOTE_OS_AUTHENT=TRUE initialization parameter because this exposes the database to certain security risks.
- Set REMOTE_OS_AUTHENT to TRUE only when migrating from a non–secure connection to an authentication service (that is, where some users still need to connect to this database based on the operating system authentication of their host client system.

**Note:** The REMOTE_OS_AUTHENT parameter only applies to operating system authentication, not to NOS authentication.

Using the network identity from the authentication service, the Oracle7 Server can provide secure external authentication over a non–secure protocol such as TCP / IP.

There is no change in connection syntax from Oracle7 Release 7.1.

Whether or not an authentication is available in Network Manager, the operating system username is retrieved by prepending the OS_AUTHENT_PREFIX for Oracle7. If the account exists, then login succeeds. If the connection is not secure, then the value of REMOTE_OS_AUTHENT checks if access should be granted.

## Proxy Authentication for Remote Login

Proxy authentication provides the client with a credential or token that identifies a user with valid network access by proxy authentication. See Figure 6 – 1 for an illustration of proxy authentication.

Proxy authentication enables a database link to inherit the security credentials established in the initial login.

Following are some important points related to proxy authentication:

- Proxy authentication is not supported by all network authentication services.
- Proxy authentication is used when an externally authenticated user attempts to use an anonymous database link.
- Secure database links also do not require any change in syntax from Oracle7 Server Release 7.1. Proxy authentication offers a secure and authenticated login, whereas operating system authentication will fail if no authentication service is available and REMOTE_OS_AUTHENT is not set to TRUE.

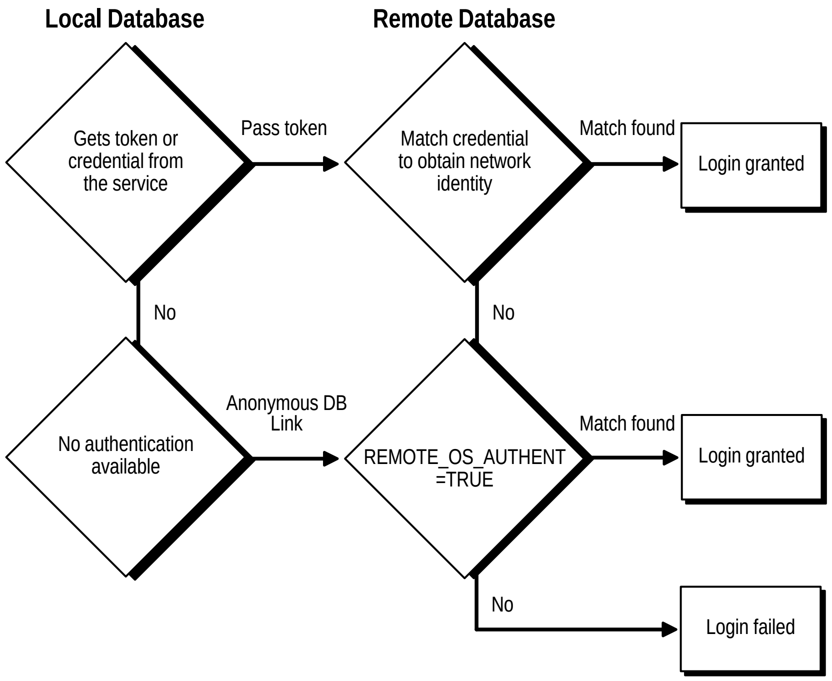


**Figure 6 – 1 Proxy Authentication for Remote Login**

## Authorization Using Network Roles

Network roles, verified by the authentication service, allow users valid network–wide access to database objects.

Use the global database name to specify roles that are valid on the various databases in the network, instead of using the SID as operating system roles do.

### Defining Oracle Network Roles

Some authentication services, such as DCE, use the following syntax to define Oracle Network roles. For example, using DCE syntax:

```
ORA_<global_db_name>_<rolename>[_[A]{D]]
```

or

```
ORA_global_<rolename>[_[A][D]]
```

“ORA_” indicates that this token applies to Oracle products.

`<global_db_name>:` specifies the database on which the token is valid.

The example shows how the literal constant “global” can be used as a valid token on all databases served by this authentication service.

[A]: represents the administrative capability for this role.

[D]: indicates this role is active by default.

**Notes:**

- The method of naming network roles can vary, depending on the authentication adapter. The previous example is the naming method for DCE authentication. However, an authentication service may use some other method than strings to denote privileges.
- Network roles are obtained from the network and not the operating system.
- The global database name consists of the local database name and the domain name.

  For example, the global database name HR.US.ACME.COM consists of the database name HR and the domain name US.ACME.COM. It must be unique within the enterprise.

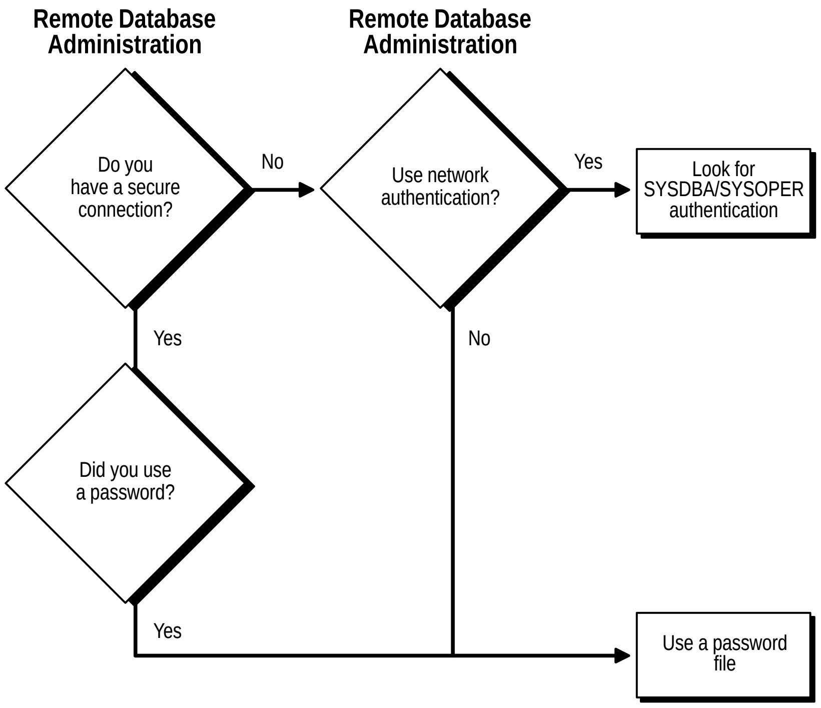
  The consequences of having two databases with the same network role is that the same role could be accessed on two different databases.
- The database reads all the user's network roles at connection time. Therefore, if the network administrator revokes a role from a user then this is not reflected until the user reconnects to the database.

## Authentication Through Network Privileges

Authentication through network privileges alleviates the need to maintain a password file by providing network privileges that map to the operating system privileges (SYSDBA and SYSOPER). The DBA or network administrator can make remote administration over a non–secure connection possible by using network privileges. See Figure 6 – 2 for an illustration of authentication through network privileges.

### How a Remote User is Authenticated through Network Privileges

The benefit of authentication through network privileges is that it removes the need to maintain a password file by providing network privileges that map to the operating system privileges (SYSDBA and SYSOPER). This makes secure remote administration possible over a non–secure network connection.



Remote Database Administration
Remote Database Administration
Do you have a secure connection?
No
Use network authentication?
Yes
Look for SYSDBA/SYSOPER authentication
Yes
No
Did you use a password?
Yes
Use a password file

**Figure 6 – 2 Authentication Through Network Privileges**

If a remote user tries to connect to a local database through a non–secure connection, the local database checks whether it should use network authentication (by checking the appropriate parameters). If the parameters indicate that network authentication should be used, then the database attempts to authenticate the user by looking for SYSDBA / SYSOPER authentication.

If a remote user tries to connect over a secure connection by using a password, the password is verified using the authentication service first. If that fails, the database performs the verification.

Following are some points related to authentication through network privileges:

- Some authentication adapters implement network privileges in the same manner that operating systems implement the SYSOPER and SYSDBA roles.
- Use the global database name to promote consistency, and prevent overlap with the namespace set aside for external roles. See *Secure Network Services Administrator's Guide* for more information.

Example:

ORA_<*global db name*>_[DBA|OPER]_SYS

This naming convention eases administration of multi–instance configurations.

**Note:** The method of naming network privileges can vary, depending on the authentication adapter. The previous example is the naming method for DCE authentication.

- Use the global database name to request internal connections.

For successful internal connections, the database instance should be up and running. Otherwise, the database name cannot be read from the control file.

The following points relate to authentication through network privileges:

- To verify initial connection, the name defined in the CONNECT_DATA string is used to fetch the proper privileges when the instance is down.
- Because the connect descriptor is clear text and can be changed, the Oracle startup code performs a consistency check and does not allow the connection to pass through if a match is not found.

## Database Security in a Distributed System

Because a distributed system allows many users to access the same data, you will need some method of limiting access to certain types of data. For example, you would want employee salaries to be available only to those authorized to see them. You may also have multiple databases in your system and will need to control who, specifically, has access to the database as a whole.

Oracle allows you to limit user access on several levels through:

- username / password
- privileges and roles

### User Access

User authentication can be performed by identifying the user with a username / password registered with the database, by operating system authentication of the user's identity, or by network authentication.

⚠ **Warning:** In a distributed system, operating system authentication of the user's identity is very susceptible to security breach. Therefore, even though operating system authentication can be performed, it is strongly discouraged.

## Privileges and Roles

*Privileges* control what a user can and cannot access or do while attached to a database. Privileges grant the right to a user to execute a particular type of SQL statement or the right to access another user's object. Oracle has two types of privileges: system privileges and object privileges.

Privileges are also defined by who they will be assigned to: the DBA or the user. For example, some DBA privileges might be:

- CREATE USER
- DROP ANY TABLE
- ALTER ANY TRIGGER

*Roles* provide a way of assigning a predetermined group of privileges to users. A role groups several privileges so that they can be simultaneously granted to or revoked from users.

See the *Oracle7 Server Administrator's Guide* for a full explanation of privileges and roles and how they are created, altered, enabled, disabled, and dropped.

**Additional Information:** Server Manager can greatly simplify creating roles. See the *Oracle Server Manager User's Guide* for more information.

## DBA Roles

There are approximately 80 different system privileges. So, in a large distributed system, it may be necessary to divide DBA responsibilities among several people. Such a division also limits potential security problems if a single DBA's account is compromised.

This can be accomplished by creating specialized DBA roles for, for example, backups, IMPORT / EXPORT, security, and so on.

The roles listed in Table 6 – 1 are automatically defined for Oracle databases. These roles are provided for backward compatibility to earlier versions of Oracle. You can grant and revoke privileges and roles to these predefined roles, as you can to any role you define.

| Role Name | Privileges Granted To Role |
|---|---|
| CONNECT[1] | ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW |
| RESOURCE[1,2] | CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER |
| DBA[1,3,4] | All system privileges WITH ADMIN OPTION |
| EXP_FULL_DATABASE[5] | SELECT ANY TABLE, BACKUP ANY TABLE, INSERT, DELETE, AND UPDATE ON THE TABLES SYS.INCVID, SYS.INCFIL, AND SYS.INCEXP |
| IMP_FULL_DATABASE[5] | BECOME USER, WRITEDOWN[6] |

**Table 6 – 1 DBA Roles and their Privileges**

[1] *created by SQL.BSQ*

[2] *grantees of the RESOURCE role also receive the UNLIMITED TABLESPACE system privilege as an explicitly grant (not as part of the RESOURCE role)*

[3] *grantees of the DBA role also receive the UNLIMITED TABLESPACE system privilege with the ADMIN OPTION as an explicit grant (not as part of the DBA role). Therefore when the DBA role is revoked, any explicit grant of UNLIMITED TABLESPACE is also revoked.*

[4] *also includes the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles if CATEXP.SQL has been run*

[5] *created by CATEXP.SQL*

[6] *a Trusted ORACLE privilege only; see the Trusted ORACLE7 Server Administrator's Guide*

CHAPTER

7

# Application Development

This chapter describes the special considerations that are necessary if you are designing an application to run in a distributed system. *Oracle7 Server Concepts* describes how Oracle eliminates much of the need to design applications specifically to work in a distributed environment. The topics covered include:

- development tools
- factors in distributing an application's data
- object naming considerations
- how to control connections via database links
- factors affecting referential integrity
- error handling
- XA–related information

The *Oracle7 Server Application Developer's Guide* provides a complete discussion of implementing Oracle7 applications. This chapter provides information specific to development for an Oracle7 distributed environment. Basic replication and the advanced replication option are discussed in *Oracle7 Server Distributed Systems, Volume II.* See the *Trusted Oracle7 Server Administrator's Guide* for additional information about designing an application to run in a multilevel distributed system.

## Development Tools

Although applications can be created using 3GL, 4GL, or SQL code, Oracle's Cooperative Development Environment (CDE) tool set provides a unified system for developing Oracle7 distributed applications.

### Case Tools

Oracle's primary CDE tools are:

- CASE*Designer: provides a graphical front–end for database design
- CASE*Dictionary: provides a repository for system development information
- CASE*Generator: generates application code modules

Oracle's CDE tools assist you in all phases of application development from design through implementation. It also simplifies the task of porting existing applications to different operating systems and platforms

**Additional Information:** See the CASE tool set documentation for full explanations of how to use CASE*Designer, CASE*Dictionary, and CASE*Generator to develop your applications.

## Factors Affecting the Distribution of an Application's Data

In a distributed database environment, you should coordinate with the database administrator to determine the best location for the data. Some issues to consider are:

- number of transactions posted from each location
- amount of data (portion of table) used by each node
- performance characteristics and reliability of the network
- speed of various nodes, capacities of disks
- criticality of access if a node or link is down
- need for referential integrity among tables

## Naming Objects

You should decide when you want to use partial and complete global object names in the definition of views, synonyms, and procedures. Keep in mind that database names should be stable and databases should not be unnecessarily moved within a network.

In a distributed system, each database must have a unique global name. The global name is composed of the database name and the network domain that contains the database. Each object in the database then has a global object name consisting of the object name and the global database name. Because Oracle ensures that the object name is unique within a database, you can ensure that it is unique across all databases by assigning unique global database names. You should coordinate with your database administrator on this task, as it is usually the DBA who is responsible for assigning database names. See "Global Naming Issues," on page 2 – 26, for more information on object naming.

## Controlling Connections Established by Database Links

When a global object name is referenced in a SQL statement or remote procedure call, database links establish a connection to a session in the remote database on behalf of the local user. The remote connection and session are only created if the connection has not already been established previously for the local user session.

The connections and sessions established to remote databases persist for the duration of the local user's session, unless the application (or user) explicitly terminates them. Terminating remote connections established using database links is useful for disconnecting high cost connections (such as long distance phone connections) that are no longer required by the application.

The application developer or user can close (terminate) a remote connection and session using the ALTER SESSION command with the CLOSE DATABASE LINK parameter. For example, assume you issue the following query:

```
SELECT * FROM emp@sales;
COMMIT;
```

The following statement terminates the session in the remote database pointed to by the SALES database link:

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

If partial global database names are specified, the local Oracle expands the name using the network domain of the local database.

> **Note:** Before closing a database link, you must first close all cursors that use the link and then end your current transaction if it uses the link.

To close a database link connection in your user session, you must have the ALTER SESSION system privilege.

## Referential Integrity in a Distributed System

Oracle does not permit declarative referential integrity constraints to be defined across nodes of a distributed system (that is, a declarative referential integrity constraint on one table cannot specify a foreign key that references a primary or unique key of a remote table). However, parent/child table relationships across nodes can be maintained using triggers. For more information about triggers that enforce referential integrity, see Chapter 7 of *Oracle7 Server Concepts*.

> **Note:** If you decide to define referential integrity across the nodes of a distributed database using triggers, be aware that network failures can limit the accessibility of not only the parent table, but also the child table.
>
> For example, assume that the child table is in the SALES database and the parent table is in the HQ database. If the network connection between the two databases fails, some DML statements against the child table (those that insert rows into the child table or update a foreign key value in the child table) cannot proceed because the referential integrity triggers must have access to the parent table in the HQ database.

## Distributed Queries

You can use a trigger or stored procedure to create a distributed query. This distributed query is decomposed by the local Oracle into a corresponding number of remote queries, which are sent to the remote nodes for execution. The remote nodes execute the queries and send the results back to the local node. The local node then performs any necessary post–processing and returns the results to the user or application.

If a portion of a distributed statement fails, for example, due to an integrity constraint violation, Oracle returns error number ORA–02055. Subsequent statements or procedure calls return error number ORA–02067 until a rollback or rollback to savepoint is issued.

You should design your application to check for any returned error messages that indicate that a portion of the distributed update has failed. If you detect a failure, you should rollback the entire transaction (or rollback to a savepoint) before allowing the application to proceed.

## Handling Errors in Remote Procedures

When a procedure is executed locally or at a remote location, four types of exceptions can occur:

- PL/SQL user–defined exceptions, which must be declared using the keyword EXCEPTION.
- PL/SQL predefined exceptions, such as NO_DATA_FOUND keyword.
- SQL errors, such as ORA–00900 and ORA–02015.
- Application exceptions, which are generated using the RAISE_APPLICATION_ERROR() procedure.

When using local procedures, all of these messages can be trapped by writing an exception handler, such as shown in the following example:

```
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   /* ...handle the exception */
```

Notice that the WHEN clause requires an exception name. If the exception that is raised does not have a name, such as those generated with RAISE_APPLICATION_ERROR, one can be assigned using PRAGMA_EXCEPTION_INIT, as shown in the following example:

```
...
null_salary EXCEPTION;
PRAGMA EXCEPTION_INIT(null_salary, –20101);
...
RAISE_APPLICATION_ERROR(–20101, ’salary is missing’);
...
WHEN null_salary THEN
...
```

When calling a remote procedure, exceptions are also handled by creating a local exception handler. The remote procedure must return an error number to the local, calling procedure, which then handles the exception as shown in the previous example. Because PL/SQL user–defined exceptions always return ORA–06510 to the local procedure, these exceptions cannot be handled. All other remote exceptions can be handled in the same manner as local exceptions.

## XA Library-Related Information

### General Information about the Oracle XA Library

For preliminary reading and additional reference information regarding the Oracle XA library, see the following documents:

- *Programmer's Guide to the Oracle Precompilers*
- *Programmer's Guide to the Oracle Call Interface*

OSDoc

**Additional Information:** For information on library linking filenames, see your Oracle operating system–specific documentation.

### README.doc

OSDoc

A README.doc file is located in a directory specified in your Oracle operating system–specific documentation and describes changes, bugs, or restrictions in the Oracle XA library for your platform since the last version.

### Basic Architecture

The Oracle XA library is an external interface that allows global transactions to be coordinated by a transaction manager other than the Oracle7 Server. This allows inclusion of non-Oracle7 Server entities called resource managers (RM) in distributed transactions.

The Oracle XA library conforms to the X/Open Distributed Transaction Processing (DTP) software architecture's XA interface specification.

For a general overview of XA, including basic architecture, see *X/Open CAE Specification – Distributed Transaction Processing: The XA Specification*.

You can obtain a copy of this document by requesting X/Open Document No. XO/CAE/91/300 or ISBN 1 872630 24 3 from:

X/Open Company, Ltd.
1010 El Camino Real, Suite 380
Menlo Park, CA 94025
U.S.A.

## X/Open Distributed Transaction Processing (DTP)

The X/Open DTP architecture defines a standard architecture or interface that allows multiple application programs to share resources, provided by multiple, and possibly different, resource managers. It coordinates the work between application programs and resource managers into global transactions.

Figure 7 – 1 illustrates a possible X/Open DTP model.

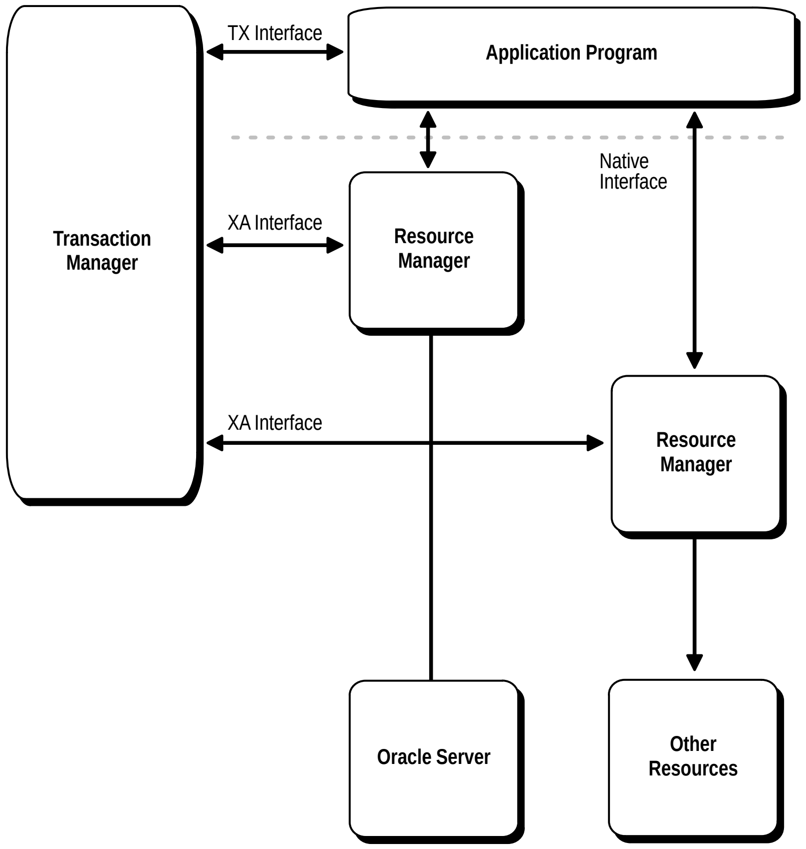


**Figure 7 – 1  One Possible DTP Model**

A resource manager (RM) controls a shared, recoverable resource that can be returned to a consistent state after a failure. For example, Oracle7 Server is an RM and uses its redo log and undo segments to return to a consistent state after a failure. An RM provides access to shared resources such as a database, file systems, printer servers, and so forth.

A transaction manager (TM) provides an application program interface (API) for specifying the boundaries of the transaction and manages the commit and recovery procedures.

Normally, Oracle7 Server acts as its own TM and manages its own commit and recovery. However, using a standards-based TM allows Oracle7 Server to cooperate with other heterogeneous RMs in a single transaction.

A TM is usually a component provided by a transaction processing monitor (TPM) vendor. The TM assigns identifiers to transactions, and monitors and coordinates their progress. It uses Oracle XA library subroutines to tell Oracle7 Server how to process the transaction, based on its knowledge of all RMs in the transaction. You can find a list of the XA subroutines and their descriptions later in this section.

An application program (AP) defines transaction boundaries and specifies actions that constitute a transaction. For example, an AP can be a precompiler or OCI program. The AP operates on the RM's resource through the RM's native interface, for example SQL. However, it starts and completes all transaction operations via the transaction manager through an interface called TX. The AP itself does not directly use the XA interface.

> **Note:** The naming conventions for your TX interface and associated subroutines are vendor-specific, and may differ from those used here. For example, you may find that the **tx_open** call is referred to as **tp_open** on your system. To check on terminology, see the documentation supplied with your transaction processing monitor.

## Transaction Recovery Management

The Oracle XA library interface follows the two-phase commit protocol, consisting of a prepare phase and a commit phase, to commit transactions.

In phase one, the prepare phase, the TM asks each RM to guarantee the ability to commit any part of the transaction. If this is possible, then the RM records its prepared state and replies affirmatively to the TM. If it is not possible, the RM may roll back any work, reply negatively to the TM, and forget any knowledge about the transaction. The protocol allows the application, or any RM, to roll back the transaction unilaterally until the prepare phase is complete.

In phase two, the commit phase, the TM records the commit decision. Then the TM issues a commit or rollback to all RMs which are participating in the transaction. Note that a TM can issue a commit for an RM only if all RMs have replied affirmatively to phase one.

## Compatibility Issues

The XA library supplied with release 7.3 can be used only against a release 7.3 server. You must use the release 7.2 XA library against a 7.2 server.

## Oracle XA Library Interface Subroutines

The Oracle XA library subroutines allow a TM to instruct Oracle7 Server what to do about transactions. Generally, the TM must "open" the resource (using **xa_open).** Typically, this will result from the AP's call to **tx_open.** Some TMs may call **xa_open** implicitly, when the application begins. Similarly, there is a close (using **xa_close**) that occurs when the application is finished with the resource. This may be when the AP calls **tx_close** or when the application terminates.

There are several other tasks the TM instructs the RMs to do. These include among others:

- starting a new transaction and associating it with an ID
- rolling back a transaction
- preparing and committing a transaction

## XA Library Subroutines

The following XA Library subroutines are available:

| | |
|---|---|
| **xa_open** | connect to the resource manager |
| **xa_close** | disconnect from the resource manager |
| **xa_start** | start a new transaction and associate it with the given transaction ID (XID), or associate the process with an existing transaction |
| **xa_end** | disassociate the process from the given XID |
| **xa_rollback** | roll back the transaction associated with the given XID |
| **xa_prepare** | prepare the transaction associated with the given XID. This is the first phase of the two-phase commit protocol. |
| **xa_commit** | commit the transaction associated with the given XID. This is the second phase of the two-phase commit protocol. |
| **xa_recover** | retrieve a list of prepared, heuristically committed or heuristically rolled back transactions |
| **xa_forget** | forget the heuristic transaction associated with the given XID |
| **xa_complete** | wait for completion of an asynchronous operation |

In general, the AP does not need to worry about these subroutines except to understand the role played by the **xa_open** string.

## Transaction Processing Monitors (TPMs)

Under UNIX, for example, a transaction processing monitor (TPM) coordinates the flow of transaction requests between the client processes that issue requests and the back-end servers that process them. Basically, a TPM coordinates transactions that require the services of several different types of back-end processes, such as application servers and resource managers that are distributed over a network.

The TPM synchronizes any commits and rollbacks required to complete a distributed transaction. The transaction manager (TM) portion of the TPM is responsible for controlling when distributed commits and rollbacks take place. Thus, if a distributed application program is written to take advantage of a TPM, the TM portion of the TPM is responsible for controlling the two-phase commit protocol. The RMs enable the TMs to do this.

Because the TM controls distributed commits or rollbacks, it must communicate directly with Oracle (or any other resource manager). It communicates through the Oracle XA library interface.

## Required Public Information

As a resource manager, Oracle is required to publish the following information:

| | |
|---|---|
| **xa_switch_t** structure | The Oracle Server **xa_switch_t** structure name is **xaosw.** This structure contains entry points and other information for the resource manager. |
| **xa_switch_t** resource mgr | The Oracle Server resource manager name within the **xa_switch_t** structure is **Oracle_XA.** |
| close string | The close string used by **xa_close ()** is ignored and may be null. |
| open string | The format of the open string used by **xa_open ()** is described in detail in the section titled "Developing and Installing Applications that use the XA Libraries" on page 7 – 12. |
| libraries | Libraries needed to link applications using Oracle XA have operating system–specific names. In general, it is similar to linking an ordinary precompiler or OCI program except with one additional library (**$ORACLE_HOME/lib/libxa.a**), and any TPM-specific libraries. |

OSDoc

See your Oracle operating system–specific documentation and the Oracle XA README.doc file to find the correct library name.

|  |  |
|---|---|
|  | Use of the Oracle XA library restricts the use of SQL to connect to and disconnect from the database and to perform transaction operations. These restrictions are described in "Interfacing to Precompilers and OCIs" on page 7 – 18. |
| requirements | You must have purchased and installed the distributed database option. |

## Additional XA Issues

Note the following additional information about Oracle XA issues:

|  |  |
|---|---|
| DDL statements | Oracle applications in global transactions may not perform DDL statements (like CREATE TABLE, DROP TABLE, CREATE INDEX) because they force a commit of pending work. |
| global and local transactions | When an application uses the XA library to connect to the Oracle Server, that application can only operate on global transactions. The Oracle Server does not allow local transactions when it is using XA. |
| transaction branches | Oracle Server transaction branches within the same global transaction share locks in a tightly coupled manner. However, if the branches are on different instances when running Oracle Parallel Server, then they are loosely coupled. |
| read-only | Oracle Server supports the read-only optimization. |
| association migration | Oracle Server does not support association migration (a means whereby a transaction manager may resume a suspended branch association in another branch). |
| dynamic registration | The optional XA feature dynamic registration is not supported. |
| asynchronous calls | The optional XA feature asynchronous XA calls is not supported. |

## Developing and Installing Applications that Use the XA Libraries

This section discusses developing and installing Oracle7 Server applications. It describes the responsibilities of both the DBA, or system administrator, and the application developer. It also defines how to construct the open string.

### Responsibilities of the DBA or System Administrator

The responsibilities of the DBA or system administrator are:

1. Define the open string with the application developer's help.

   How to define the open string is described on page 7 – 13.

2. Make sure the V$XATRANS$ view exists on the database.

   This view should have been created during the XA library installation. You can manually create the view if needed by running the SQL script XAVIEW.SQL**.** This SQL script should be executed as the Oracle user SYS. Grant the SELECT privilege to the V$XATRANS$ view for all Oracle Server accounts which will be used by Oracle XA library applications.

   OSDoc

   **Additional Information:** See your Oracle operating system–specific documentation for the location of the XAVIEW.SQL script.

3. Define additional server security groups if needed.

   Server security groups are defined by the open string field, **GPwd=P/***group_password.*

4. Install the resource manager, using the open string information, into the TPM configuration, following the TPM vendor instructions.

   The DBA or system administrator should be aware that a TPM system will start the process that connects to an Oracle7 Server. See your TPM documentation to determine what environment exists for the process and what user ID it will have.

   Be sure that correct values are set for ORACLE_HOME and ORACLE_SID.

   Also be sure to grant the user ID write permission to the directory in which the XA trace file will be written. See "Defining the Open String" on page 7 – 13 for information on how to specify a *sid* or a trace directory that is different from the defaults.

5. Start up the relevant databases to bring Oracle XA applications online.

   This should be done before starting any TPM servers.

**Responsibilities of the Application Developer**

The application developer's responsibilities are:

1. Define the open string with the DBA or system administrator's help.

   Defining the open string is described later in this section.

2. Develop the applications.

   Observe special restrictions on transaction-oriented SQL statements for precompilers. See "Interfacing to Precompilers and OCIs," on page 7 – 18.

3. Link the application according to TPM vendor instructions.

OSDoc

See your Oracle operating system–specific documentation for specific information on the exact libraries needed.

## Defining the Open String

The open string is used by the transaction monitor to open the database. The maximum number of characters in an open string is 256. The syntax of the open string is summarized as follows; later sections provide more detail on required and optional fields.

**Oracle_XA**{+*required_fields...*} [+*optional_fields...*]

where *required_fields* are:

**Acc=P//**

or

**Acc=P**/*user/password*

**SesTm=***session_time_limit*

and where *optional_fields* are:

**DB=***db_name*

**GPwd=P/***group_password*

**LogDir=***log_dir*

**MaxCur=***maximum_#_of_open_cursors*

**SqlNet=***connect_string*

Note the following:

- You can enter the required fields and optional fields *in any order* when constructing the open string.
- All field names are case insensitive. Their values may or may not be case-sensitive depending on the platform.
- There is no way to use the "+" character as part of the actual information string.

## Required Fields

Required fields for the open string are described in this section.

**Acc=P//**

or

**Acc=P***/user/password*

| | |
|---|---|
| **Acc** | Specifies user access information. |
| **P** | Indicates that explicit user and password information is provided. |
| **P//** | Indicates that no explicit user or password information is provided and that the operating system authentication form will be used.<br><br>For more information see "Managing Users and Resources," in Chapter 12 of the *Oracle7 Server Administrator's Guide.* |
| *user* | A valid Oracle Server account. |
| *password* | The corresponding current password. |

For example, **Acc=P/scott/tiger** indicates that user and password information is provided. In this case, the user is **scott** and the password is **tiger**.

As previously mentioned, make sure that **scott** has the SELECT privilege on the V$XATRANS$ table.

**Acc=P//** indicates that no user or password information is provided, thus defaulting to operating system authentication.

**SesTm=***session_time_limit*

| | |
|---|---|
| **SesTm** | Specifies the maximum length of time a transaction can be inactive before it is automatically deleted by the system. |

| | |
|---|---|
| *session_time_limit* | This value should be the maximum time allowed in a transaction between one service and the next, or a service and the commit or rollback of the transaction. |
| | For example, if the TPM uses remote procedure calls between the client and the servers, then SesTM applies to the time between the completion of one RPC and the initiation of the next RPC, or the **tx_commit**, or the **tx_rollback**. |
| | The unit for this time limit is in seconds. The value of 0 indicates no limit, but entering a value of 0 is strongly discouraged. For example, **SesTM=15** indicates that the session idle time limit is 15 seconds. |

## Optional Fields

Optional fields are described below.

**DB=***db_name*

| | |
|---|---|
| **DB** | Specifies the database name |
| *db_name* | Indicates the name used by Oracle precompilers to identify the database. |
| | Application programs that use only the default database for the Oracle precompiler (that is, they do not use the AT clause in their SQL statements) should omit the **DB=***db_name* clause in the open string. |
| | Applications that use explicitly named databases should indicate that database name in their **DB=***db_name* field. |
| | OCI programs need to call the **sqlld2()** function to obtain the correct connection. See "Using OCIs with the Oracle XA Library," on page 7 – 20. |
| | The *db_name* is not the *sid* and is not used to locate the database to be opened. Rather, it correlates the database opened by this open string with the name used in the application program to execute SQL statements. The *sid* is set from either the environment variable ORACLE_SID of the TPM application server or the *sid* given in the SQL*Net clause in the open string. The SQL*Net clause is described later in this section. |

Some TPM vendors provide a way to name a group of servers that use the same open string. The DBA may find it convenient to choose the same name both for that purpose and for *db_name*.

For example, **DB=payroll** indicates that the database name is “payroll”, and that the application server program will use that name in AT clauses.

For more information about precompilers (specifically Pro*C / C++), see “Interfacing to Precompilers and OCIs” on page 7 – 18.

**GPwd=P/***group_password*

| | |
|---|---|
| **GPwd** | Specifies the server security group password. |
| **P** | Indicates that an explicit server security group password is currently provided. |
| *group_password* | Indicates the actual server security group password. |

Server security groups provide an optional extra layer of protection between different applications running against the same Oracle Server instance. If no server security group option is specified, then the application using this open string will be part of an Oracle Server-defined server security group.

A transaction must be executed wholly within a server security group. For example, if a debit application specifies a different Oracle Server security group than a credit application, then the two may not be used in the same transaction.

For example, **GPwd=P/banking** indicates that the current server group password is “banking”.

**LogDir=***log_dir*

| | |
|---|---|
| **LogDir** | Specifies the directory on a local machine where the Oracle XA library error and tracing information may be logged. |
| *log_dir* | Indicates the pathname of the directory where the tracing information should be stored. The default is **$ORACLE_HOME/rdbms/log** if ORACLE_HOME is set, otherwise it is the current directory. |

For example, **LogDir=/xa_trace** indicates that the error and tracing information is located under the **/xa_trace** directory.

**Note:** Ensure that the directory you specify for logging exists and the application server can write to it, otherwise useful trace files may be lost without any error indication.

**MaxCur=***maximum_#_of_open_cursors*

| | |
|---|---|
| **MaxCur** | Specifies the number of cursors to be alocated when the database is opened. It serves the same purpose as the precompiler option **maxopencursors**. |
| *maximum_#_of_ open_cursors* | Indicates the number of open cursors to be cached. |

For example, **MaxCur=5** indicates that the precompiler should try to keep five open cursors cached.

**Note:** This parameter overrides the precompiler option **maxopencursors** that you might have specified in your source code or at compile time.

For more information on **maxopencursors**, see Chapter 6, Running the Oracle Precompilers," in the *Programmer's Guide to the Oracle Precompilers.*

**SqlNet=***db_link*

| | |
|---|---|
| **SqlNet** | Specifies the SQL*Net database link. |
| *db_link* | Indicates the string to use to log on to the system. The syntax for this string is the same as that used to set the TWO-TASK environment variable.<br><br>See your SQL*Net documentation for information about SQL*Net database links. |

For example, **SqlNet=hqfin@NEWDB** indicates the database with *sid***=NEWDB** accessed at host **hqfin** by TCP/IP.

The **SqlNet** parameter can be used to specify the ORACLE_SID in cases where you cannot control the server environment variable. It must also be used when the server needs to access more than one Oracle Server database. To use the SQL*Net string without actually accessing a remote database, use the Pipe driver.

For example:

```
SqlNet=localsid1
```

where:

| | |
|---|---|
| localsid1 | is an alias defined in the SQL*net **tnsnames.ora** file. |

Make sure that all databases to be accessed with a SQL*Net database link have an entry in **/etc/oratab**.

## Interfacing to Precompilers and OCIs

This section describes how to use the Oracle XA library with precompilers and Oracle Call Interfaces (OCIs).

### Using Precompilers with the Oracle XA Library

When used in an Oracle XA application, cursors are valid only for the duration of the transaction. Explicit cursors should be opened after the transaction begins, and closed before the commit or rollback. Also, you must use the **release_cursor=yes** option when compiling your precompiler application.

There are two options to choose from when interfacing with precompilers:

- using precompilers with the default database
- using precompilers with a named database

The following examples use the precompiler Pro*C / C++.

#### Using Precompilers with the Default Database

To interface to a precompiler with the default database, make certain that the **DB=***db_name* field, used in the open string, is not present. The absence of this field indicates the default connection as defined by **sqllib**, and only one default connection is allowed per process.

The following is an example of an open string identifying a default Pro*C / C++ connection.

```
ORACLE_XA+SqlNet=host@MAIL+ACC=P/scott/tiger+GPwD=P/mailgrp
    +SesTM=10+LogDir=/usr/local/logs
```

Note that the **DB=***db_name* is absent, indicating an empty database ID string.

The syntax of a SQL statement would be:

```
EXEC SQL UPDATE EMP SET SAL = sal*1.5;
```

## Using Precompilers with a Named Database

To interface to a precompiler with a named database, include the **DB=***db_name* field in the open string. Any database you refer to must reference the same *db_name* you specified in the corresponding open string.

An application may include the default database, as well as one or more named databases, as shown in the following examples.

For example, suppose you want to update an employee's salary in one database, her department number (DEPTNO) in another, and her manager in a third database. You would configure the following open strings in the transaction manager:

```
ORACLE_XA+DB=MANAGERS+SqlNet=hqfin@SID1+ACC=P/scott/tiger
    +GPwd+P/pay+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+DB=PAYROLL+SqlNet=SID2+ACC=P/scott/tiger
    +GPwd=P/mgr+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+SqlNet=hqemp@SID3+ACC=P/scott/tiger+GPwd=P/mgr
    +SesTM=10+LogDir=/usr/local/xalog
```

Note that there is no **DB=***db_name* field in the last open string.

In the application server program, you would enter declarations such as:

```
EXEC SQL DECLARE PAYROLL DATABASE;
EXEC SQL DECLARE MANAGERS DATABASE;
```

Again, the default connection (corresponding to the third open string that does not contain the **db_name** field) needs no declaration.

When doing the update, you would enter statements similar to the following:

```
EXEC SQL AT PAYROLL UPDATE EMP SET SAL=4500 WHERE EMPNO=7788;
EXEC SQL AT MANAGERS UPDATE EMP SET MGR=7566 WHERE EMPNO=7788;
EXEC SQL UPDATE EMP SET DEPTNO=30 WHERE EMPNO=7788;
```

There is no AT clause in the last statement because it is referring to the default database.

In Oracle precompilers version 1.5.3 or later, you can use a character host variable in the AT clause, as the following example shows:

```
EXEC SQL BEGIN DECLARE SECTION;
    DB_NAME1 CHARACTER(10);
    DB_NAME2 CHARACTER(10);
EXEC SQL END DECLARE SECTION;
      .
      .
SET DB_NAME1 = 'PAYROLL'
SET DB_NAME2 = 'MANAGERS'
      .
      .
```

```
EXEC SQL AT :DB_NAME1 UPDATE...
EXEC SQL AT :DB_NAME2 UPDATE...
```

**Additional Information:** For more information on concurrent logons, see the *Programmer's Guide to the Oracle Precompilers.*

**Note:** Applications using XA should not create Oracle Server database connections of their own. Any work performed by them would be outside the global transaction, and may confuse the connection information used by the Oracle XA library.

### Using OCI with the Oracle XA Library

OCI applications that use the Oracle XA library should not call **olon()** or **orlon()** to log on to the resource manager. Rather, the logon should be done through the TPM. The applications can execute the function **sqlld2()** to obtain the **lda** structure they need to access the resource manager.

Because an application server can have multiple concurrent open Oracle Server resource managers, it should call the function **sqlld2()** with the correct arguments to obtain the correct **lda** structure.

If **DB=***db_name* is not present in the open string, then execute:

```
sqlld2(lda, NULL, 0);
```

to obtain the **lda** for this resource manager.

Alternatively, if **DB=***db_name* is present in the open string, then execute:

```
sqlld2(lda, db_name, strlen(db_name));
```

to obtain the **lda** for this resource manager.

**Additional Information:** For more information about using the **sqlld2()** function, see the *Programmer's Guide to the Oracle Call Interface.*

## Transaction Control

This section explains how to use transaction control within the Oracle XA library environment.

When the XA library is used, transactions are not controlled by the SQL statements which commit or roll back transactions. Rather, they are controlled by an API accepted by the TM which starts and stops transactions. Most of the TMs use the TX interface for this. It includes the following functions:

| | |
|---|---|
| **tx_open** | logs into the resource manager(s) |
| **tx_close** | logs out of the resource manager(s) |
| **tx_begin** | starts a new transaction |
| **tx_commit** | commits a transaction |
| **tx_rollback** | rolls back the transaction |

Most TPM applications are written using a client-server architecture where an application client requests services and an application server provides services. The examples that follow use such a client-server model. A service is a logical unit of work, which in the case of the Oracle Server as the resource manager, comprises a set of SQL statements that perform a related unit of work.

For example, when a service named "credit" receives an account number and the amount to be credited, it will execute SQL statements to update information in certain tables in the database. In addition, a service might request other services. For example, a "transfer fund" service might request services from a "credit" and "debit" service.

Usually application clients request services from the application servers to perform tasks within a transaction. However, for some TPM systems, the application client itself can offer its own local services.

You can encode transaction control statements within either the client or the server; as shown in the examples.

To have more than one process participating in the same transaction, the TPM provides a communication API that allows transaction information to flow between the participating processes. Examples of communications APIs include RPC, pseudo-RPC functions, and send / receive functions.

Because the leading vendors support different communication functions, the examples that follow use the communication pseudo-function **tpm_service** to generalize the communications API.

X / Open has included several alternative methods for providing communication functions in their preliminary specification. At least one of these alternatives is supported by each of the leading TPM vendors.

## Examples of Precompiler Applications

The following examples illustrate precompiler applications. Assume that the application servers have already logged onto the TPM system, in a TPM-specific manner.

The first example shows a transaction started by an application server, and the second example shows a transaction started by an application client.

**Example 1:** Transaction started by an application server.

***Client:***

```
tpm_service(“ServiceName“);            /* Request Service*/
```

***Server:***

```
ServiceName()
{
<get service specific data>
tx_begin();        /* Begin transaction boundary */
EXEC SQL UPDATE ....;

/*This application server temporarily becomes a client and
requests *other service.*/

tpm_service(“AnotherService“);
tx_commit();              /* Commit the transaction */
<return service status back to the client>
}
```

**Example 2** Transaction started by an application client.

***Client:***

```
tx_begin();            /* Begin transaction boundary */
tpm_service(“Service1“);
tpm_service(“Service2”);
tx_commit();              /* Commit the transaction */
```

***Server:***

```
Service1()
{
<get service specific data>
EXEC SQL UPDATE ....;
<return service status back to the client>
}

Service2()
{
<get service specific data>
EXEC SQL UPDATE ....;
...
<return service status back to client>
}
```

## Migrating Precompiler or OCI Applications to TPM Applications

To migrate existing precompiler or OCI applications to a TPM application using the Oracle XA library, you must do the following:

1. Reorganize the application into a framework of “services.”

   This means that application clients request services from application servers.

   Some TPMs require the application to use the **tx_open** and **tx_close** functions, whereas other TPMs do the logon and logoff implicitly.

   If you do not specify the **sqlnet** parameter in your open string, the application will use the default SQL*Net driver. Thus, you need to be sure that the application server is brought up with the ORACLE_HOME and ORACLE_SID environment variables properly defined. This is accomplished in a TPM-specific fashion. See your TPM vendor documentation for instructions on how to accomplish this.

2. Ensure that the application replaces the regular connect and disconnect statements.

   For example, replace the connect statements EXEC SQL CONNECT (for precompilers) or **olon()** (for OCIs) by **tx_open(). R**eplace the disconnect statements EXEC SQL COMMIT / ROLLBACK RELEASE WORK (for precompilers), or **ologof()** (for OCIs) by **tx_close()**.

3. Ensure that the application replaces the regular commit / rollback statements and begins the transaction explicitly.

   For example, replace the commit / rollback statements EXEC SQL COMMIT / ROLLBACK WORK (for precompilers), or **ocom()/orol()** (for OCIs) by **tx_commit()/tx_rollback()** and start the transaction by calling **tx_begin()**.

4. Be sure to use **release_cursor=yes**, and to close all explicit cursors before committing a transaction.

The following table lists the TPM functions that replace regular Oracle commands when migrating precompiler or OCI applications to TPM applications.

| **Regular Oracle Commands** | **TPM Functions** |
|---|---|
| CONNECT *user/password* | **tx_open** (possibly implicit) |
| implicit start of transaction | **tx_begin** |
| SQL | service that executes the SQL |
| COMMIT | **tx_commit** |
| ROLLBACK | **tx_rollback** |
| disconnect | **tx_close** (possibly implicit) |
| SAVEPOINT *savepoint* | illegal |
| SET TRANSACTION READ ONLY | illegal |

**Table 7 – 1 TPM Replacement Commands**

## XA Library Thread Safety

If you use a transaction monitor that supports threads, the Oracle XA library allows you to write applications that are thread safe. Certain issues must be kept in mind, however.

A *thread of control* (or thread) refers to the set of connections to resource managers. In an unthreaded system, each process could be considered a thread of control, since each process has its own set of connections to resource managers and each process maintains its own independent resource manager table.

In a threaded system, each thread has an autonomous set of connections to resource managers and each thread maintains a *private* resource manager table. This private resource manager table must be allocated for each new thread and deallocated when the thread terminates, even if the termination is abnormal.

Note that, in an Oracle system, once a thread has been started and establishes a connection, only that thread can use that connection. No other thread can make a call to that connection.

### The Open String Specification

The **xa_open** string parameter, **xa_info**, provides the clause, Threads=, which must be specified as true to enable the use of threads by the transaction monitor. The default is false.

### OCI Clients

The open string parameter, Threads, must be specified as yes and the client must written to allocate new logon area (LDA). Note that, in most cases, threads will be created by the transaction monitor and that the application will not know when a new thread is created. Therefore, it is advisable to allocate an LDA on the stack within each service that is written for a transaction monitor application. Before doing any Oracle–related calls in that service, the sqlld2 function must be called and the LDA initialized. This LDA can then be used for all OCI calls within that service.

### Restrictions

The following restrictions apply when you are using threads:

- Any Pro* or OCI code that executes as part of the application server process on the transaction monitor cannot be threaded unless the transaction monitor is explicitly told when each new application thread is started. This is typically accomplished by using a special C compiler provided by the transaction monitor vendor.
- The Pro* statements, EXEC SQL ALLOCATE and EXEC SQL USE are not supported. Therefore when threading is enabled, embedded SQL statements cannot be used across non–XA connections.

## Troubleshooting

This section discusses how to find information in case of problems or system failure. It also discusses trace files and recovery of pending transactions.

### Trace Files

The Oracle XA library logs any error and tracing information to its trace file. This information is useful in supplementing the XA error codes. For example, it can indicate whether an **xa_open** failure is caused by an incorrect open string, failure to find the Oracle Server instance, or a login authorization failure.

The name of the trace file is:

**xa**_*db_namedate*.**trc**

where *db_name* is the database name you specified in the open string field **DB**=*db_name,* and *date* is the date when the information is logged to the trace file.

If you do not specify **DB=***db_name* in the open string, it automatically defaults to the name "NULL".

The trace file can be placed in one of the following locations:

- The trace file can be created in the **LogDir** directory as specified in the open string.
- If you do not specify **LogDir** in the open string, then the Oracle XA application attempts to create the trace file in the **$ORACLE_HOME/rdbms/log** directory, if it can determine where **$ORACLE_HOME** is located.
- If the Oracle XA application cannot determine where **$ORACLE_HOME** is located, then the trace file is created in the current working directory.

## Trace File Examples

Examples of two types of trace files are discussed below:

The example, xa_NULL040292.trc, shows a trace file that was created on April 2, 1992. Its **DB** field was not specified in the open string when the resource manager was opened.

The example, xa_Finance121591.trc, shows a trace file was created on December 15, 1991. Its **DB** field was specified as "Finance" in the open string when the resource manager was opened.

Note that multiple Oracle XA library resource managers with the same **DB** field and **LogDir** field in their open strings log all trace information that occurs on the same day to the same trace file.

Each entry in the trace file contains information that looks like this:

```
1032.12345.2:  xa_switch rtn ORA-22
```

where "1032" is the time when the information is logged, "12345" is the process ID (PID), "2" is the resource manager ID, **xa_switch** is the module name, and **ORA-22** is the Oracle Server information that was returned.

### In-doubt or Pending Transactions

In-doubt or pending transactions are transactions that have been prepared, but not yet committed to the database.

Generally, the transaction manager provided by the TPM system should resolve any failure and recovery of in-doubt or pending transactions. However, the DBA may have to override an in-doubt transaction in certain circumstances, such as when the in-doubt transaction is:

- locking data that is required by other transactions
- not resolved in a reasonable amount of time

For more information about overriding in-doubt transactions in the circumstances described above, or about how to decide whether the in-doubt transaction should be committed or rolled back, see your TPM documentation.

> **Additional Information:** For more information on pending transactions, see Chapter 5.

**Oracle Server SYS Account tables**

There are two tables under the Oracle Server SYS account that contain transactions generated by regular Oracle Server applications and Oracle XA applications. They are DBA_2PC_PENDING and DBA_2PC_NEIGHBORS

For transactions generated by Oracle XA applications, the following column information applies specifically to the DBA_2PC_NEIGHBORS table.

- the DBID column is always **xa_orcl**
- the DBUSER_OWNER column is always *db_name***xa.oracle.com**

Remember that the *db_name* is always specified as **DB=***db_name* in the open string. If you do not specify this field in the open string, then the value of this column is **NULLxa.oracle.com** for transactions generated by Oracle XA applications.

For example, you could use the SQL statement below to obtain more information about in-doubt transactions generated by Oracle XA applications.

```
SELECT * FROM DBA_2PC_PENDING p, DBA_2PC_NEIGHBORS n
    WHERE p.LOCAL_TRAN_ID = n.LOCAL_TRAN_ID
      AND
      n.DBID = ’xa_orcl’;
```

## Restrictions

### General Restrictions

General restrictions for the Oracle XA library are listed in this section.

#### Database Links

Oracle XA Applications can access other Oracle Server databases through database links, with the following restrictions:

- Use the Multi-Threaded Server configuration.

  This means the transaction processing monitors (TPMs) will use shared servers to open the connection to Oracle. The O/S network connection required for the database link will be opened by the dispatcher instead of the TPM server process. Thus, when a particular service or RPC completes, the transaction can be detached from the server so that it can be used by other services or RPCs.
- Access to the other database must use SQL*Net version 2.
- The other database being accessed should be another Oracle Server database.

Assuming that these restrictions are satisfied, Oracle Server will allow such links and will propagate the transaction protocol (prepare, rollback, and commit) to the other Oracle Server databases.

**Warning:** If these restrictions are not satisfied, when you use database links within an XA transaction, it creates an O/S network connection in the Oracle Server that is linked to the TPM server process. Since this O/S network connection cannot be moved from one process to another, you cannot detach from this server, and when you complete a service or RPC, you will receive an ORA#23 message.

If using the Multi-Threaded Server configuration is not possible then, access the remote database through the Pro*C/C++ application using EXEC SQL AT syntax.

#### Oracle Parallel Server Option

On some platforms, you cannot use the Oracle XA library together with the Oracle Parallel Server option. To run the Oracle XA library with the Parallel Server option, the platform's implementation of the distributed lock manager must support transaction-based rather than process-based deadlock detection.

OCDoc

**Additional Information:** For more information about the Parallel Server and the Oracle XA library, see your Oracle operating system–specific documentation.

## SQL-based Restrictions

SQL-based restrictions are listed in this section.

Rollbacks and Commits

Because the transaction manager is responsible for coordinating and monitoring the progress of the transaction, the application should not contain any Oracle Server-specific statement that independently rolls back or commits a transaction.

Do not use EXEC SQL ROLLBACK WORK for precompiler applications. Similarly, an OCI application should not execute **orol()**. You can roll back a transaction by the initiator by calling **tx_rollback().**

Similarly, a precompiler application should not have the EXEC SQL COMMIT WORK statement. An OCI application should not execute **ocom()**. Instead, use **tx_commit()** or **tx_rollback()** to end a transaction.

DDL statements

Because a DDL SQL statement such as CREATE TABLE implies an implicit commit, the Oracle XA application cannot execute any DDL SQL statements.

Savepoint

Do not use savepoint. For example, do not use:

`EXEC SQL SAVEPOINT` *`savepointname`*`.`

SET TRANSACTION

Do not use the SET TRANSACTION READ ONLY | READ WRITE | USE ROLLBACK SEGMENT SQL statement.

Connecting or Disconnecting with EXECSQL

Do not use the EXEC SQL command to connect or disconnect. That is, do not use EXEC SQL COMMIT WORK RELEASE or EXEC SQL ROLLBACK WORK RELEASE.

APPENDIX

A

# Operating System Dependencies

This Guide occasionally refers to other Oracle manuals that contain detailed information for using Oracle on a specific operating system. These Oracle manuals are often called *installation or user's guides*, although the exact name may vary on different operating systems. Throughout this guide, references to these manuals are marked with the icon shown in the left margin.

OSDoc

This appendix lists all references made in this manual to operating–system–dependent behavior for the Oracle distributed systems and networking. Use this appendix as a guide when moving data between operating systems or when designing operating–system independent applications.

Operating–system specific topics are listed alphabetically, with page numbers of sections that discuss these topics.

## General

## Networking

## Database Administration

## Transaction Recovery Management

## XA Library

# Index

# Reader's Comment Form

**Oracle7™ Server Distributed Systems, Volume I: Distributed Data**
**Part No. A32543–1**

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065 U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

Thank you for helping us improve our documentation.