Oracle® Rdb

Oracle SQL/Services Server Release 7.1.5 Configuration Guide

Release 7.1 for OpenVMS Alpha

June 2001

Part No. A90405-01

This document contains configuration information specific to Oracle SQL/Services release 7.1.5 for OpenVMS Alpha. This release is shipping in conjunction with Oracle Rdb release 7.1 for OpenVMS Alpha.

Oracle Rdb Oracle SQL / Services Server Release 7.1.5 Configuration Guide, Release 7.1 for OpenVMS Alpha

Part No. A90405-01

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Rdb Oracle SQL/Services Server Release 7.1.5 Configuration Guide, Release 7.1 for OpenVMS Alpha

Part No. A90405-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825   Attn: Oracle Rdb
- Postal service:
  Oracle Corporation
  Oracle Rdb Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle Rdb is a general-purpose database management system based on the relational data model.

Oracle SQL / Services, a client / server component of Oracle Rdb, enables a client application program, invoked on a remote client computer running on a supported operating system or transport, to access Oracle Rdb databases and other databases supported by SQL on an OpenVMS server system. See the *Guide to Using the Oracle SQL/Services Client API* for a complete list of supported clients.

This manual describes how to maintain and tune an Oracle SQL / Services server system.

Intended Audience

This manual is written for the system manager responsible for maintaining and fine-tuning Oracle SQL / Services. System managers should refer to the installation guide, which provides information important to the installation of an Oracle SQL / Services system.

Operating System Information

You can find information about the versions of the operating system and optional software that are compatible with this release of Oracle Rdb and Oracle SQL / Services in the installation guides and release notes for Oracle Rdb7 and Oracle SQL / Services.

Contact your Oracle representative if you have other questions about compatibility.

Structure

This manual contains the following chapters:

Chapter 1	Introduces the Oracle SQL / Services system.
Chapter 2	Describes how to manage an Oracle SQL / Services system.
Chapter 3	Describes how to maintain an Oracle SQL / Services server.
Chapter 4	Describes the Oracle SQL / Services system management commands.

Related Documentation

For more information, see the other manuals in this documentation set, especially the following:

- *New and Changed Features for Oracle Rdb, Release 7.1*
- *Oracle Rdb7 Guide to SQL Programming*
- *Oracle Rdb7 SQL Reference Manual*
- *Guide to Using the Oracle SQL/Services Client API*
- The release notes and installation documents for Oracle Rdb Release 7.1 and Oracle SQL / Services Release 7.1.5
- *Oracle Rdb7 Guide to Database Maintenance*

The Oracle Rdb Oracle SQL / Services Release 7.1.5 Release Notes for Release 7.1.5 and the Oracle Rdb Release Notes, Release 7.1.0 for OpenVMS Alpha for Release 7.1 are provided only as part of the software kit as PostScript and .txt files in the following directory location by platform unless otherwise specified:

Document	OpenVMS
Oracle Rdb Oracle SQL/Services Release 7.1.5 Release Notes	SYS$HELP
Oracle Rdb Release Notes, Release 7.1.0 for OpenVMS Alpha	SYS$HELP

Conventions

In this manual, Oracle Rdb refers to Oracle Rdb for OpenVMS Alpha software. Release 7.1 of Oracle Rdb software is often referred to as V7.1 or Rdb7.

The SQL interface to Oracle Rdb is referred to as SQL. This interface is the Oracle Rdb implementation of the SQL standard adopted in 1999, in general referred to as the ANSI/ISO SQL standard or SQL:1999. See the *Oracle Rdb Release Notes, Release 7.1.0 for OpenVMS Alpha* for addition information about this SQL standard.

Beginning with release 7.1, Oracle SQL / Services is a multiversion-only kit. The installation installs files using a variant naming convention. That is, variant file names and names of utilities may have a two-digit version number appended as the last two characters of its name. For example, the management client is SQLSRV_MANAGE71 and its log files are *71.log, and so forth, if Oracle SQL / Services release 7.1.5 is installed as a multiversion kit on the same node as a previous version of Oracle SQL / Services.

OpenVMS means the OpenVMS Alpha operating systems.

The following conventions are also used in this manual:

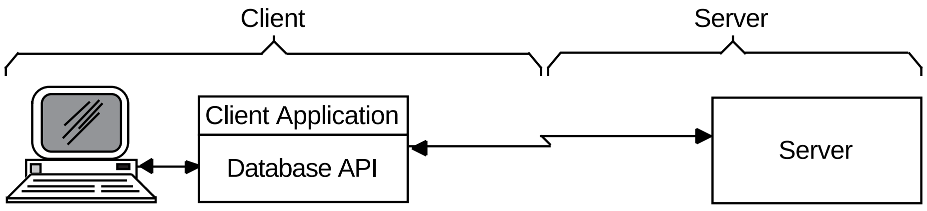
Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
$	The dollar sign represents the DIGITAL Command Language prompt in OpenVMS.
boldface text	Boldface type in text indicates a term defined in the text.
`monospaced boldface text`	Monospaced boldface type in text indicates user input.

See Section 4.1 for more information on syntax conventions used by the SQLSRV_MANAGE utility.

1 Introduction to an Oracle SQL/Services System

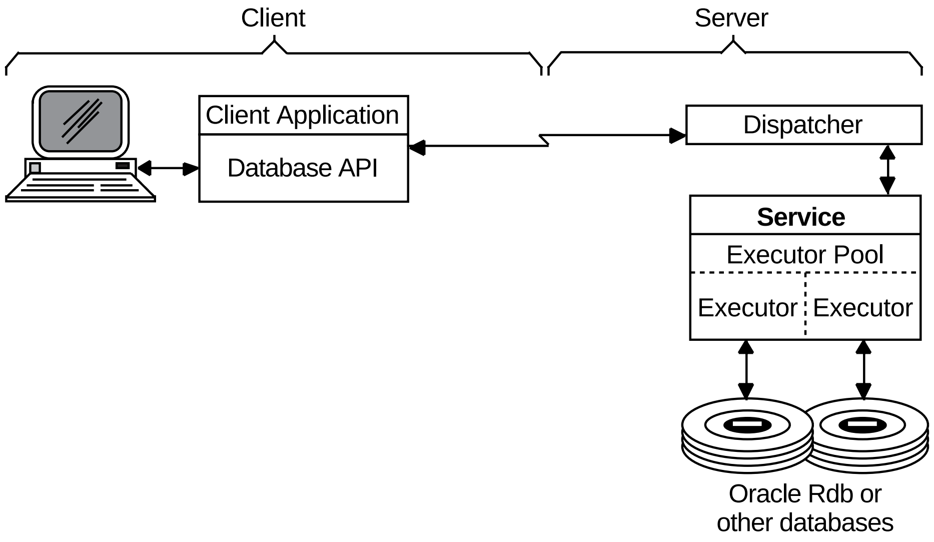
A client / server system in its simplest form consists of a client, a network, and a server system. A **client** is a software program that uses a database application programming interface (API) to make database requests of a server, as shown in Figure 1–1. The client may reside on the same platform as the server. Typically, however, the client application runs on a workstation or PC and accesses a database on a large server platform using a network that supports several transport protocols.

Figure 1–1 Simplest Client/Server Architecture



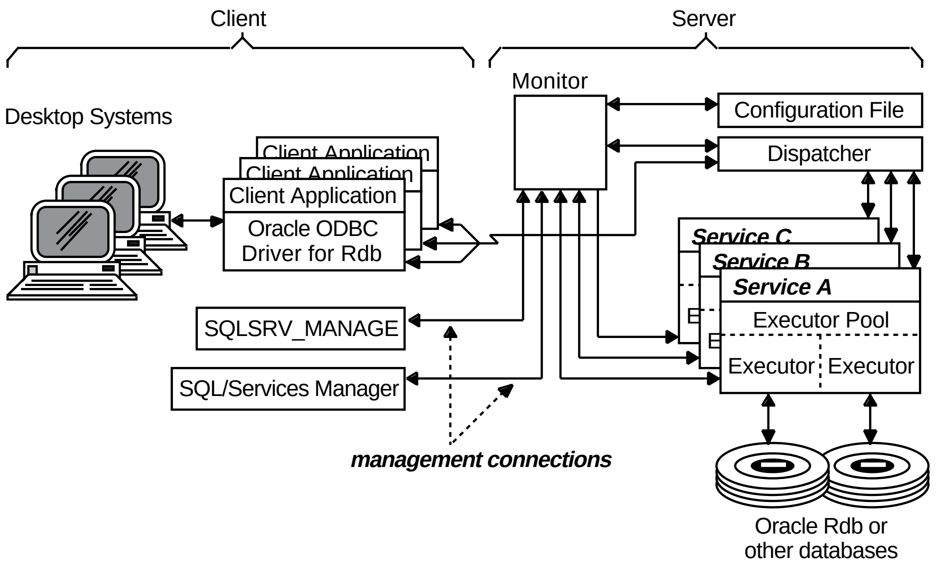
An Oracle SQL / Services **server** is a collection of cooperating processes on one node that includes a dispatcher process and a pool of executor processes that work on behalf of a service, as shown in Figure 1–2. The **dispatcher** process handles all network communication between the client and the server. It reads client requests, queues these requests for the executors of a service, and returns the responses from the executors of a service back to the client. An **executor** process works on behalf of a service and accepts the client requests from the dispatcher's queue, invokes the database engine to process the requests, and returns the results to the dispatcher. A **service** is a set of attributes that describes how clients access a database.

Figure 1–2 Oracle SQL/Services Server System



The Oracle SQL / Services server system also includes a monitor process to manage and control the server, an SQLSRV_MANAGE utility that runs on an OpenVMS local or remote system, or an Oracle SQL / Services Manager GUI server management utility that runs as a Windows client, and a configuration file in which to store server component definitions, as shown in Figure 1–3.

Figure 1–3 Oracle SQL/Services Client/Server Architecture



An Oracle SQL/Services **client** is a software program that accesses data by selecting a service provided by a server using an agreed upon interface such as the Oracle SQL/Services API, the Microsoft ODBC API, or the Oracle OCI interface. The server responds by receiving and processing client requests and sending the results back to the client.

A **network** is made up of communications hardware and software through which the client APIs communicate with the Oracle SQL/Services server. Request messages from the client and response messages from the server travel over a DECnet, Transmission Control Protocol/Internet Protocol (TCP/IP), NetWare (IPX/SPX), AppleTalk-DECnet gateway, or SQL*Net communications link.

Server

An Oracle SQL/Services **server** describes the attributes of a collection of cooperating processes on one node that provides one or more services. The server in general includes all server component attribute definitions, which are contained in a configuration file. See Service and Dispatcher topics, included in this list, for more information about these server components. For the server object specifically, the attributes include information such as the version of the server, the configuration file specification, the size of shared memory, and what network transports are supported for server management.

There can be only one server defined per configuration file. You can have only one server per version of Oracle SQL / Services started on a node at any given time.

Service

An Oracle SQL / Services **service** is a set of attributes that describes how clients access a database. Oracle SQL / Services accommodates the needs of different clients by supporting a range of service attributes that you tailor for each service provided by a server. The definition of a service includes information such as who can use the service, the database that is accessed by the service, the database engine version used by the service, how many clients can simultaneously use the service, and the number of executors that will be working on behalf of the service.

Executor

An Oracle SQL / Services **executor** is the process that works on behalf of a service.

An executor accepts client requests from dispatchers, calls SQL to process the requests, and returns the results to dispatchers. There is a pool of executor processes for each service that is started.

Dispatcher

An Oracle SQL / Services **dispatcher** is a process that is responsible for handling network communications for the clients and for the routing and scheduling of client requests to executors of a service. A dispatcher supports all services defined for a server. A single dispatcher typically supports more than one network protocol, but can support a single protocol. All clients except system management clients connect directly to a dispatcher.

Monitor

An Oracle SQL / Services **monitor** process provides overall management and control for the server, such as server startup and shutdown, reading and writing the configuration database, monitoring functions, and other management operations.

SQLSRV_MANAGE Client

The Oracle SQL / Services server management command-line interface lets you manage an Oracle SQL / Services server from an OpenVMS, Digital UNIX, or Windows NT system.

Oracle SQL/Services Manager Client GUI

The Oracle SQL / Services server management graphical user interface (GUI) utility lets you manage an Oracle SQL / Services server from any Windows client. The Oracle SQL / Services Manager GUI runs on Windows 95, Windows 98, Windows 2000, and Windows NT X86.

Configuration file

A **configuration file** contains all defined attributes for one server and its components, which include all service definitions, dispatcher definitions, and the list of users that are authorized to access the services provided by that server. This is also known as an **Oracle SQL/Services server configuration,** in that it represents one set of component definitions that are managed together for a server. Only one server can be defined in a configuration file. Typically, each server node has its own configuration file; however, it is possible to share a configuration file among multiple nodes in an OpenVMS cluster.

1.1 Server Management Utilities

You can manage an Oracle SQL / Services server in the following ways:

- Use the SQLSRV_MANAGE utility

  You can use the SQLSRV_MANAGE utility from a local or remote node on an OpenVMS system and manage the server online or offline (you must be on a local node to manage the server offline).

  Usually, you use the SQLSRV_MANAGE utility to manage a server configuration online by establishing a system management connection to a running server, then performing system management functions that operate on the running server as well as on the configuration file. In addition, you can use the SQLSRV_MANAGE utility to manage a server configuration offline by directly manipulating server component attributes in a configuration file. The only system management functions that you must perform offline are creating a new server configuration and starting a server. The SQLSRV_MANAGE utility accepts commands from the standard input device or from script files, and can be run interactively or in a batch job on an OpenVMS system.

- Use the Oracle SQL / Services Manager graphical user interface (GUI)

  You can use the Oracle SQL / Services Manager graphical user interface (GUI) from a remote Windows client and manage the server online using one of the following clients: Windows 95, Windows 98, Windows 2000, or Windows NT X86.

You use the Oracle SQL/Services Manager GUI to manage a server configuration online by establishing a system management connection to a running server, then performing system management functions that operate on the running server or the configuration file or both.

You cannot use the Oracle SQL/Services Manager GUI to perform the offline server management functions of creating a new server configuration, starting a server, or shutting down a running server; nor can you execute script files. However, you can modify and restart a running server using the Oracle SQL/Services Manager GUI.

Usually, you will use either of these two utilities interactively to manage the server and its components online. For most server management tasks, use the Oracle SQL/Services Manager GUI because it is easy to learn and use. The only time you may need to use the SQLSRV_MANAGE utility is to start a server or to run any special scripts you might create.

See Section 2.3 for more information about managing a server using the SQLSRV_MANAGE utility. For more information about managing a server using the Oracle SQL/Services Manager GUI, invoke the utility on a Windows client and select the Help pull-down menu.

1.1.1 Privileges Needed to Manage a Server

To start a server using the SQLSRV_MANAGE utility on OpenVMS, you must use an account that has been granted the SETPRV privilege or that has been granted all privileges. To make offline modifications to a server using the SQLSRV_MANAGE utility on OpenVMS, you must use an account that has been granted the NETMBX, SYSLCK, and SYSPRV privileges. To make online modifications to a server using the SQLSRV_MANAGE utility or the Oracle SQL/Services Manager GUI, you must use an account that has been granted use of the SQLSRV_MANAGE system management service for that server; however, you are not required to use an account that has been granted elevated privileges.

These privilege requirements are either less restrictive or identical to those needed to install Oracle SQL/Services on the OpenVMS platform. For more information, see the installation documentation for Oracle Rdb and Oracle SQL/Services.

1.1.2 Running the SQLSRV_MANAGE Utility

To run the SQLSRV_MANAGE utility, you first define a symbol to invoke the utility as follows:

```
sqlsrv_manage71 :== $SYS$SYSTEM:sqlsrv_manage71
```

You then enter the command sqlsrv_manage71 to invoke the SQLSRV_MANAGE utility. To use the SQLSRV_MANAGE utility interactively, invoke the utility, then enter system management commands in response to the SQLSRV> command-line prompt. To manage a server online, the first command you use is usually the CONNECT TO SERVER command. To manage a server offline, you first use a SET CONFIGURATION_FILE command to specify the name of the server configuration file, if the file is not stored in the default location (see the SET CONFIGURATION_FILE Command for more information).

You can also use scripts with the SQLSRV_MANAGE utility. A SQLSRV_MANAGE script is a file containing the same commands that you would enter at the SQLSRV> prompt. You can invoke a SQLSRV_MANAGE script interactively at the SQLSRV> prompt using the @ command. Alternatively, you can invoke the SQLSRV_MANAGE utility to read system management commands directly from a script. See the –input Switch in Chapter 4 for more information.

Scripts are a practical tool for making changes to a server on a regular basis. For example, suppose you want to increase the minimum and maximum number of executors for a service to meet a peak load condition. You can use one script to increase the values and another to decrease the values. You can automate the execution of the scripts using batch jobs.

1.1.3 Running the Oracle SQL/Services Manager GUI

To run the Oracle SQL / Services Manager GUI, click the Oracle SQL / Services Manager GUI icon to bring up the Connect To Server window and enter the node name, user name, and password. Select the transport and the TCP / IP port ID, DECnet object name that you want to use to establish the connection. Once connected, you can begin to manage the server.

1.2 Online Versus Offline Server Management

You can manage a server either online or offline using the SQLSRV_MANAGE utility.

Online Server Management

Typically, you manage the server online. To manage a server online, you always connect to the server using the CONNECT TO SERVER command. Once connected, any changes you make to the server are written to the configuration file. If you alter a dynamic attribute, the change is also made to the running server. See Section 2.3.1, Section 2.3.2, and Section 2.3.3 for a list of dynamic attributes. If you alter a nondynamic attribute of an object that is started, the system management utility displays a message that the object must be restarted for the change to take effect. The only time you need to restart the server is if the change to the server is to a nondynamic attribute of the server object itself, in which case changes take effect upon a server restart operation.

Offline Server Management

On occasion, you may need to manage a server offline to recover from an alteration that rendered the server unusable, such as setting too low a value for shared memory. To manage a server offline, you must use the SQLSRV_MANAGE utility. You cannot manage a server offline using the Oracle SQL / Services Manager GUI. If the configuration file is not stored in the default location (see the SET CONFIGURATION_FILE Command for more information), you must first select the configuration file by using the SET CONFIGURATION_FILE command before issuing any system management commands. Usually, you will manage a server offline only when the server is not running. However, you can manage a server offline even if the server is running. Any changes you make to the server configuration are written to the configuration file but *do not* affect the running server until the objects that have been changed are restarted. You must restart the entire server for a change to an attribute of the server object itself to take effect. You need only shut down and start the particular dispatcher or service for a change to an attribute of a dispatcher or service object to take effect. The only exception is that if you grant or revoke use of a service to or from a user name or identifier, then the change takes effect immediately.

Table 1–1 summarizes which Oracle SQL / Services server management commands can be performed online, offline, or both and any restrictions that may apply.

Table 1–1 Oracle SQL/Services Server Management Online and Offline Commands

Command	Online	Offline	Comments
ALTER DISPATCHER	X	X	Offline changes do not affect a running dispatcher.
ALTER SERVER	X	X	Offline changes do not affect a running server.
ALTER SERVICE	X	X	Offline changes do not affect a running service.
CONNECT TO SERVER	X	–	For online server management only.
CREATE DISPATCHER	X	X	Can create a dispatcher either online or offline.

Table 1–1 (Cont.) Oracle SQL/Services Server Management Online and Offline

Command	Online	Offline	Comments
CREATE SERVER	–	X	Can only create a server offline.
CREATE SERVICE	X	X	Can create a service either online or offline.
DISCONNECT SERVER	X	–	For online server management only.
DROP DISPATCHER	X	X	Can delete a dispatcher either online or offline.
DROP SERVER	–	X	Can only delete a server offline.
DROP SERVICE	X	X	Can delete a service either online or offline.
GRANT USE ON SERVICE	X	X	Offline changes affect running server.
KILL EXECUTOR	X	–	Can only kill an executor online.
RESTART SERVER	X	–	Can only restart a server online.
REVOKE USE ON SERVICE	X	X	Offline changes affect running server.
SET CONFIGURATION_ FILE	–	X	For offline server management only.
SET CONNECTION	X	–	For online server management only.
SHOW DISPATCHER	X	X	Can display definitional attributes of a dispatcher online or offline; can only show the run-time attributes of a dispatcher (such as its state) online.
SHOW SERVER	X	X	Can display definitional attributes of a server online or offline; can only show the run-time attributes of a server (such as its state) online.
SHOW SERVICE	X	X	Can display definitional attributes of a service object online or offline; can only show the run-time attributes of an object (such as its state) online.
SHUTDOWN DISPATCHER	X	–	Can only shut down a dispatcher online.
SHUTDOWN SERVER	X	–	Can only shut down a server online.
SHUTDOWN SERVICE	X	–	Can only shut down a service online.

Table 1–1 (Cont.) Oracle SQL/Services Server Management Online and Offline

Command	Online	Offline	Comments
START DISPATCHER	X	–	Can only start a dispatcher online.
START SERVER	–	X	Can only start a server offline.
START SERVICE	X	–	Can only start a service online.

Chapter 2 and Chapter 3 describe managing and maintaining the server. Chapter 4 contains reference material that describes SQLSRV_MANAGE commands. These chapters are written primarily for the Oracle SQL / Services system administrator who is using the SQLSRV_MANAGE utility and its command-line interface. As appropriate, only the command-line interface of SQLSRV_MANAGE commands and syntax are used in text and examples in these chapters.

An Oracle SQL / Services system administrator who is using the Oracle SQL / Services Manager GUI client from a Windows-based PC system can refer to Chapter 2 and Chapter 3, but should be aware that the names of commands and syntax vary between the SQLSRV_MANAGE command-line interface and the Oracle SQL / Services Manager GUI, and no attempt is made to describe these differences, as most differences are relatively minor. Refer to the Oracle SQL / Services Manager GUI client Windows help for more specific information.

2

Managing an Oracle SQL/Services System

Managing an Oracle SQL/Services system requires knowledge of the client and network components, together with dispatchers, services, and a server, as described in Chapter 1. You should have a general understanding of how each component works with other components in the client/server architecture and how the components within the server system operate. This chapter describes how to create and manage the server components.

2.1 Getting Started

After you install and start the default Oracle SQL/Services server, you may want to perform some additional tasks to ensure its optimum performance and to troubleshoot problems. These tasks include:

- Planning an Oracle SQL/Services server configuration
- Managing server components
- Setting shared memory size
- Setting up dispatchers and transport selection
- Setting up services and types of reuse
- Setting up security on servers
- Deciding which types of service to provide to clients
- Considering security for selecting the service owner user name
- Setting MIN_EXECUTORS, MAX_EXECUTORS, and IDLE_EXECUTOR_ TIMEOUT attributes
- Using an SQL initialization file

- Understanding how Oracle SQL/Services implements the database access authorization models

Each topic is discussed in the sections that follow.

2.2 Planning an Oracle SQL/Services Server Configuration

Your initial working Oracle SQL/Services server is defined by a configuration file. That file contains object definitions and characteristics for the server, dispatchers, services, and a set of authorized users for each service. You can display the current definition of each object with a SHOW command, read through the attribute settings, and from this basic understanding, take the following steps to plan your server configuration:

1. Determine your own requirements for your server system.
2. Learn about each object and how best to manage it.
3. Apply what you learned toward meeting your server system requirements.

Determining Server System Requirements

As the Oracle SQL/Services system administrator, you must determine the requirements for your server system. You should investigate the following:

- Is Oracle SQL/Services installed on a single node or in a cluster? Do different nodes require different dispatchers and services?
- What do you know about your user community? How many clients are there in total? How many clients will use the system at peak periods?
- What transports are available for client/server communication? How many ports are available for each transport?
- What version of Oracle Rdb do you have installed?
- What are the specific applications users want to run? Are users attaching to the same database or many different databases? What kinds of transactions will be run?

These are the most important questions to answer. Other questions may arise as you find answers to these questions that will help you to understand your own server requirements. As you seek answers to these questions, you should also begin to devise a plan for how to best meet the server needs of your user community and how to tune your server system to achieve maximum performance.

Learning About Server Objects

To start, ask the following questions about each server object:

- Which attributes do I need to monitor?
- Which attributes should I be most concerned about managing?

To answer these questions, it is important to understand the meaning of the default value of each attribute and then determine which attributes need to be monitored and adjusted. In general, all default settings of attributes for the default server system are sufficient to get started. Table 2–1, Table 2–2, and Table 2–3 provide a summary of the default values for the server, dispatcher, and service objects. Following each table is a brief description of which attributes to monitor and adjust.

Achieving Server System Requirements

By answering specific questions about the most important attributes for each server component, you can determine what modifications you may need to make to your server system. As you implement your plan, you learn how to create and alter server component objects and apply these changes toward meeting your server system requirements.

As you learn how to monitor and tune each object, you can begin to optimize the performance of the server and tailor your Oracle SQL/Services server to make it ideal for your database client/server environment. For example, once you know what applications your users want to run, you can decide on the kinds of services to provide for these client applications.

The most important items that you should consider for establishing a running server are discussed in Section 2.3, Section 2.4, Section 2.5, Section 2.6, Section 2.7, and Section 2.8.

After you tailor an Oracle SQL/Services server to meet your client/server requirements, the next task is to understand more about maintaining the server (see Chapter 3 for more information).

2.3 Managing Server Components

Managing server components consists of managing the server, dispatchers, and services and performing tasks such as creating these objects, starting, shutting down, and restarting these objects, altering object attributes, and deleting these objects. Section 2.3.1 through Section 2.3.3 describe managing each of these objects.

2.3.1 Managing a Server

Managing a server involves knowing how to create a server; how to start, stop, and restart a server; and how to tailor the attributes of a server to suit the specific requirements of your client / server configuration.

Creating a Server

When you install Oracle SQL / Services, the installation procedure automatically creates and starts a server on that node. Unless you encounter a nonrecoverable error condition that renders the configuration file unusable, you normally will not have to create or re-create a server on a node on which you performed the Oracle SQL / Services installation. However, you should periodically save a backup copy of your configuration file. See Copying a Configuration File in this section for details of how to make a copy of a configuration file.

If your configuration file becomes corrupted, due perhaps to a disk failure, and you do not have a backup copy, then you can delete the corrupted file and re-create your initial server configuration using the SQLSRV_CREATE71.COM command procedure. You can also use this procedure to re-create the Oracle RMU dispatcher and Oracle RMU service if you delete them and subsequently need to re-create them.

In an OpenVMS cluster environment, the installation procedure creates and starts a server only for the node on which you perform the installation. If you plan to use Oracle SQL / Services on other nodes in the cluster, you must create and start a server on each of those nodes or make a single configuration file available to the other nodes, then start the server on those nodes.

There are two ways to create and start a server on other nodes in an OpenVMS cluster:

- Use the SQLSRV_CREATE71.COM procedure provided by the installation

  The preferred method to create and start a server on another node in a cluster is to invoke the SQLSRV_CREATE71.COM DCL command procedure provided by the Oracle SQL / Services installation procedure in the SYS$MANAGER directory (see the *Oracle SQL/Services Installation Guide* for more information). This procedure is used by the installation procedure itself and so will create and start a server that is identical to the one created on the node where the original installation was performed.

- Copy a configuration file from another node in the cluster

  Another way to create a server on another node in a cluster is to copy a configuration file to that node, make any necessary changes for the node, then start the server on that node. This approach is more difficult because it can be error-prone, but nevertheless is an option. See Copying a Configuration File in this section for more information.

Alternatively, you may choose to share a single configuration file among multiple nodes in a cluster. The simplest way to make a single configuration file available to all nodes in a cluster is to shut down the server on the node on which you performed the installation, then rename the SQLSRV_CONFIG_FILE71.DAT file from the SYS$SPECIFIC:[SYSMGR] directory to the SYS$COMMON:[SYSMGR] directory. If you choose to share a single configuration file among multiple nodes in a cluster, you must take care not to delete an object on one node if you intend to continue to use it on other nodes.

You do not need to perform additional tasks if you want to provide exactly the same dispatchers and services on each node in the cluster. However, if you need to support different network protocols or provide specific services on different nodes in the cluster, then you must tailor your configuration accordingly. To provide different dispatchers or services on different nodes, you must set the AUTOSTART attribute to OFF for any services and dispatchers that should not be started on all nodes, then write a SQLSRV_MANAGE script for each node that starts only the required dispatchers and services for that node. Note that you cannot configure a service or dispatcher object in a shared configuration file to have different attributes for different nodes.

Caution: Oracle Corporation recommends that you do not make offline modifications to a configuration file if there is a server running that is using the same file. In this situation, the SQLSRV_MANAGE utility does not prevent you from deleting a dispatcher or service object offline while the dispatcher or service is running.

Similarly, neither the SQLSRV_MANAGE utility nor the Oracle SQL/Services Manager utility prevents you from deleting a dispatcher or service object online while the dispatcher or service is running on a different node in an environment where two or more nodes share the same configuration file. If this happens, then the SQLSRV_MANAGE utility displays a warning message if you show a dispatcher or service that has been deleted but that is still running.

The Oracle SQL/Services Manager utility displays the delete icon (no-entry sign) for a dispatcher or service that has been deleted but is still running.

Starting, Shutting Down, and Restarting a Server

The server is automatically started by the SQLSRV_CREATE71.COM DCL command procedure when a system boots. The server is automatically shut down when a system shuts down. Generally, the only time you will need to restart a server is if you alter a nondynamic attribute of the server object, in which case you must restart the server for the change to take effect.

Altering a Server

Once you create a server, you may need to alter some server attributes, such as the maximum amount of shared memory available to the server. Table 2–1 lists all of the attributes of a server, their default values, and indicates if an attribute can be modified dynamically. Following the table is a brief description of the major server attributes.

Table 2–1 Default Settings for Server Object Attributes

Attribute	Default Setting	Dynamic Attribute
MAX_SHARED_MEMORY_SIZE	2000 kilobytes	
Configuration File	SYS$MANAGER: SQLSRV_CONFIG_FILE71.DAT	
PROCESS_STARTUP_TIMEOUT	0	Yes
PROCESS_SHUTDOWN_TIMEOUT	0	Yes
Network ports	DECnet - SQLSRV_SERVER	
Network ports	TCP/IP - 2199	

Oracle SQL/Services uses shared memory for interprocess communications. The MAX_SHARED_MEMORY_SIZE attribute is the only server attribute you need to monitor on a periodic basis using the SHOW SERVER Command. Section 2.4 describes what to look for and when to make adjustments.

The server uses network ports to listen to system management clients. These network ports must be unique in a multiversion environment because you can only have one version of Oracle SQL/Services using the default network ports. During a multiversion installation, you must specify what alternate network ports you want the server to use. You need not make any further changes to these network ports unless you decide to make the current version of Oracle SQL/Services the default, and you want to use the default system management network ports. If system management clients are having problems connecting, use the SHOW SERVER command to monitor these network ports and to ensure each is running.

See the ALTER SERVER Command for more information about altering server attributes.

If you alter a dynamic attribute of a running server online, then the change takes effect immediately. However, if you alter a nondynamic attribute of a running server online, then you must restart the server for the change to take effect.

If you alter a nondynamic attribute of a running server using the SQLSRV_MANAGE utility, it displays an alternate success status indicating that you must restart the server for the change to take effect. For example:

```
SQLSRV> ALTER SERVER MAXIMUM_SHARED_MEMORY_SIZE 4000;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
SQLSRV>
```

The SQLSRV_MANAGE utility displays the values of any altered nondynamic attributes that will take effect when the server is restarted. For example:

```
** The Server will be updated as follows when it is restarted **
    Max Shared memory size:  4000
```

If you alter a nondynamic attribute of a running server using the Oracle SQL/Services Manager utility, it displays an alternate success status indicating that you must restart the server for the change to take effect and inquires if you want to restart the server immediately. The Oracle SQL/Services Manager utility highlights the values of any altered nondynamic attributes that will take effect when the server is restarted by displaying a flag-shaped icon next to each attribute.

When you restart a server, all dispatchers and services of the server are also restarted, and all client network connections to the server are disconnected. Therefore, you should schedule alterations to the server object when few or no clients will be using the server.

Copying a Configuration File

However, you can use the COPY command to make a copy of a configuration file only if there is not a running server that is using the file. To make a copy of a configuration file currently being used by a running server, you must use the DCL BACKUP/IGNORE=INTERLOCK command.

Deleting a Server

The only time you need to delete a server is when the configuration file has become corrupt, due perhaps to a disk failure, and is completely unusable. Deleting a server is an offline operation and deletes the configuration file (see the DROP SERVER Command). Alternatively, you can use the DELETE command.

If, for some reason, you must delete a running server, you must first shut it down online (see the SHUTDOWN SERVER Command) and then delete it offline using the DROP SERVER command.

2.3.2 Managing a Dispatcher

Managing a dispatcher involves knowing how to create a dispatcher; how to start, stop, and restart a dispatcher; and how to tailor the attributes of a dispatcher to suit the specific requirements of your client/server configuration.

Creating a Dispatcher

The Oracle SQL/Services installation procedure and the SQLSRV_CREATE71.COM command procedure creates and starts three dispatchers named SQLSRV_DISP for use by Oracle SQL/Services clients, OCI_DISP for use by OCI clients, and RMU_DISP for use by Oracle RMU clients.

If you plan to use the SQL*Net network transport, then you will create another dispatcher after you decide which network ports you will use. You might also create other dispatchers if you decide to provide individual dispatcher processes for each transport available on your network. When you create a new dispatcher, you must ensure that the network ports that you specify are not used by any other dispatchers on the node. If a dispatcher is unable to listen on any of its network ports, it writes an error message to its log file and terminates.

Starting, Shutting Down, and Restarting a Dispatcher

Dispatchers that have the AUTOSTART attribute set to ON are automatically started when you install Oracle SQL/Services and whenever a server is started. If necessary, you can disable this action by starting a server with the START SERVER AUTOSTART OFF command. Dispatchers are automatically shut down when the server shuts down. One of the few times you must shut down a dispatcher is if it failed to start. A failed dispatcher is always left in a failed state and must be shut down. Once shut down, the reason for failure, which can be due to an incorrectly specified argument value in its definition, can be corrected using an ALTER DISPATCHER command, and then the dispatcher can be started using the START DISPATCHER command. Generally, the only time you will need to restart a dispatcher is if you alter a nondynamic attribute of a dispatcher object, in which case you have to restart the dispatcher for the change to take effect.

Altering a Dispatcher

As circumstances change, you may find it necessary to alter some dispatcher attributes. For example, to support additional users, you may need to increase the maximum number of connections allowed to a dispatcher. To provide better performance, you may want to increase the maximum client buffer size.

Because you run multiple versions of the Oracle SQL/Services server, you may want to alter the network port specifications to use the default network ports when you stop using the older version of Oracle SQL/Services.

Table 2–2 lists all of the attributes of a dispatcher and their default values, and indicates if an attribute can be modified dynamically. Following the table is a brief description of the major dispatcher attributes.

Table 2–2 Default Settings for Dispatcher Object Attributes

Attribute	Default Setting	Dynamic Attribute
AUTOSTART	ON	
MAX_CONNECTIONS	100	
IDLE_USER_TIMEOUT	0	Yes
MAX_CLIENT_BUFFER_SIZE	5000	
Log File	SYS$MANAGER:<dispatcher-name>.LOG	
Dump File	SYS$MANAGER:<dispatcher-name>.DMP	
Message Protocol	SQLSERVICES	
Network ports	DECnet - 81	
Network ports	TCP/IP - 118	
Network ports	IPX/SPX - 0x84b1	
Network ports	SQL*Net - no default, see listener.ora file for a list of listener objects you can use	

Set a higher value for the MAX_CONNECTIONS argument if you expect more than 100 clients to connect to the dispatcher at the same time.

Set a higher value for the MAX_CLIENT_BUFFER_SIZE argument if you know certain applications will benefit by using a larger buffer size.

The dispatcher uses network ports to listen to Oracle SQL/Services, Oracle ODBC Driver for Rdb, and Oracle OCI clients.

These network ports must be unique in a multiversion environment because you can have only one version of Oracle SQL/Services using the default network ports for a dispatcher on a node. During a multiversion installation, you must specify which alternate network ports you want the dispatcher to use. You need not make any further changes to these network ports unless you want to create one dispatcher listening exclusively on DECnet network ports and another dispatcher listening exclusively on TCP/IP network ports, and so forth, because of the network traffic. If clients are having problems connecting to dispatchers, use the SHOW DISPATCHER command to monitor these network ports and to ensure each is running.

See the ALTER DISPATCHER Command for more information about altering dispatcher attributes.

If you alter a dynamic attribute of a running dispatcher online, then the change takes effect immediately. However, if you alter a nondynamic attribute of a running dispatcher online, then you must restart the dispatcher for the change to take effect.

If you alter a nondynamic attribute of a running dispatcher using the SQLSRV_MANAGE utility, it displays an alternate success status indicating that you must restart the dispatcher for the change to take effect. For example:

```
SQLSRV> ALTER DISPATCHER sqlsrv_disp MAX_CLIENT_BUFFER_SIZE 10000;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
```

The SQLSRV_MANAGE utility displays the values of any altered nondynamic attributes that will take effect when the dispatcher is restarted. For example:

```
** This Dispatcher will be updated as follows when it is restarted **
        Max client buffer size:    10000 bytes
```

If you alter a nondynamic attribute of a running dispatcher using the Oracle SQL/Services Manager utility, it displays an alternate success status indicating that you must restart the dispatcher for the change to take effect and inquires if you want to restart the dispatcher immediately. The Oracle SQL/Services Manager utility highlights the values of any altered nondynamic attributes that will take effect when the dispatcher is restarted by displaying a flag-shaped icon next to each attribute.

When you restart a dispatcher, all client network connections to the dispatcher are disconnected. Therefore, you should schedule alterations to a dispatcher when few or no clients will be using the network ports managed by that dispatcher.

Deleting a Dispatcher

To delete a dispatcher as an online operation, you must first shut it down (see the SHUTDOWN DISPATCHER Command and the DROP Command). The only time you want to delete a dispatcher is if it is no longer needed.

2.3.3 Managing a Service

Managing a service involves knowing how to create a service; how to start, stop, and restart a service; and how to tailor the attributes of a service to suit the specific requirements of your client/server configuration.

Creating a Service

The Oracle SQL/Services installation procedure and the SQLSRV_CREATE71.COM command procedure creates and starts three services: a universal service named GENERIC for use by Oracle SQL/Services clients, a database service named OCI_ SAMPLE for use by Oracle OCI clients, and an Oracle RMU service named RMU_ SERVICE for use by Oracle RMU clients.

As the server administrator, you may need to create other universal services for different versions of Oracle Rdb. Similarly, you may want to create one or more database services for specific Oracle Rdb databases on your system. When you create a service, you must decide who will be authorized to access the service, how many executors will be needed to support clients who will use the service, and so forth. See Section 2.6 for more information about universal and database services.

Starting, Shutting Down, and Restarting a Service

Services that have the AUTOSTART attribute set to ON are automatically started when you install Oracle SQL/Services and whenever a server is started. If necessary, you can disable this action by starting a server with the START SERVER AUTOSTART OFF command. Services are automatically shut down when the system shuts down. Usually, you set the AUTOSTART attribute to ON for most services you create so that they are available to clients all of the time.

However, you may decide to start certain services manually. For example, you may create a transaction reusable service for a particular database to determine if you can achieve better performance than using a session reusable service. In this situation, you might choose to set the AUTOSTART attribute to OFF while you test the new service. One reason to shut down a service is if you must prevent clients from accessing the database provided by a service. For example, you would shut down a service while you restored a database after encountering a disk failure. Another reason you must shut down a service is if it failed to start. A failed service is always left in a failed state and must be shut down. Once shut down, the reason for failure, which can be due to an incorrectly specified argument value in its definition, can be corrected using an ALTER SERVICE command, and then the service can be started using the START SERVICE command. Generally, the only time you will need to restart a service is if you alter a nondynamic attribute of a service object, in which case you have to restart the service for the change to take effect.

Altering a Service

After you create a service, you may need to tune the performance of your system by adjusting the number of executors or the number of clients per executor for a service. If new users are added to the network, you may need to authorize access to a service to those users. If you upgrade a database to a higher version of Oracle Rdb, you will need to alter a service to specify a new SQL version to be used by the executors of the service. Table 2–3 lists all of the attributes of a service, their default values, and indicates if an attribute can be modified dynamically. Following the table is a brief description of the major service attributes.

Table 2–3 Default Settings for Service Object Attributes

Attribute	Default Setting	Dynamic Attribute
AUTOSTART	ON	
DEFAULT_CONNECT_USERNAME	None	Yes
REUSE_SCOPE	SESSION	
SQL_VERSION	STANDARD	
PROTOCOL	SQLSERVICES	
PROCESS_INITIALIZATION	None	
ATTACH	None	
OWNER	None	
SCHEMA	None	
SQL_INIT_FILE	None	
DATABASE_AUTHORIZATION	CONNECT_USERNAME	
APPLICATION_TRANSACTION_ USAGE	SERIAL	Yes
IDLE_USER_TIMEOUT	0	Yes
IDLE_EXECUTOR_TIMEOUT	1800	Yes
MIN_EXECUTORS	0	Yes

Table 2–3 (Cont.) Default Settings for Service Object Attributes

Attribute	Default Setting	Dynamic Attribute
MAX_EXECUTORS	1	Yes
CLIENTS_PER_EXECUTOR	1	Yes

Create as many service objects as you need to accommodate the databases accessed by applications that your user community intends to run. See Section 2.6, Section 2.7, Section 2.8, and Section 2.12 for more information.

Set the MIN_EXECUTORS, MAX_EXECUTORS, and IDLE_EXECUTOR_TIMEOUT attributes for each service based on user activity over time to provide efficient services to your clients. See Section 2.10 for more information.

You may need to adjust the CLIENTS_PER_EXECUTOR attribute value to attain the best performance when tuning a transaction reusable service.

Giving users or identifiers access to services or modifying their current access is another task you need to perform on a continual basis. Use the GRANT USE ON SERVICE and REVOKE USE ON SERVICE commands to perform these tasks. Use the SHOW SERVICE command to determine the users or identifiers who currently have access to a particular service.

See the ALTER SERVICE Command for more information about altering service attributes.

If you alter a dynamic attribute of a running service online, then the change takes effect immediately. However, if you alter a nondynamic attribute of a running service online, then you must restart the service for the change to take effect.

If you alter a nondynamic attribute of a running service using the SQLSRV_ MANAGE utility, it displays an alternate success status indicating that you must restart the service for the change to take effect. For example:

```
SQLSRV> ALTER SERVICE payroll SQL_INIT_FILE PAYROLL_DIR:PAYROLL.SQLINIT;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
```

The SQLSRV_MANAGE utility displays the values of any altered nondynamic attributes that will take effect when the dispatcher is restarted. For example:

```
** This Service will be updated as follows when it is restarted **
    SQL init file:            payroll_dir:payroll.sqlinit
```

If you alter a nondynamic attribute of a running service using the Oracle SQL/Services Manager utility, it displays an alternate success status indicating that you must restart the service for the change to take effect and inquires if you want to restart the service immediately. The Oracle SQL/Services Manager utility highlights the values of any altered nondynamic attributes that will take effect when the service is restarted by displaying a flag-shaped icon next to each attribute.

When you restart a service, all client network connections from applications using the service are disconnected. Therefore, you should schedule alterations to a service when few or no clients will be using the service.

Deleting a Service

The only time you may need to delete a universal service is when there are no more databases for that specific version of Oracle Rdb in use. Similarly, you may want to delete a database service if it is no longer used or if there are too few users using it to justify this type of service. In either case, to delete the service online, you must first shut it down (see the SHUTDOWN SERVICE Command and the DROP Command).

2.4 Setting Shared Memory Size

You can set the size of shared memory that the server uses by specifying a value for the MAX_SHARED_MEMORY_SIZE argument of the ALTER SERVER command. By default, the server uses 2000 kilobytes (2 megabytes) of shared memory.

Setting the MAX_SHARED_MEMORY_SIZE argument is important for optimizing the resource usage of the server system. The goal is to use the smallest amount of shared memory possible to provide the required services. This section explains how the Oracle SQL/Services server uses shared memory and how to set the MAX_SHARED_MEMORY_SIZE argument for the best resource usage.

You can change the value for shared memory using the ALTER SERVER command. However, this is not a dynamic attribute and requires that you restart the server. For example:

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> ALTER SERVER MAX_SHARED_MEMORY_SIZE 4000;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
SQLSRV> RESTART SERVER;
Disconnected from Server
```

The following are the two main costs associated with allocating shared memory:

- Disk space for the system page file

  You must ensure that you have sufficient system page file space available to accommodate an increase in the size of shared memory. You must also ensure that the GBLPAGFIL SYSGEN parameter is set high enough to accommodate an increase in the size of shared memory.

- Virtual memory for each component process of the server

  Mapping shared memory makes each component process of the server use more virtual memory, and thus consumes incremental operating system resources.

Oracle SQL/Services manages shared memory in units of 65,536 bytes. Therefore, the actual size of shared memory may be less than the MAX_SHARED_MEMORY_SIZE attribute because the size is rounded down to the nearest even 65,536-byte unit.

The server shared memory does not shrink or grow as the server runs. More or less of it may be in use at a given time. When you issue a SHOW SERVER command in the SQLSRV_MANAGE application for the server that you are connected to, SQLSRV_MANAGE will show three values:

- Total memory

  This number is static for a given run of the server. You can alter the MAX_SHARED_MEMORY_SIZE argument for a server, and it takes effect when you issue a RESTART SERVER command. Total memory is the number of shared memory units mapped by the server.

- Free memory

  This is the number of shared memory units that are completely unused by the Oracle SQL/Services server.

- Partly allocated memory

  A shared memory unit may not be completely used in one piece. It is often subdivided into smaller pieces. Shared memory units that are subdivided and partly used are reported as partly allocated memory. It is currently not practical to display the usage within the subdivided unit.

Free memory and partly allocated memory describe the shared memory units that can still be allocated. By subtracting those units from the total units, you can determine the shared memory units that are entirely used.

The minimum value for MAX_SHARED_MEMORY_SIZE is 132 KB. A minimum value of 132 KB for the size of shared memory provides two shared memory units. This is sufficient to start the monitor, connect to it from the SQLSRV_MANAGE application, and run one or two executors serving one or two clients.

The maximum value for MAX_SHARED_MEMORY_SIZE is 2,000,000 KB. Lower values should suffice for all applications.

In general, plan for the following shared memory usage:

- For each executor and dispatcher that you plan to run, allow about 3 KB.
- For each Oracle SQL/Services client connection that you plan to support, you need to take into account the base shared memory usage for a client and add to that the memory used for communication buffers.

  The base shared memory usage is about 11 KB.

  An Oracle SQL/Services application minimally consumes two communication buffers. The default buffer size is 1.3 KB, so the minimum size for an Oracle SQL/Services client is 15 KB (11 KB base + 4 KB for messages buffers).

  If you use a 5 KB message buffer size, the minimum size is about 21 KB (11 KB base + 2 * 5 KB for message buffers).

  However, not all Oracle SQL/Services applications use only two buffers. When a multi-tuple fetch or insert operation is initiated, you may get additional buffers for the client. How many additional buffers you get is based on the application. The dispatcher imposes a limit of 11 buffers that can be used at any one time.

A strategy for determining optimal shared memory size is as follows:

1. Pick a generous size for your shared memory based on the rough sizing method mentioned previously.
2. Run your system under normal load.
3. Occasionally issue a SHOW SERVER command from SQLSRV_MANAGE on the server that you are managing. It will show you the memory usage.
4. Adjust your shared memory size:
   - Downward, if you see a constant number of free memory units.
   - Upward, if you see no free memory units. You may also see client connections terminated by the server due to a lack of shared memory. This is reported in the log files. In certain rare situations, the entire server can fail due to insufficient shared memory.

5. As you add new users and applications to the server, review the shared memory usage.

2.5 Setting Up Dispatchers and Transport Selection

A client communicates with the server and dispatcher by using a network transport supported on your system. The Oracle SQL/Services release 7.1 and higher server supports the TCP/IP, DECnet, and NetWare (IPX/SPX) transports.

When you create a dispatcher object, you can specify whether you want the dispatcher to support one or more transports. If you want a dispatcher to support only one transport, you must create additional dispatchers to support each of the transports that your Oracle SQL/Services clients use. For Oracle SQL/Services, you can use one or more dispatchers for each server configuration. Each dispatcher defined must be listening on one or more unique network port IDs or objects.

The following example illustrates how to create a dispatcher that supports the SQL*Net transport:

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> CREATE DISPATCHER sqlnet_disp NETWORK_PORT SQLNET LISTENER_NAME ’LISTENER’;
SQLSRV> START DISPATCHER sqlnet_disp;
```

See the *Guide to Using the Oracle SQL/Services Client API* for more information on using the SQL*Net transport.

The following example illustrates how to shut down and delete a dispatcher that supports two transports, and create two other dispatchers, each supporting just a single transport. First, ensure that no clients are using any transports supported by the dispatcher that you plan to delete. Shutting down a dispatcher will disconnect the network connections from any clients that are using the dispatcher. If no clients are using the dispatcher, then shut down and delete the dispatcher. Finally, create and start the new dispatchers.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHOW CLIENTS;

Service: SQLSRV_MANAGE

  Connect         Client                              Executor
  Username        Node             State              PID
```

```
Application
  root           127.0.0.1       RUNNING BOUND    00000ec3
SQLSRV_MANAGE
SQLSRV> SHUTDOWN DISPATCHER sqlsrv_disp;
SQLSRV> DROP DISPATCHER sqlsrv_disp;
SQLSRV> CREATE DISPATCHER sqlsrv_tcpip NETWORK_PORT TCPIP;
SQLSRV> CREATE DISPATCHER sqlsrv_decnet NETWORK_PORT DECNET;
SQLSRV> START DISPATCHER sqlsrv_tcpip;
SQLSRV> START DISPATCHER sqlsrv_decnet;
```

2.6 Setting Up Services and Types of Reuse

The Oracle SQL / Services server provides universal services and database services.

Universal Service

A universal service allows a client application to determine which database is to be accessed. An executor process for a universal service, therefore, is not preattached to a specific database. Each time a client application connects to a universal service, it must issue one or more database attach statements before performing any data access operations.

You can use universal services with Oracle Rdb to provide access to local and remote Oracle Rdb databases.

Database Service

A database service allows a client application to access data within a specific database. Therefore, an executor process for a database service is preattached to a single database. When a client connects to a database service, it can immediately begin to access data in the preattached database.

You can use database services with Oracle Rdb to provide access to local and remote databases with the restriction that you must set the database authorization attribute to the service owner to access remote databases (see Section 2.8.2 for more information).

For Oracle Rdb V6.1 and higher for an executor process providing either type of database service, the following SQL statements cannot be prepared within this context:

- ATTACH
- DECLARE DATABASE
- CREATE DATABASE

- ALTER DATABASE
- DROP DATABASE
- CONNECT
- SET CONNECT
- DISCONNECT

Note: A client connected to a database service can access data only from the preattached database; it cannot access data from any other database.

Types of Reuse

The Oracle SQL/Services server provides services that have either a session reuse or transaction reuse attribute.

Session Reuse An executor for a session reusable service processes requests for one client session at a time. A session begins when a client connects to the service using either an sqlsrv_associate call or an ODBC connect function and the connection is bound to an executor process. A session ends when a client disconnects from the service and the connection is unbound from the executor process. A session reusable service is so named because an executor does not begin to process a new session until the current session ends. The session reuse attribute may be applied to either universal or database services. See Section 2.6.1 and Section 2.6.2 for more information.

Transaction Reuse An executor for a transaction reusable service processes requests for one transaction for a client at a time; however, the executor is shared by many concurrent client sessions. A transaction begins when a client issues an SQL statement that either implicitly or explicitly starts a transaction. A transaction ends when the client issues a successful SQL COMMIT or ROLLBACK statement or executes a stored procedure that commits or rolls back a transaction. A transaction reusable service is so named because an executor does not begin to process a new transaction until the current transaction ends. The transaction reuse attribute may be applied only to database services. See Section 2.6.3 for more information.

In summary, the Oracle SQL/Services server provides the following three fundamental types of service:

- Session reusable universal service

- Session reusable database service
- Transaction reusable database service

Table 2–4 summarizes the attributes and settings associated with each service.

Table 2–4 Oracle SQL/Services Service Attributes

	Service		
Service Definition Attribute	**Session Reusable Universal**	**Session Reusable Database**	**Transaction Reusable Database**
Prestarted	Yes	Yes	Yes
Preattached	No	Yes	Yes
Execute ATTACH statement by using the Oracle SQL/Services API or the Oracle ODBC Driver for Rdb	Yes	No	No
Execute multiple attachments	Yes	No	No
Number of clients per executor	1	1	>1

Section 2.6.1, Section 2.6.2, and Section 2.6.3 describe each type of service in more detail.

2.6.1 Session Reusable Universal Services

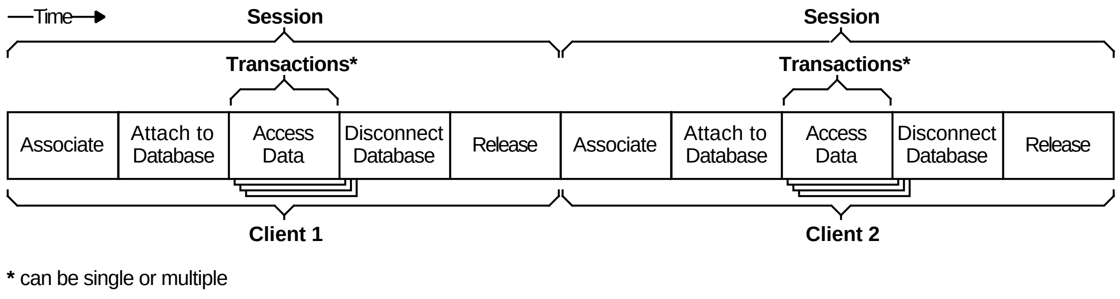
An executor for a session reusable universal service processes requests for a single client session at one time and is not preattached to a specific database.

You use a session reusable universal service when you want to allow client applications to determine which database to use or if you have a node with a large number of infrequently accessed databases for which it would be impractical to provide individual database services. See Section 2.6.4 and Section 2.8 for more information on deciding which types of service to provide.

Executor processes for session reusable universal services may be prestarted or started on demand. By prestarting a sufficient number of executor processes for a session reusable universal service, you enable clients to avoid the process startup delay when they connect to the service. Clients will always incur the overhead of attaching to the required database when using a session reusable universal service.

Figure 2–1 illustrates how a session reusable universal service works. When a client connects to a session reusable universal service, the client connection is assigned and bound to an executor process. Once bound, the client makes API calls to attach to one or more databases, access data, and finally disconnect from any attached databases. When the client releases the connection, the executor process unbinds from the client connection. The executor process is then available for use by another client.

Figure 2–1 Oracle SQL/Services Session Reusable Universal Services



2.6.2 Session Reusable Database Services

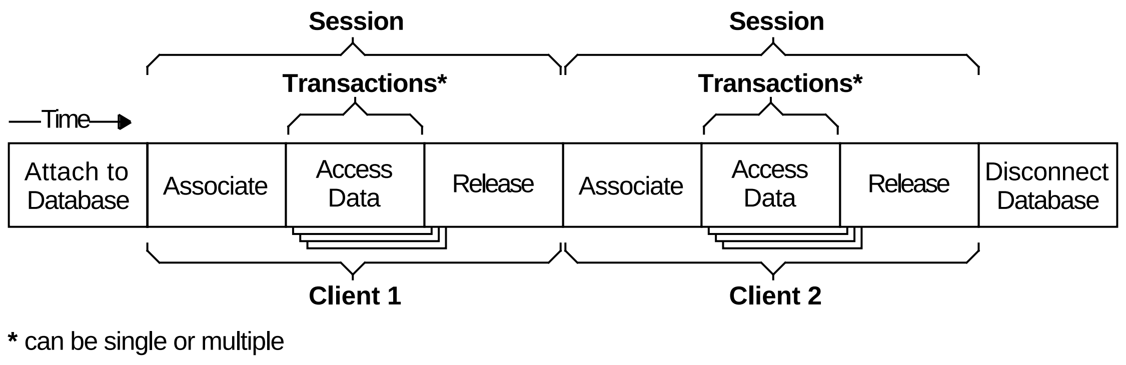
An executor for a session reusable database service processes requests for a single client session at one time and is preattached to a single database.

You use a session reusable database service when you want to provide clients with a service that accesses a specific database whose transactions are of long or unknown duration. See Section 2.6.4 and Section 2.8 for more information on deciding which types of service to provide.

Executor processes for session reusable database services may be prestarted or started on demand. By prestarting a sufficient number of executor processes for a session reusable database service, you enable clients to avoid process startup and database attach delays when they connect to the service.

Figure 2–2 illustrates how a session reusable database service works. When a client connects to a session reusable database service, the client connection is assigned and bound to an executor process. Once bound, because the executor is preattached to a database, the client can immediately make API calls to access data in the database. When the client releases the connection, the executor process unbinds from the client connection. The executor process is then available for use by another client.

Figure 2–2 Oracle SQL/Services Session Reusable Database Services



2.6.3 Transaction Reusable Database Services

An executor for a transaction reusable database service is preattached to a single database, processes requests for the transaction of one client at one time, and is shared by many concurrent client sessions. Once assigned to a particular executor process, a specific client connection remains assigned to that executor process until the client application disconnects from the service.

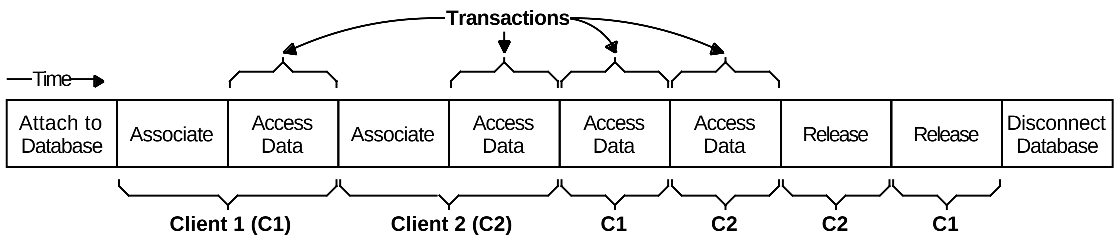
You use a transaction reusable database service to provide clients with a service that accesses a specific database where the database workload consists of transactions of known, relatively short duration. When used in the appropriate situations, transaction reusable database services can improve performance by reducing system resource usage and database contention. This is because multiple clients share a single executor process, thus reducing the total number of executor processes required on the system.

Transaction reusable database services are not so well suited to situations where transactions are of long or varying duration. If transaction reusable database services are employed in such a situation, users will tend to experience unpredictable response times because a client executing a long transaction will tie up an executor process, making it unavailable for other users. See Section 2.6.4 and Section 2.8 for more information on deciding which types of service to provide.

Executor processes for transaction reusable database services are always prestarted so that the server can distribute client connections evenly across the set of executor processes started for the service. Because multiple client connections share a single transaction reusable executor process, you need not prestart as many executor processes as when using session reusable executors. Fewer executor processes, with a high number of clients per executor, are required when the workload consists of very short transactions. More executor processes, with a lower number of clients per executor, are required as the transaction duration increases.

Figure 2–3 illustrates how a transaction reusable database service works. When a client connects to a transaction reusable database service, the client connection is assigned to an executor process; however, the client connection does not stay bound to the executor process after the executor has processed the initial connection. Multiple client connections may be assigned to a single executor process. A client connection is bound to an executor process when a transaction is started, at which time the client makes API calls to access data in the database. When the client program ends the transaction, by using an SQL COMMIT or ROLLBACK statement or by executing a stored procedure that commits or rolls back a transaction, the executor process unbinds from the client connection. The executor process then becomes available for use by another client connection. When the client releases the connection, the client connection is deassigned from the executor process.

Figure 2–3 Oracle SQL/Services Transaction Reusable Database Services



2.6.4 When to Use Session Reusable Versus Transaction Reusable Database Services

Table 2–5 summarizes the factors to consider in deciding whether to use session reusable database services or transaction reusable database services.

Table 2–5 When to Use Session Reusable Database Services Versus Transaction Reusable Database Services

	Database Service	
Attribute	**Session Reusable**	**Transaction Reusable**
If client API requests are:	Long duration Unknown length	Short duration Known length

If service use frequency is:	Infrequent
	Set no. executors:
	Min=0
	Max=high value
		Not applicable to use service use frequency, set no. executors:
		Min=Max (required)

	Frequent Set no. executors: Min=Max	
If number of clients per executor is:	1 (required)	>1 If short transactions, set to a higher number. If longer transactions, set to a lower number.

2.7 Setting Up Security on Servers

Oracle SQL/Services, in combination with the underlying database engine, provides various security mechanisms that you can employ to control the services and data that users are allowed to access. An Oracle SQL/Services server environment can be viewed as having three tiers where security is checked. The tiers are as follows:

- **Tier 1: Client identification and authentication**

  The Oracle SQL/Services server first checks the identification and authentication of users requesting access to the server. This occurs when the client first connects to the server.

- **Tier 2: Service access authorization**

  The Oracle SQL/Services server next checks that each user requesting access to a particular service has been authorized to use that service.

- **Tier 3: Database and data access authorization**

  Finally, the underlying database engine checks each database access request made by an executor process.

Each of these security tiers is discussed in the sections that follow.

2.7.1 Client Identification and Authentication

The first server security tier is client identification and authentication. This occurs when the client application first connects to the server. The result of the successful completion of the first tier is a connect user name that is used for authorization checks in subsequent tiers.

Typically, a user will supply a user name and password when accessing a service. When a client connects to an Oracle SQL/Services server, the server ensures that the user's account exists on the system and that the password is valid. Following successful authentication, the client-supplied user name is used as the connect user name. If the user name and password check fails or if the password has expired, then the connection is rejected and an error message is returned to the client. If the user does not supply a user name and password, the server then checks the network transport of the connection.

If the client selected the DECnet transport, then the server checks to see if a proxy exists for the node name or user name of the client or both. The server first looks up the client's DECnet node name and DECnet user name, if any, in the Oracle SQL/Services proxy file, SYS$STARTUP:sqlsrv$proxy.dat. If a match is found, then the local proxy user name is used as the connect user name. If no match is found, but the client is on the same node as the server, then the user name of the client process is used as the connect user name.

For the system management client only, the server uses the user name of the client process as the connect user name, if the user:

- Selected the TCP/IP transport
- Logged in to the server node
- Has SYSPRV or BYPASS privileges

As a system administrator, you can choose to allow access to a service without requiring a user name and password by specifying a default connect user name. If the user does not supply a user name and password and a default connect user name is not specified (unknown users are not authorized to access the service), then the connection is rejected and an error message is returned to the client. If the client does not supply a user name and password and if a default connect user name has been specified (unknown users are authorized to access the service), then the connect user name is set to the default connect user name. If the client does supply a user name, then the user name is used as the connect user name, regardless of whether or not a default connect user name is specified.

When a system administrator connects to a system management service of a server, the server performs the same user name and password check as when a client connects to a service. If the user name and password checks fail, then the connection is rejected and an error message is returned to the system management application. You cannot specify a default connect user name for the system management service; therefore, you cannot authorize unknown users to access the system management service.

2.7.2 Service Access Authorization

The second server security tier verifies that the user is authorized to access the selected service.

Each service has a list of user names and identifiers that are authorized to access the service. When you create a new service, only the service owner, a privileged user with the SYSPRV privilege, is authorized to access the service. As a system administrator, you are responsible for granting appropriate users access to services provided by the server. You can grant access to a service based on an individual user name, an identifier, or you can grant access to a service to all users (for example, GRANT USE ON SERVICE GENERIC TO PUBLIC).

When a user connects to a service, the server checks to see if the connect user name or an identifier held by the connect user name has been authorized to use the service, or if access to use the service has been granted to all users. If the user is not authorized to access the service, then the connection is rejected and an error message is returned to the client.

A system management service of a server also has a list of user names and identifiers that are authorized to access the service and thus manage the server. When you create a server, typically done as part of the installation, only the privileged user with SYSPRV privilege is authorized to manage the server. As a system administrator, you are responsible for granting access to any additional users who will manage the server. If an unauthorized user attempts to connect to a system management service of a server, then the connection is rejected and an error message is returned to the system management application.

2.7.3 Database and Data Access Authorization

The third and final server security tier occurs at the database level in an executor process. Whenever an executor process executes an SQL statement, the underlying database engine performs a security check to determine if the user name executing the request is authorized to do so. Oracle SQL / Services allows database requests to be executed using either the connect user name or the service owner, depending on the type of service you are providing and the version of Oracle Rdb specified for the service. As a system administrator, you determine which user name is authorized by the database engine by specifying the database access authorization attribute of each service to be either the connect user name or service owner.

- Database access authorization set to connect user name

  If you set the database access authorization to connect user name, then the underlying database uses the connect user name to determine if a client is authorized to execute a database request. The connect user name is the client-specified user name, a DECnet proxy user name, or the default connect user name.

  Note: You cannot use database services with database access authorization set to the connect user name to provide access to remote Oracle Rdb databases. However, Oracle SQL / Services does allow you to provide a database service for a remote Oracle Rdb database if you create the service with database access authorization set to the service owner.

- Database access authorization set to service owner

  If you set the database access authorization to service owner, then the underlying database uses the service owner's user name to determine if a client is authorized to execute a database request.

Table 2–6 illustrates that you can provide different combinations of service types to clients with Oracle SQL / Services V6.1 and lower and Oracle SQL / Services V7.0 and higher.

Table 2–6 Comparison of Service Combinations Between Previous Versions and Version 7.0 and Higher

		Oracle SQL/Services Version	
Service Type	**Database Access Authorization**	**V6.1 and Lower**	**V7.0 and Higher**
Universal	Connect user name	Supported	Supported
Universal	Service owner	Not supported	Supported
Database	Connect user name	Not supported	Supported[1]
Database	Service owner	Supported	Supported

[1] Requires Oracle Rdb V6.1 or higher.

2.7.4 How Server Security Tiers Work Together

Figure 2–4 illustrates how the three server security tiers work together for three connect examples in which a client logs in to the system. Each example shows client identification and authentication, service access authorization, and the resulting database and data access authorization based on the service definition for each service.

Figure 2–4 Oracle SQL/Services Server Security

	Connect 1	Connect 2	Connect 3
Service Definition	Service name=X Owner='fred' Attach='payroll_db' Database authorization= service owner No default connect username argument Grant use to 'freda', 'ned'	Service name=Y Owner='bert' No attach argument Database authorization= connect username No default connect username argument Grant use to PUBLIC	Service name=Z Owner='joe' Attach='account_db' Database authorization= connect username Default connect username='jane' Grant use to 'janet', 'jane'
Client Connect to Server	User name='ned' Password='pwned' Service name=X	User name='holly' Password='pwholly' Service name=Y	No user name No password Service name=Z
Tier 1: **Client Identi-fication and Authentication**	User name='ned' Authenticated using password 'pwned' Connect user name set to 'ned'	User name ='holly' Authenticated using password 'pwholly' Connect user name set to 'holly'	Default connect username='jane' Connect user name set to 'jane'
Tier 2: **Service Access Authorization**	Connect user name 'ned' authorized access to service X using 'ned'	Connect user name 'holly' authorized access to service Y using PUBLIC	Connect user name 'jane' authorized access to service Z using 'jane'
Tier 3: **Database and Data Access Authorization**	Database attached using service owner user name 'fred' Data accessed using service owner user name 'fred'	Database attached using connect user name 'holly' Data accessed using connect user name 'holly'	Database attached using service owner user name 'joe' Data accessed using connect user name 'jane'

First Connect Example

In the first connect example, a user requests access to service X. The user specifies a user name and a password, so these are authenticated by the server in the first security tier and the connect user name is set to 'ned'. No default connect user name is specified, so unknown users are not allowed to access service X; therefore, all users requesting access to service X must supply a valid user name and a password. In the second security tier, the server checks that the connect user name, ’ned’ in this example, is authorized to access the service. Users ’freda’ and ’ned’, as well as a privileged user with SYSPRV privilege, have been granted the right to use service X. So user ’ned’ is authorized to access the service. Service X is a database service; therefore, the executor process attaches to the ’payroll_db’ database by using the service owner’s user name, ’fred’. Database access authorization for service X is set to the service owner, so all database attaches and data access requests are also made under the service owner’s user name, ’fred’ in this example.

Second Connect Example

In the second connect example, a user requests access to service Y. The user specifies a user name and a password, so these are authenticated by the server in the first security tier and the connect user name is set to ’holly’. No default connect user name is specified, so unknown users are not allowed to access service Y; therefore, all users requesting access to service Y must supply a valid user name and password. In the second security tier, the server checks that the connect user name, ’holly’ in this example, is authorized to access the service. All users have been granted the right to use service Y, so user ’holly’ is authorized to access the service. Service Y is a universal service; therefore, an executor is not preattached to a specific database. Database access authorization for service Y is set to the connect user name, so all database attachments and data access requests are made under the connect user name, ’holly’ in this example.

Third Connect Example

In the third connect example, a user requests access to service Z. The user does not specify a user name and a password, so the server checks if unknown users are authorized to access the requested service. A default connect user name is specified, so unknown users are allowed to access service Z as user ’jane’; therefore, the connect user name is set to ’jane’. In the second security tier, the server checks that the connect user name, ’jane’ in this example, is authorized to access the service. Users ’janet’ and ’jane’, as well as a privileged user with SYSPRV privilege have been granted the right to use service Z, so user ’jane’ is authorized to access the service. Service Z is a database service; therefore, the executor process attaches to the ’account_db’ database by using the service owner’s user name, ’joe’. However, database access authorization for service Z is set to the connect user name, so all data access requests are made under the connect user name, ’jane’ in this example.

2.8 Deciding Which Types of Service to Provide to Clients

The guidelines in Section 2.8.1 to Section 2.8.5 can help you to decide the attributes of the services you provide to clients.

2.8.1 Which Services to Provide?

The following guidelines can help you to decide whether to provide clients with a universal service or a database service.

Provide One or More Universal Services If:

- Your system has a large number of databases that are accessed infrequently and where creating a database service for each database would be unmanageable.
- You have legacy applications or third-party applications that are able to select the database to be used only by connecting to a universal service and executing an SQL ATTACH statement.
- You are providing an application development environment where application developers need full control over the databases they are using.
- You need to provide access to remote Oracle Rdb databases when database authorization is by connect user name.

Provide Database Services If:

- Your system has a number of frequently accessed databases that can be managed easily by using database services.
- You want to provide fast client connection times using prestarted executor processes that are preattached to the appropriate database.
- You have one or more databases with suitable transaction workload characteristics that can take advantage of the performance gains made possible with transaction reusable database services.

Note: You cannot use database services with database access authorization set to the connect user name to provide access to remote Oracle Rdb databases. However, Oracle SQL/Services does allow you to provide a database service for a remote Oracle Rdb database if you create the service with database access authorization set to the service owner.

2.8.2 Setting Database Access Authorization?

The following guidelines can help you understand and decide what type of service to provide to clients and whether or not to set database access authorization to the connect user name or the service owner.

Universal Services

Database Access Authorization Set to Connect User Name For clients using a universal service, set database access authorization to the connect user name if you want client applications to attach to and access databases by using the client-supplied user name, the DECnet proxy user name, or the default connect user name. With database access authorization set to the connect user name, client access to databases is based on the use granted to individual users or groups of users using the underlying database security mechanisms.

Example 2–1 illustrates how to create the universal service named GENERIC. Note that GENERIC is the service name that an Oracle SQL/Services or Oracle ODBC Driver for Rdb client will use by default if no service name is supplied. This universal service has database access authorization set to the connect user name, access granted to all users, a minimum of 1 executor process, and a maximum of 20 executor processes.

Example 2–1 Default Universal Service with Database Access Authorization Set to Connect User Name

```
SQLSRV> CREATE SERVICE GENERIC
_SQLSRV>       OWNER 'SQLSRV$DEFLT'
_SQLSRV>       DATABASE_AUTHORIZATION CONNECT USERNAME
_SQLSRV>       MIN_EXECUTORS 1
_SQLSRV>       MAX_EXECUTORS 20;
SQLSRV> GRANT USE ON SERVICE GENERIC TO PUBLIC;
SQLSRV> START SERVICE GENERIC;
```

Database Access Authorization Set to Service Owner For clients using a universal service, set database access authorization to the service owner only if you need client applications to attach to and access databases by using a single, fixed user name, the service owner user name. Use this approach if you have one or more databases that must be accessed under a fixed user name using a universal service. You can use the GRANT USE ON SERVICE command to restrict the users that can access such a service.

Caution: If you set database access authorization to the service owner for a universal service, be sure that the service owner user name does not have access to any databases containing secure or sensitive data that would otherwise be protected against access from unauthorized users.

Usually, you will *not* set database authorization to service owner for a universal service.

Example 2–2 illustrates how to create a universal service that might be used for testing purposes that has database access authorization set to the service owner. Authorization to use the service is granted to only two development accounts in addition to the service owner user name account.

Example 2–2 Universal Service with Database Access Authorization Set to Service Owner

```
SQLSRV> CREATE SERVICE GEN_DEVEL OWNER ’noprivs’
_SQLSRV>                         DATABASE_AUTHORIZATION SERVICE OWNER
_SQLSRV>                         MIN_EXECUTORS 0
_SQLSRV>                         MAX_EXECUTORS 5;
SQLSRV> GRANT USE ON SERVICE GEN_DEVEL TO ’develop’, ’test’;
SQLSRV> START SERVICE GEN_DEVEL;
```

Database Services

Database Access Authorization Set to Connect User Name For clients using a database service, set database access authorization to the connect user name if you want clients to access the database by using the client-supplied user name, the DECnet proxy user name, or the default connect user name. With database access authorization set to the connect user name, client access to the database is based on the use granted to individual users or groups of users using the underlying database security mechanisms.

Example 2–3 illustrates how to create a database service to access the policies and procedures database of a company where the database is accessed under the client's user name. Access to the service is granted to all users, while access to data in the database is based on the underlying database security mechanisms. Unknown users are authorized to use the service under the default connect user name 'readpp', which has read-only access to data in the database. The service is owned by the

’ppdb’ account, which will be used to attach to the database when an executor process is started.

Example 2–3 Session Reusable Database Service with Database Access Authorization Set to Connect User Name

```
SQLSRV> CREATE SERVICE P_AND_P
_SQLSRV>                ATTACH ’FILENAME pp_disk:[pp]pp_database’
_SQLSRV>                OWNER ppdb
_SQLSRV>                DATABASE_AUTHORIZATION CONNECT USERNAME
_SQLSRV>                DEFAULT_CONNECT_USERNAME readpp
_SQLSRV>                MIN_EXECUTORS 0
_SQLSRV>                MAX_EXECUTORS 10;
SQLSRV> GRANT USE ON SERVICE P_AND_P TO PUBLIC;
SQLSRV> START SERVICE P_AND_P;
```

Database Access Authorization Set to Service Owner For clients using a database service, set database access authorization to service owner if you want client applications to access the database by using the service owner user name. Use this approach when you want to grant access to specific data within the database and to specific database operations to a single user name by using the underlying database security mechanisms, and then grant use of the service to a restricted set of user names by using the GRANT USE ON SERVICE command.

Example 2–4 illustrates how to create a database service to access the order-entry database of a company where the database is accessed under the service owner user name, 'ordent'.

Access to the service is granted only to the 'ordent1', 'ordent2', 'ordent3', and 'ordmgr' users, in addition to the service owner and privileged users with SYSPRV privilege. The database name, 'oe_database', is defined as a logical name OE_ DISK:[OE]OE_DATABASE, so the database can be physically moved if necessary without having to modify the service definition.

The transaction workload characteristics of the database allow the service to be transaction reusable, support up to 100 users distributed over 5 executor processes, and have up to 20 users per process.

Example 2–4 Transaction Reusable Database Service with Database Access Authorization Set to Service Owner

```
SQLSRV> CREATE SERVICE ORD_ENT REUSE SCOPE IS TRANSACTION
_SQLSRV>                        ATTACH ’FILENAME OE_DATABASE’
_SQLSRV>                        OWNER ordent
_SQLSRV>                        DATABASE_AUTHORIZATION SERVICE OWNER
_SQLSRV>                        MIN_EXECUTORS 5
_SQLSRV>                        MAX_EXECUTORS 5
_SQLSRV>                        CLIENTS_PER_EXECUTOR 20;
SQLSRV> GRANT USE ON SERVICE ORD_ENT TO ordent1,
_SQLSRV>                                 ordent2,
_SQLSRV>                                 ordent3,
_SQLSRV>                                 ordmgr;
SQLSRV> START SERVICE ORD_ENT;
```

2.8.3 Specify a Default Connect User Name?

The following guidelines can help you decide whether or not to specify the default connect user name to authorize unknown users' access to databases on your system through either a universal service or a database service.

Using Universal Services

Specify a default connect user name for a universal service only if you need to allow unknown users access to databases on your system. You may choose this approach in a development environment to allow simple access to databases used for testing and debugging.

Caution: If you specify a default connect user name to authorize use of a universal service to unknown users, ensure that any databases containing secure or sensitive data are protected with the appropriate access restrictions at the database level.

Usually, you will not specify a default connect user name to authorize use of a universal service to unknown users.

Using Database Services

Specify a default connect user name to authorize use of a database service to unknown users if you want to allow access to data in a particular database without requiring a user name and password. For example, you may consider providing access to nonsensitive, public-access data in a database by using this mechanism in combination with database access authorization set to the connect user name (see Example 2–3).

2.8.4 Grant or Restrict Access to a Service?

The following guidelines can help you decide whether to grant access to a service to all users or restrict access to a service to a specified list of users.

Grant Access to a Service to All Users If:

- You have universal or database services where database access authorization is set to the connect user name and you want to provide all users with the most flexible method of access to data in databases on your system, subject to underlying database security in individual databases.
- You have a database service where database access authorization is set to the service owner, but access to the database by using the service owner user name is restricted to nonsensitive, public-access data that you want to make available to all users.

Restrict Access to a Service to a Specified List of Users If:

- You have a universal service with database access authorization set to the service owner in order to access a set of databases using a fixed user name.
- You have a database service with database access authorization set to the service owner where you want to grant access to data in a database to a single user name by using the underlying database security mechanisms, and then control access to that data by using Oracle SQL/Services security mechanisms.

Caution: Restricting access to services to a specified list of user names by using Oracle SQL/Services does not prevent other users from trying to log in to your system and attempting to access the same databases (using a tool such as interactive SQL) provided by those services. Even if you restrict access to a service to a specified list of user names, you should still protect secure and sensitive data in databases by using underlying database security mechanisms.

2.8.5 Provide Arbitrary or Predefined Access to Data?

The following guidelines can help you decide whether to provide arbitrary access to data or predefined access to data.

Arbitrary Access to Data

You restrict the tables that users can access and the operations that they can perform on those tables by using either underlying database security mechanisms alone or in combination with Oracle SQL / Services security mechanisms. However, once access to data has been granted, users can then execute arbitrary SQL statements against that data, subject to the access they have been granted. For example, if users have INSERT access to a table, they can insert any data they wish into that table. In some situations, allowing arbitrary access to data in a database may not be desirable.

Predefined Access to Data

In some situations, it is desirable to restrict users' ability to manipulate data to a set of predefined operations. You do this by creating a set of definer's rights stored procedures that provide all of the necessary access to data in one or more tables. By restricting access to the tables to the stored procedures' definer's user name, you prevent access from all other users. You then grant access to the stored procedures by using either underlying database security mechanisms alone or in combination with Oracle SQL / Services security mechanisms.

2.9 Considering Security for Selecting the Service Owner User Name

The security criteria that you use to select and configure an account for use as a service owner account is based on the platform on which the server is running, the type of services you are providing, and the database authorization attribute of the services.

2.9.1 Execution Environment for Database Requests

The following guidelines can help you select and configure an account for use as a service owner account on OpenVMS systems based on the service type and the database authorization attribute of the service.

Universal Services

Database Access Authorization Set to Connect User Name For a universal service with database authorization set to connect user name, you should select an account with a nonsystem user identification code (UIC) that has minimal privileges. The Oracle SQL / Services installation procedure creates a nonprivileged account named SQLSRV$DEFLT that may be used for all universal services with database authorization set to connect user name.

Database Access Authorization Set to Service Owner For a universal service with database authorization set to service owner, you should select an account with a nonsystem UIC that has minimal privileges and that has been granted the necessary database access to only those databases that are designed to be accessed by the service. To ensure the security and integrity of your data, the account you select will usually be severely restricted in the access it has to databases on your system and the data contained therein.

Database Services

Database Access Authorization Set to Connect User Name For a database service with database authorization set to connect user name, you should select an account with a nonsystem UIC that has minimal privileges. Because all database requests are executed using the connect user name, the account you select as the service owner user name need only be granted the right to attach to the database. For example, by granting to the SQLSRV$DEFLT account the right only to attach to the database, you can use the nonprivileged account created by the Oracle SQL/Services installation procedure.

Database Access Authorization Set to Service Owner For a database service with database authorization set to service owner, you should select an account with a nonsystem UIC that has minimal privileges and that has been granted the right to access certain specific data within the database and that has been granted the right to execute certain specific operations against that data. The amount of access you grant to the service owner account will be specific to each database for which you provide a database service with database authorization set to service owner.

You must configure each service owner account with minimum settings for the following process quotas shown in Table 2–7 to ensure that an executor process can successfully attach to a database and execute requests against that database. If you do not configure a service owner account with sufficient values for these quotas, then database requests may fail with a variety of errors based on the particular quota the executor exhausts. The minimum values suggested in Table 2–7 should be sufficient for most applications; however, complex applications may require higher values.

Table 2–7 Suggested Minimum Process Quota Values for the Service Owner Account

Name	Description	Suggested Minimum Setting
FILLM	File limit	50
BIOLM	Buffered I/O limit	60
DIOLM	Direct I/O limit	60
ASTLM	AST limit	250

Table 2–7 (Cont.) Suggested Minimum Process Quota Values for the Service Owner

Name	Description	Suggested Minimum Setting
TQELM	Timer queue entry limit	255
ENQLM	Enqueue limit	18000
BYTLM	Nonpaged pool limit	50000
PGFLQUO	Page file quota	40000

You should configure each service owner account with appropriate values for the following process quotas shown in Table 2–8. Although an executor process will function correctly unless these quotas are considerably underconfigured, you may be able to improve the performance of your system by increasing the values of these quotas.

Table 2–8 Suggested Minimum Process Quota Values for Working Set Parameters for the Service Owner Account

Name	Description	Suggested Minimum Setting
WSDEF	Working set default	2048
WSQUO	Working set quota	3072
WSEXTENT	Working set extent	4096

See the *Oracle Rdb Installation and Configuration Guide, Release 7.1* for more information on Oracle Rdb requirements.

2.9.2 Execution Environment for External Functions and Procedures

You can define external functions and procedures to execute within the context of the executor process or in an independent server process that Oracle Rdb creates specifically to execute external functions and procedures.

To define an external function or procedure to execute within the context of the executor process, use the SQL BIND ON CLIENT SITE syntax. From the perspective of the Oracle Rdb database engine, the database client is the Oracle SQL/Services executor process, not the Oracle SQL/Services client. To define an external function or procedure to execute in an independent server process, use the SQL BIND ON SERVER SITE syntax.

See Section 2.8 for a complete, in-depth discussion of how Oracle SQL/Services implements the database access authorization models. See the *Oracle Rdb7 SQL*

Reference Manual and the *Oracle Rdb7 Guide to SQL Programming* for more information on defining external functions and procedures.

2.9.2.1 External Functions and Procedures Executing in the Context of the Executor Process

Because you can define external functions and procedures to execute within the context of the executor process, you should consider this when you configure service owner accounts.

Universal Services

Database Access Authorization Set to Connect User Name External functions and procedures defined to execute in the context of the executor process always execute with the rights and privileges of the connect user name using this type of service.

Database Access Authorization Set to Service Owner External functions and procedures defined to execute in the context of the executor process always execute with the rights and privileges of the service owner user name under this type of service.

Database Services

Database Access Authorization Set to Connect User Name External functions and procedures defined to execute in the context of the executor process always execute with the rights and privileges of the *service owner user name* under this type of service. This is because Oracle SQL/Services cannot reconfigure an executor process once it has attached to the database. To have external functions and procedures execute with the rights and privileges of the connect user name, you must define the external functions and procedures to execute in an independent server process using the SQL BIND ON SERVER SITE syntax.

Database Access Authorization Set to Service Owner External functions and procedures defined to execute in the context of the executor process always execute with the rights and privileges of the service owner user name under this type of service.

2.9.2.2 External Functions and Procedures Executing in the Context of an Independent Process

You can define external functions and procedures to execute within the context of an independent server process. With this model, the execution environment for external functions and procedures is based on the database authorization attribute regardless of whether you are using universal or database services.

Universal and Database Services

Database Access Authorization Set to Connect User Name
External functions and procedures defined to execute in the context of an independent server process always execute with the rights and privileges of the connect user name with this type of authorization.

Database Access Authorization Set to Service Owner
External functions and procedures defined to execute in the context of independent server process always execute with the rights and privileges of the service owner user name with this type of authorization.

2.10 Setting the MIN_EXECUTORS, MAX_EXECUTORS, and IDLE_ EXECUTOR_TIMEOUT Attributes

The use of some services on your system may be fairly constant over time, whereas the use of other services may vary over time with peaks and lulls in the user activity. By setting appropriate values for the MIN_EXECUTORS, MAX_ EXECUTORS, and IDLE_EXECUTOR_TIMEOUT attributes for a service, you can provide an efficient service to your clients.

You must always set the MIN_EXECUTORS attribute to the same value as the MAX_EXECUTORS attribute for a transaction reusable service. This is to allow Oracle SQL/Services to distribute new client connections evenly over the pool of available executor processes for a service.

2.10.1 Configuring a Fixed Number of Executors for a Service

For a service that has a fairly constant number of users connected to it over time, usually you will set the MIN_EXECUTORS attribute to the same value as the MAX_ EXECUTORS attribute. This ensures that a constant number of executors are prestarted and are always available to client applications. This avoids the delay while an executor process is started for a new client connection.

2.10.2 Configuring a Variable Number of Executors for a Service

For a service where the number of connected users varies over time, with more users at peak times and fewer users at less busy times, you can choose to adjust the number of executors to suit any load. To do this, you can choose to have the Oracle SQL/Services server automatically start new executor processes as they are needed, or you can prestart new executor processes in anticipation of increased demand at peak times.

2.10.2.1 Starting New Executor Processes as They Are Needed

The simple approach to handling peaks and lulls in the demand for a service is to set the MIN_EXECUTORS attribute to a value that supports the activity for the service at normal times, set the MAX_EXECUTORS attribute to a value that supports the activity for the service at peak times, then let Oracle SQL/Services create new executor processes as demand increases during peak periods of use. By choosing a suitable value for the IDLE_EXECUTOR_TIMEOUT attribute, you can ensure that executors remain active once they have been started, even if demand might decrease for a little while. Although this approach is very easy to configure and manage, a disadvantage with this approach is that new users who connect to the service at the beginning of a peak period will encounter a slight delay if new executor processes must be created.

2.10.2.2 Prestarting New Executor Processes Ahead of Increased Demand

A more complex approach to handling peaks and lulls in the demand for a service is to set the MAX_EXECUTORS attribute to a value that supports the activity for the service at peak times, then create SQLSRV_MANAGE scripts that can be used to increase the value of the MAX_EXECUTORS attribute for the service at peak times and decrease the value of the MAX_EXECUTORS attribute for the service at a time when demand for the service starts to decrease.

You can automatically invoke the SQLSRV_MANAGE scripts to increase the MAX_EXECUTORS attribute value in anticipation of the increase in the number of users of a service and decrease the MAX_EXECUTORS attribute value at the end of a peak period by writing command procedures for batch jobs. The advantage of this approach of prestarting executors ahead of demand is that new users who connect to the service at the beginning of a peak period will not encounter delays as executor processes are created. A disadvantage of this approach is that it is more complex to manage.

2.11 Using an SQL Initialization File

You can use the SQL_INIT_FILE argument of the CREATE SERVICE or ALTER SERVICE command to specify a file containing SQL statements that tailor the SQL environment for a client connection. For example, you can set the SQL dialect and default character set by using an SQL initialization file. The statements in an SQL initialization file are executed every time a client connects to a service.

See Section 4.1 for more information about syntax conventions used in an SQL initialization file.

2.12 Understanding Database Access Authorization Models

In Section 2.7, Section 2.8, and Section 2.9, you learned that Oracle SQL/Services allows you to authorize database access using the service owner user name or the connect user name. You also learned how these models affect the environment within which database requests and external functions are executed. This section describes in detail how Oracle SQL/Services implements database authorization by connect user and by service owner.

2.12.1 Accessing an Oracle Rdb Database

To understand how Oracle SQL/Services implements database authorization by connect user name and by service owner, it is first necessary to understand that four user names are involved in accessing an Oracle Rdb database in the Oracle SQL/Services environment:

- Operating system process user name
- Oracle Rdb system user name
- Oracle Rdb session user name
- Oracle Rdb current user name

Following is an explanation of the four user names.

2.12.1.1 Operating System Process User Name

The process user name is the user name under which an Oracle SQL/Services executor process runs a local attach, or the user name of the Oracle Rdb remote server process in a remote attach.

The process user name is set based on the SERVICE OWNER service attribute for local attaches, whereas it is based on the ATTACH statement and the configuration of the remote Oracle Rdb server node for remote attaches. Associated with the process user name are a number of process attributes. These attributes include:

- UIC
- Privileges
- Rights list
- Account name
- Default directory
- Logical names, including
  - SYS$DISK
  - SYS$LOGIN_DEVICE
  - SYS$LOGIN
  - SYS$SCRATCH
  - LNM$GROUP (for group logical name table)

2.12.1.2 Oracle Rdb System User Name

Each attached database in an executor process has a value for the system user name. The Oracle Rdb system user name is used to determine if the process is authorized to attach to the database and also serves as the default value for the Oracle Rdb session user name.

The Oracle Rdb system user name for an attached database defaults to the process user name but may be overridden by the SQL ATTACH statement attribute of a database service or by a client application accessing a universal service, depending on the type of service being provided, the attributes of that service, and the version of Oracle Rdb being used.

The Oracle Rdb system user name for an attached database is established at the time of attachment to the database and remains fixed for the life of the attachment. You can override the default value for the system user name when using Oracle Rdb V6.1 or higher by specifying a user name and a password in the attach-string argument of an SQL ATTACH statement or in the connect-string argument of an SQL CONNECT statement. See the *Oracle Rdb7 SQL Reference Manual* for more information on the SQL ATTACH and CONNECT statements.

The number of attached databases in an executor process providing a universal service is determined by the client application. Different attached databases may have different system user names.

An executor process providing a database service has only one attached database.

2.12.1.3 Oracle Rdb Session User Name

All database requests are executed within the context of an SQL connect. Each SQL connect in an executor process has a value for the session user name. The session user name for an SQL connect defaults to the Oracle Rdb system user name, but may be overridden by Oracle SQL / Services or by a client application, depending on the type of service being provided, the attributes of that service, and the version of Oracle Rdb being used.

The session user name for an SQL connect is determined at the time the SQL connect is established and remains fixed for the life of the SQL connect. You can override the default value for the session user name when using a universal service with Oracle Rdb V6.1 or higher by specifying a user name and a password as arguments to the SQL CONNECT statement. See the *Oracle Rdb7 SQL Reference Manual* for more information on the SQL CONNECT statement.

The number of SQL connects in an executor process providing a universal service is determined by the client application. Different SQL connects may have different session user names. An SQL connect in an executor process providing a universal service can reference one or more than one database attach.

The number of SQL connects in an executor process providing a database service is determined by the version of Oracle Rdb in use and the service reuse attribute. See Section 2.6 for information on reuse attributes.

The ATTACH statement of a database service is always executed in the context of the default SQL connect. You cannot use an SQL CONNECT statement to attach to a database using a database service. The session user name for the default connect defaults to the system user name. If an SQL initialization file is specified for the service, then the statements contained therein are executed in the context of the default SQL connect after the SQL ATTACH statement.

For Oracle Rdb V6.1 and higher, a new SQL connect is created for each client application that connects to the service. If the service is defined with database authorization by the service owner, then the session user name for each SQL connect of a client application defaults to the system user name. If the service is defined with database authorization by connect user, then each SQL connect of a client application is created using the connect user name for each individual client connection. When a client application disconnects from the service, the SQL connect of the client application is deleted. For a session reusable database service, there is a maximum of one client application SQL connect per executor. For a transaction reusable database service, there is one client application SQL connect for each concurrent client connection.

2.12.1.4 Oracle Rdb Current User Name

The current user name is always set to the value of the session user name except during the execution of a definer's rights stored procedure, in which case, the current user name is set to the definer's user name.

Whenever a database request is started, Oracle Rdb must determine if the process issuing the request is authorized to execute the request. To perform this check, Oracle Rdb first merges the system privileges of the process accessing the database with the database privileges of the current user name. For a local attach, the process accessing the database is the Oracle SQL / Services executor process. For a remote attach, the process accessing the database is the Oracle Rdb server process.

The process privilege mask of the operating system is used as the system privileges for the executor process.

After Oracle Rdb merges the privileges, it then determines if the combination of these privileges is sufficient to execute the request. Because Oracle Rdb combines the privileges in this way, you must carefully choose the service owner user name for a database service. See Section 2.9 for more information.

For example, consider a database service called PAYROLL that is defined with a service owner user name of SYSTEM and with database authorization set to the connect user name. User SMITH might not normally be authorized to update a table called EMPLOYEE_PAY in the payroll database. However, if user SMITH accesses the payroll database using the PAYROLL service, the database privileges for user name SMITH, when combined with the system privileges for the SYSTEM user name, which include SYSPRV and BYPASS, allow this user full access to the EMPLOYEE_PAY table and all other tables in the database.

2.12.2 Setting the Process User Name and the Oracle Rdb System User Name

To set the Oracle Rdb system user name, Oracle SQL / Services uses a process user name impersonation mechanism to set the process user name and all associated process attributes of an executor process. By setting the process user name, Oracle SQL / Services automatically establishes the correct default for the Oracle Rdb system user name. Furthermore, by setting the process user name, Oracle SQL / Services also establishes the correct environment for the consistent execution of external functions and procedures that execute within the context of the executor process.

Oracle SQL/Services sets the process user name at different times, based on the type of service you provide:

- Universal service

  Oracle SQL/Services sets the process user name every time a new client connect is assigned to an executor process for a universal service. This ensures the correct environment at all times for the execution of external functions and procedures that execute within the context of the executor process.

- Database service

  Oracle SQL/Services sets the process user name once for an executor process for a database service at the time the executor process is first started. However, to ensure the correct and successful execution of database requests once an executor is attached to a database, Oracle SQL/Services cannot and does not reset the process user name when a new client connect is assigned to an executor process. This behavior provides a consistent environment for the execution of external functions and procedures that execute within the context of the executor process. However, it means that all such functions and procedures are executed under the service owner user name, rather than the connect user name for a service with the database authorization attribute set to connect user. See Section 2.9.2 for more information on using external functions and procedures with Oracle SQL/Services, including information on how to define external functions and procedures to execute within the context of an independent server process with the rights and privileges of the connect user name.

When Oracle SQL/Services creates an executor process, the Oracle SQL/Services monitor process merges the *authorized* and *default* privileges of the service owner account. The combination of these privileges becomes the *authorized privilege* mask of the executor process. When Oracle SQL/Services resets the process user name of an executor process, it sets the *process privilege* mask and *current privilege* mask of the executor process. To set the *process privilege* and *current privilege* masks of an executor process, Oracle SQL/Services merges the *authorized* and *default* privileges of the new process user name. However, Oracle SQL/Services cannot set the *authorized privilege* mask of an executor process. Therefore, you must ensure that a service owner account does not have excess *authorized* or *default* privileges. Typically, you will grant only the TMPMBX and NETMBX privileges to a service owner account.

3

Maintaining an Oracle SQL/Services Server

After you set up Oracle SQL / Services and configure one or more servers, you should periodically perform maintenance tasks, which include:

- Relocating log, dump, and lock files (optional)
- Monitoring server activity
- Monitoring client connections
- Recovering from failures
- Isolating problems
- Solving server errors
- Submitting software problem reports

Each of these topics is described in the sections that follow.

3.1 Monitoring Server Activity

Monitoring server activity consists in part of using the SHOW commands to show the operational state of objects. For example, for service and dispatcher objects, a SHOW command will inform you if the object is running. If you find that a service or dispatcher object is not running and should be running, then it probably failed and you should check the log and dump files to determine why the object stopped running. After resolving the problem, issue either a START SERVICE or START DISPATCHER command and specify the service or dispatcher name of the object you want to start up. Perform another SHOW command to confirm that the service or dispatcher object is running.

Using the SHOW SERVICES command, you can also monitor client activity during peak load periods for all services provided on that server. For example, if the number of active clients approaches the maximum number allowed, you should consider increasing the maximum number of clients allowed to reduce the chances of client connection failures. You can dynamically increase the MAX_EXECUTORS value for a particular service by using the ALTER SERVICE command.

3.2 Monitoring Client Connections

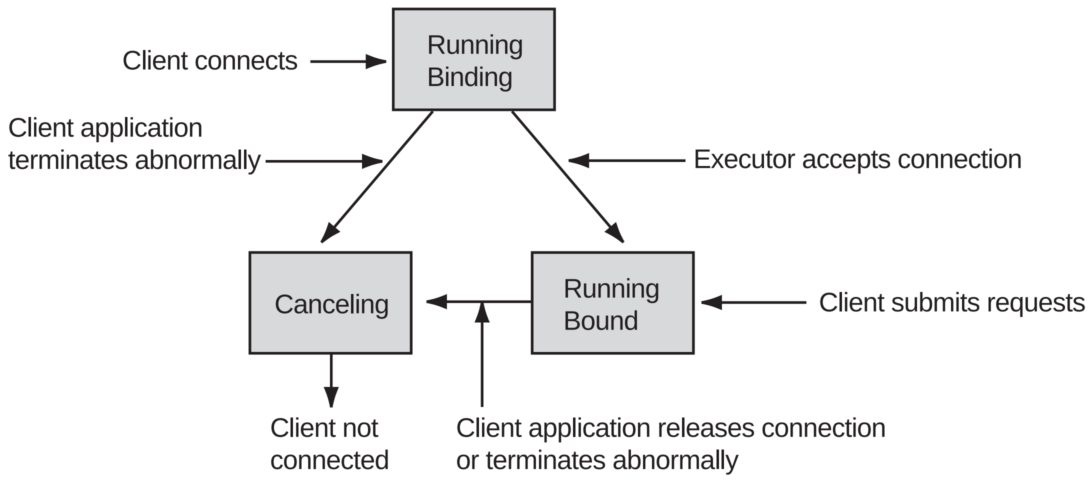
You can use the SHOW CLIENTS command to show the state of clients as each connects to a service, submits requests, and releases the connection. The occurrence, sequence, and duration of connection states are different for each type of service. The client state can help you determine what each connection is doing and if connections are being serviced normally. However, the connection state information by itself may not be sufficient for troubleshooting all problems. For more information on troubleshooting problems, see Section 3.4.2.

Section 3.2.1 and Section 3.2.2 describe the states that a client connection can display, the sequences that can occur, and the relative duration of each state when serviced by either a session reusable service or a transaction reusable database service.

3.2.1 Client Connection States for Session Reusable Services

Figure 3–1 shows the three possible connection states that a SHOW CLIENTS command can display for a client connection when serviced by an executor process for a session reusable service relative to client and executor events. Of these three states, the Running Binding and Running Bound states are of most interest.

Figure 3–1 Client Connection States for Session Reusable Services


Client connects
Running Binding
Client application terminates abnormally
Executor accepts connection
Canceling
Running Bound
Client submits requests
Client not connected
Client application releases connection or terminates abnormally

The connection from a client attempting to connect to a session reusable service is in a Running Binding state while it waits for an executor to accept the connection. A connection is in the Running Binding state only momentarily if a free executor process is available to accept the connection. However, a connection remains in the Running Binding state for a longer period of time if a new executor process must be created for the connection, which may take several seconds.

When an executor process accepts a connection, the connection state transitions from Running Binding to Running Bound. Once an executor for a session reusable service accepts a connection, the executor remains bound to that connection for the duration of the connection. Therefore, Running Bound is the predominant state of a connection assigned to an executor for a session reusable service.

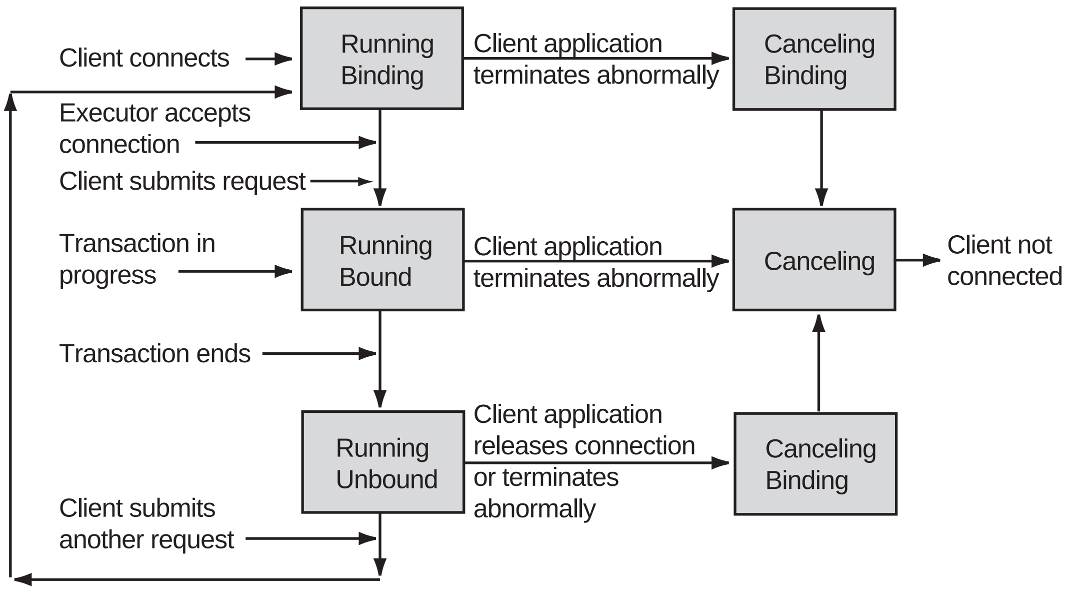
A connection transitions to the Canceling state when the application releases the connection normally, or if the application terminates abnormally. A connection typically remains in the Canceling state only momentarily. However, a connection may remain in the Canceling state for a longer period of time if other database activity delays the cleanup of an outstanding database transaction.

3.2.2 Client Connection States for Transaction Reusable Database Services

An executor for a transaction reusable service processes requests for one transaction for one client at a time; however, the executor is shared by many concurrent client connections. A transaction begins when a client issues an SQL statement that either implicitly or explicitly starts a transaction. A transaction ends when the client issues a successful SQL COMMIT or ROLLBACK statement or executes a stored procedure that commits or rolls back a transaction. Once assigned to an executor process, a client connection remains tied to that process for the life of the connection; no other executor process can be used to process transactions on behalf of a particular connection. A new client connection is normally assigned to the executor with the least number of existing connections; however, for certain applications, it may be necessary to change this behavior using the ALTER SERVICE APPLICATION TRANSACTION USAGE CONCURRENT attribute.

Figure 3–2 shows the five possible connection states that a SHOW CLIENTS command can display for a client connection when serviced by executors for a transaction reusable database service relative to client and executor events. Of these five states, the Running Binding and Running Bound states are of most interest.

Figure 3–2 Client Connection States for Transaction Reusable Database Services


Client connects
Running Binding
Client application terminates abnormally
Canceling Binding
Executor accepts connection
Client submits request
Transaction in progress
Running Bound
Client application terminates abnormally
Canceling
Client not connected
Transaction ends
Running Unbound
Client application releases connection or terminates abnormally
Canceling Binding
Client submits another request

The connection from a new client attempting to connect to a transaction reusable service is in a Running Binding state while it waits for the assigned executor to accept the connection. Likewise, when an existing client begins a new transaction, the connection is in a Running Binding state while it waits for the assigned executor to process the new transaction. A connection remains in the Running Binding state until the executor completes the transaction for the current connection, plus any other transactions for connections that may be queued up already waiting for the executor. For well-designed applications that are executing short transactions, connections remain in the Running Binding state for short periods of time. However, this time increases as the rate at which clients execute transactions increases and as the average length of transactions increases.

When an executor process binds to a new or an existing connection, the connection state transitions from Running Binding to Running Bound. Once bound to a connection, the executor remains bound to that connection until the end of the transaction. In a new connection, the executor remains bound to the connection only for the time necessary to establish a new database session for the new connection. At the end of a transaction, or after accepting a new connection, the executor unbinds from the connection, and the connection state transitions from Running Bound to Running Unbound. When the connection starts a new transaction, the connection state transitions from Running Unbound to Running Binding and the cycle continues.

A connection in the Running Binding or Running Unbound state transitions to the Canceling Binding state when the application releases the connection normally or when the application terminates abnormally. When the executor completes the transaction for the currently bound connection, plus any other transactions for connections that may be queued up already waiting for the executor, the Canceling Binding connection transitions from the Canceling Binding state to the Canceling state. A connection in the Running Bound state transitions directly to the Canceling state when the application releases the connection normally or when the application terminates abnormally. When in the Canceling state, the executor cleans up the database session of the connection, then unbinds from the connection for the last time.

3.3 Recovering from Failures

Oracle SQL/Services handles process failures in different ways depending on the type of process that fails.

3.3.1 Monitor Process Failures

Oracle SQL/Services does not attempt to recover if the monitor process fails. If a monitor process does fail, then all of the processes in the server configuration are shut down. In this way, a monitor process failure does not leave the system in an inconsistent state.

3.3.2 Dispatcher Process Failures

Oracle SQL/Services does not automatically restart a failed dispatcher process; however, the server will continue running unless the failure occurred during a critical operation in which the integrity of the server might be compromised. Therefore, you must manually restart a failed dispatcher process.

3.3.3 Executor Process Failures

Oracle SQL/Services automatically tries to restart a failed executor process unless the failure occurred during a critical operation in which the integrity of the server might be compromised. However, if an executor process fails more than twice during startup, then Oracle SQL/Services shuts down the service and marks it as failed.

3.4 Isolating Problems

You can isolate problems that you might experience with an Oracle SQL/Services server system by:

- Inspecting log files
- Investigating different types of problems

The following sections describe what log files are generated, what is contained in each type of log file, a number of different error conditions that you may encounter, and how to identify a particular problem.

3.4.1 Inspecting Log Files

When a problem arises, you can attempt to isolate the problem by inspecting the log files generated on the client side as well as those generated on the server side.

There are up to three kinds of logging on the client side:

- Oracle SQL/Services client logging
- ODBC logging
- Winsock logging

3.4.1.1 Oracle SQL/Services Client Logging

You enable Oracle SQL/Services client logging by using a parameter to the sqlsrv_ associate routine, or using an sqsapiw.ini file for an sqsapi32.ini file for a Windows

95 (32-bit) client, Windows 98, Windows 2000, or Windows NT X86 client. The Oracle SQL / Services client API creates log files named clientxx.log in the applications default directory. A clientxx.log file records calls to Oracle SQL / Services API services data values and message protocol.

Check a clientxx.log file to see what SQL statements you are passing to the server, or what error messages you are getting back from the server. See the *Guide to Using the Oracle SQL/Services Client API* for more information.

3.4.1.2 ODBC Logging

If you are using the Oracle ODBC Driver for Rdb, you can turn on ODBC logging using a win.ini file for an rdbodbc.ini file for a Windows 95 (32-bit), Windows 98, Windows 2000, or Windows NT X86 client.

This type of logging records ODBC calls and entry points, the SQL statements your application generates, and error messages. The log file is named odbcrdb.log and is located in the working directory of your application.

3.4.1.3 Winsock Logging

If you are using a Winsock transport on Windows 95, Windows 98, Windows 2000, or Windows NT X86, this logging option helps you diagnose network problems. It is enabled using an sqsapi32.ini file for a Windows 95 (32-bit), Windows 98, Windows 2000, or Windows NT X86 client. The log file is named sqsapiw.log and is located in the working directory of your application.

See the *Guide to Using the Oracle SQL/Services Client API* for more information about client log files.

Oracle SQL / Services uses the following convention to generate log file names for server components, where nodename is the node name, component-id is the server component, and version is the version number (for multiversion installation) or is left blank (for a standard installation):

- If the SCSNODE SYSGEN parameter is set

  ```
  sqs_<nodename><component-id><instance><version>.log
  ```

- If the SCSNODE SYSGEN parameter is blank

  ```
  sqs_<component-id><instance><version>.log
  ```

There are several kinds of logging on the server side:

- Oracle SQL / Services monitor log file

- Oracle SQL / Services dispatcher log files
- Oracle SQL / Services executor log files

3.4.1.4 Oracle SQL/Services Monitor Log File

Oracle SQL / Services logs the following information in the monitor log file:

- Dispatcher and executor process startup and shutdown informational messages
- Dispatcher and executor process failure error messages, including names and locations of component log files
- Oracle SQL / Services authentication and authorization failures for Oracle SQL / Services system management clients
- Name and location of a monitor process bugcheck dump if the monitor encounters a nonrecoverable error

Use the following command to list monitor log files:

```
$ DIRECTORY SYS$MANAGER:SQS*MON*.LOG
```

For example:

```
SYS$MANAGER:SQS_NODE1_SQLSRV_MON_0071.LOG
```

3.4.1.5 Oracle SQL/Services Dispatcher Log Files

Oracle SQL / Services logs the following information in a dispatcher log file:

- Oracle SQL / Services authentication and authorization failures for Oracle SQL / Services and Oracle RMU clients
- Server-side client network link disconnections due to executor process failures
- Client-side client network link failures
- Name and location of a dispatcher process bugcheck dump if the dispatcher encounters a nonrecoverable error

Use the following command to list dispatcher log files (assuming you used "dis" in the first 10 characters of the names of all of the dispatchers in your server):

```
$ DIRECTORY SYS$MANAGER:SQS*DIS*.LOG
```

For example, the dispatcher log file name for a dispatcher named SQLSRV_DISP may appear as:

```
SYS$MANAGER:SQS_NODE1_SQLSRV_DIS00371.LOG
```

3.4.1.6 Oracle SQL/Services Executor Log Files

Oracle SQL/Services logs the following information in an executor log file:

- Executor process startup errors
- Oracle Rdb authentication and authorization failures for Oracle SQL/Services clients for database services with database authorization set to connect user
- Oracle Rdb and SQL error messages
- Name and location of an executor process bugcheck dump if the executor encounters a nonrecoverable error
- For OCI services, log messages for SQL*Net for Rdb as specified by the ALTER SESSION LOG BRIEF or ALTER SESSION LOG FULL command.

Executor log files are created in the default directory of the service owner account. For example, use the following commands to list executor log files for a service named GENERIC with a service owner account named SQLSRV$DEFLT that has a default directory of SYS$SYSDEVICE:[SQLSRV$DEFLT].

```
$ DIRECTORY SYS$SYSDEVICE:[SQLSRV$DEFLT]SQS*GENERI.LOG
```

For example:

```
SYS$SYSDEVICE:[SQLSRV$DEFLT]SQS_NODE1_GENERI004000171.LOG
```

3.4.2 Investigating Different Types of Problems

As a system administrator, you may be called upon to investigate a number of different types of problems. The following is a set of general error conditions with guidelines for each that may help you track down and identify a particular problem.

3.4.2.1 Network Transport Problems

A problem sometimes experienced by new users or with a new server configuration is the inability to connect to the server at all. In this situation, client applications receive network (-2003 and -2036) errors from Oracle SQL/Services API routines.

In the event of this type of error, first verify that the dispatcher supporting the selected transport is running and that the specified network port or object is active. If you are using alternate network ports or objects in a multiversion environment, verify that you specified the correct network port or object at the client. If the dispatcher appears to be functioning correctly, use a transport-specific tool, such as the TCP/IP Ping utility, to verify connectivity between the client and server nodes.

If the dispatcher is not running or the selected network port or object is not active, check the dispatcher log to determine the reason for the problem. If a dispatcher process fails completely, then Oracle SQL/Services writes the name and location of the dispatcher log file to the monitor log. Check the dispatcher log file to determine if a bugcheck dump was produced when the dispatcher failed.

3.4.2.2 User Authentication and Authorization Problems

Authentication and authorization (-2034, -2049, and -1028) errors are another class of problems that may be experienced by new users. In this case, the server is functioning correctly. However, users are unable to connect to the server or to a particular service provided by the server.

You should first check the dispatcher log file to determine the reason for the error. Remember to check the appropriate dispatcher log if you have configured multiple dispatchers for different transports. For example, to resolve an authentication or authorization problem, you may need to authorize network access or grant the SQLSRV$CLIENT identifier to a user's account. Or perhaps you need to grant access to a particular service to a new user. All of these types of errors are logged in the dispatcher log file.

If the user is connecting to a database service with database authorization set to connect user, then you must also authorize the user's account to access the database. If a user is not authorized to access the database, then Oracle Rdb returns a no privilege (-1008) error, the text of which Oracle SQL/Services returns to the client application and writes to the executor's log file.

3.4.2.3 Executor Failures During Service Startup

You may sometimes encounter errors when initially creating and starting a new service. Whenever an executor process fails to start correctly, Oracle SQL/Services writes the name of the executor's log file to the monitor log. From the executor log, you can then find the reason for the error.

For example, to determine why a new service fails after you start it, display the contents of the monitor log to determine the log file names of the failed service's executors. Then display the contents of one of the executor log files to determine the reason for the failure. For example, perhaps you typed an invalid SQL ATTACH statement, mistyped the database file name, or perhaps you did not grant the right to attach to the database to the service owner account of a database service. All problems such as these result in the service's SQL ATTACH statement failing.

3.4.2.4 Executor Problems During Client Connect

In some situations, a service may start successfully, but an executor process created for a new client connection might fail during startup. This can happen if the MIN_ EXECUTORS attribute of a service is set to 0. In this case, you can successfully start the service, but the service eventually changes to the failed state as executors created for new client connects fail during startup. This problem can also occur if a database is changed after a service is started. For example, if the right to attach to the database is revoked from the service owner account of a database service after the service is started and the minimum number of executors have been created, then new executors that are created for the service will fail trying to execute the service's ATTACH statement.

If a user tries to connect to a service and an executor created for the new connect fails during startup, the monitor records the executor failure event in the monitor log together with the name of the executor log. The dispatcher then logs a summary error message in the dispatcher log and returns the executor failed (-2035) error code to the client application along with an executor startup error message. If a user tries to connect to a service that previously changed to the failed state, then the dispatcher logs the event in the dispatcher and returns the executor failed (-2035) error code to the client application along with a service failed error message.

To investigate problems of this nature, first check the dispatcher log to determine why new client connects are being rejected. Then review the monitor log to find an entry detailing an executor failure for the service. Finally, check the executor log to determine the reason for the failure.

3.4.2.5 Executor Problems During Client Request Execution

You may experience a situation where most users are successfully accessing a service, but the executor for one particular user fails. In this situation, the dispatcher returns the executor failed (-2035) error to the client application and the monitor records the executor failure event in the monitor log, together with the name of the executor log. You first check the monitor log to determine the name of the log file for the failed executor, then check the log of the executor to determine the reason for the failure. For example, perhaps data being accessed by a particular user is located on a disk that is beginning to fail. Alternatively, perhaps Oracle SQL/Services or Oracle Rdb or SQL encountered an internal error. In this situation, check the executor log file to see if a bugcheck dump file was produced by one of these components.

3.4.2.6 Server Failed Due to an Internal Error

In extremely rare circumstances, it is possible for an entire server to fail. For example, perhaps a component encountered an internal error and failed while performing a critical operation. In this situation, the entire server shuts down so as not to further compromise the integrity of the server configuration.

The Oracle SQL/Services monitor process manages all of the processes in an Oracle SQL/Services server configuration. Therefore, the monitor log file is the best place to start. In this situation, the monitor will always produce a bugcheck dump; however, the reason for the error may have been the earlier failure of a dispatcher or executor process. Therefore, your next step is to review the log files of any dispatchers and executors that failed just prior to the server failure. Check these log files for references to any Oracle SQL/Services and Oracle Rdb and SQL bugcheck dumps.

If you find a reference to a bugcheck dump file while isolating a problem, refer to Section 3.6 for more information about submitting a problem report form to Oracle Corporation. The bugcheck dump file is directed by default to SYS$MANAGER unless you specified another location by using the CREATE SERVER or ALTER SERVER command. The only exception is that executor bugcheck dump files are written to the current default directory.

3.5 Solving Server Errors

The server error message files contain all of the server errors with explanations of the error and possible user actions. PostScript and text versions of the server error message files are located in the following directories:

- SYS$HELP:sqlsrv_messages71.ps - Oracle SQL/Services server error message PostScript file
- SYS$HELP:sqlsrv_messages71.txt - Oracle SQL/Services server error message text file

3.6 Reporting Software Problems

Contact your Oracle Corporation support representative for assistance.

If you experience problems with the server, include the following items on magnetic tape along with your problem report:

- Copies of the monitor log files, dispatcher log files, any applicable executor log files, and any relevant client log files

- A copy of the Oracle SQL/Services configuration file
- Copies of any bugcheck dump files produced

4 Management Commands

This chapter describes the syntax and semantics of the SQLSRV_MANAGE utility of Oracle SQL / Services. This utility is used to manage the Oracle SQL / Services server and its components. See Section 4.1 for a description of syntax conventions.

See the Oracle SQL / Services Manager GUI help for information on using this utility.

The SQLSRV_MANAGE commands include management commands (definitional, show, and operational) and environment commands and switches. Section 4.2 describes how the SQLSRV_MANAGE management commands work.

4.1 Syntax Conventions

The SQLSRV_MANAGE utility uses the following syntax conventions and semantics for both its environment and management commands:

[ ]	Brackets enclose optional clauses from which you can choose none, one, or more of the enclosed options. Do not include brackets in your option.
{ }	Braces indicate that you must choose at least one of the enclosed options. Do not include braces in your option.
|	The vertical bar means that you can select only one of the options shown.
,	The comma means that you can choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.
< >	Angle brackets enclose user-supplied names.
::=	An argument followed by a double colon and equal sign represents the definition of the argument.

White space and new lines	White space and new lines (carriage returns) are not significant in the syntax diagram.
Keywords	Keywords are not case sensitive. Keywords are presented in uppercase characters and are underlined.
. . .	Horizontal ellipsis points in commands mean that parts of the command not directly related to the example have been omitted.
;	All statements must be terminated with a semicolon (;) with the exception of the EXIT and HELP commands, in which the semicolon is optional.

The following syntax and semantics are also used.

<identifier>

An <identifier> is a string starting with a letter and composed of letters (a to z, A to Z), numbers (0 to 9), hyphens (-), and underscores (_). Such identifiers are in uppercase. For example:

```
AARDVARK_1-1101
```

<quoted-string>

A <quoted-string> can use either single or double quotation marks containing any characters within it except a new line character. For example:

```
'user1'
"Today is 5/6/94"
```

Single-quoted strings can contain embedded double quotation marks, and double quoted strings can contain embedded single quotation marks. For example:

```
'Contestant number three said, "My name is Data"'
"Today's beach report is 'sunny and warm'"
```

A new line character inside a string is assumed to be a syntax error; that is, an unterminated quoted string.

Quoted strings are also useful for representing strings that start with a number. For example:

```
'71_user'
```

<number>

A <number> is an integer. It can start with a plus or minus sign and can consist of one or more numbers from 0 to 9. Numbers can be represented either in decimal or hexadecimal format. To represent a number in hexadecimal format, precede the numeric value with the value ’0x’ or ’0X’. For example:

```
-0123456789
0x0000088a
```

<version-data-type>

A <version-data-type> is a software version number with a major and minor version number consisting of one or more numbers from 0 to 9, separated by a decimal point. The major version number is to the left of the decimal point and the minor version number is to the right of the decimal point. The syntax is as follows:

```
n[nnn...].n[nnn...]
```

For example:

```
7.1
6.10
```

Comments

Comments start with two consecutive hyphens (– –) and continue to the next new line. For example:

```
-- This is a comment line.
```

Order of Command Arguments

The order of the command arguments of the management commands is not important. If you enter a command that contains two or more arguments, the arguments do not need to be in the order presented in the format description of that command.

Use of Underscores Between Keywords in Arguments

On the command-line interface, a space can replace the underscore between any keywords in arguments. For example, rather than enter the two keywords NETWORK_PORT (with the underscore separator), you can enter NETWORK PORT (with a space separator) on the command line, and the SQLSRV_MANAGE utility correctly parses these two keywords without returning an error.

SQL Initialization Files

SQL initialization files use the following syntax conventions:

- Leading and trailing white space on a line is ignored.
- Comments start with two consecutive hyphens (– –), must start at the beginning of a line, and continue to the next new line.
- Each SQL initialization statement must be able to be dynamically prepared, executed, and released by the SQL EXECUTE IMMEDIATE statement.
- SQL statements cannot span multiple lines.
- A trailing semicolon (;) at the end of an SQL statement is ignored to allow SQL initialization files to be invoked and verified using interactive SQL.

The following example illustrates a sample SQL initialization file:

```
--
-- This SQL initialization file sets the SQL dialect and default
-- character set for an executor process.
--
SET DIALECT 'SQL89';
SET DEFAULT CHARACTER SET 'KANJI';
```

4.2 How SQLSRV_MANAGE Commands Work

SQLSRV_MANAGE commands work as follows.

Server Configuration Commands

The following commands operate on the server, dispatcher, and service objects in a server configuration:

- ALTER SERVER, CONNECT TO SERVER, CREATE SERVER, DISCONNECT SERVER, DROP SERVER, RESTART SERVER, SET CONFIG_FILE, SET CONNECTION, SHOW SERVER, SHOW SETTINGS, SHOW VERSION, SHUTDOWN SERVER, START SERVER
- ALTER DISPATCHER, CREATE DISPATCHER, DROP DISPATCHER, SHOW DISPATCHER, SHUTDOWN DISPATCHER, START DISPATCHER
- ALTER SERVICE, CREATE SERVICE, DROP SERVICE, GRANT USE ON SERVICE, KILL EXECUTOR, REVOKE USE ON SERVICE, SHOW CLIENTS FOR SERVICE, SHOW SERVICE, SHUTDOWN SERVICE, START SERVICE

Environment Use Commands and Switches

The following commands operate on the SQLSRV_MANAGE system management environment:

- –input and –output switches
- SHOW CONNECTS, SHOW SETTINGS
- CONNECT TO SERVER, DISCONNECT SERVER, SET CONFIG_FILE, SET CONNECTION
- @ , CLOSE, EXIT, HELP, OPEN, SET CONFIRM, SET OUTPUT, SET VERIFY

Table 4–1 describes the three different groups of Oracle SQL / Services objects and shows how each object is acted upon by a set of command verbs.

Table 4–1 Oracle SQL/Services Objects and How Each Object Is Acted Upon by a Command

Object	Command	Description
Dispatcher	ALTER	Change a dispatcher object definition in the configuration file and dynamically change selected attributes for a running server.
	CREATE	Create a dispatcher object for the current server and add the definition to the configuration file.
	DROP	Delete a dispatcher object definition for an inactive dispatcher for the current server from the configuration file.
	SHOW	Show a dispatcher object definition.
	SHUTDOWN	Shut down the specified dispatcher object.
	START	Start a dispatcher process for the defined dispatcher object for the current server.
Server	ALTER	Change a server object definition in the configuration file and dynamically change selected attributes for a running server.
	CONNECT	Connect to a running server.

Table 4–1 (Cont.) Oracle SQL/Services Objects and How Each Object Is Acted Upon by a Command

Object	Command	Description
	CREATE	Create a configuration file and a server object.
	DISCONNECT	Disconnect from a running server.
	DROP	Delete a server object definition and delete the configuration file for an inactive server.
	RESTART	Restart the server including all automatically started dispatchers and services for the current server object.
	SET CONFIG_FILE	Set the current configuration so subsequent commands can modify a server's configuration file.
	SET CONNECTION	Set the connection to the server object with the specified connection name.
	SHOW	Show the server object definition.
	SHOW SETTINGS	Show the current configuration file.
	SHOW VERSION	Show the version of the SQLSRV_ MANAGE management client.
	SHUTDOWN	Shut down the current server object.
	START	Start the server, including all automatically started dispatcher and executor processes for the current server object.
Service	ALTER	Change a service object definition in the configuration file and dynamically change selected attributes for a running service.
	CREATE	Create a service object and add the definition to the configuration file.
	DROP	Delete a service object definition from the configuration file for an inactive service.
	GRANT USE ON	Grant the USE privilege descriptor for a service object to a user name or identifier.

Table 4–1 (Cont.) Oracle SQL/Services Objects and How Each Object Is Acted Upon by a Command

Object	Command	Description
	KILL EXECUTOR	Kill an executor process.
	REVOKE USE ON	Revoke the USE privilege descriptor for a service object from a user name or identifier.
	SHOW CLIENTS	Show the active users of a service.
	SHOW	Show a service object definition including the USE privilege descriptor for a service object for all user names and identifiers.
	SHUTDOWN	Shut down the specified service object.
	START	Start the specified service object.

Table 4–2 describes the SQLSRV_MANAGE environment commands and switches.

Table 4–2 SQLSRV_MANAGE Environment Commands and Switches

Command or Switch	Description
–input switch	Specify the name of an input file from which the SQLSRV_ MANAGE utility reads input.
–output switch	Specify the name of an output file to which the SQLSRV_ MANAGE utility writes output.
@	Run an indirect command file.
CLOSE	Close an output file.
CONNECT TO SERVER	Connect to a running server.
DISCONNECT SERVER	Disconnect from a running server.
EXIT	Exit the SQLSRV_MANAGE utility.
HELP	Get help on a topic.
OPEN	Open an output file.
SET CONFIG_FILE	Set the current configuration so that subsequent commands can modify a server's configuration file.
SET CONFIRM	Require confirmation for certain management operations.

Table 4–2 (Cont.) SQLSRV_MANAGE Environment Commands and Switches

Command or Switch	Description
SET CONNECTION	Change the current connection to a server to another connection from among a group of established connections.
SET OUTPUT	Direct output to the default device when enabled.
SET VERIFY	Echo command file input to the default output device as it is read.
SHOW CONNECTS	Show information about the current server object and all of the active connections that SQLSRV_MANAGE has to servers.
SHOW SETTINGS	Show information about the verify and output settings.
SHOW VERSION	Show the version of the SQLSRV_MANAGE management client.

–input Switch

Specifies the name of the input file from which the SQLSRV_MANAGE utility reads input.

Format

–i[n[put]]	<file-spec>;
	<file-spec> ::=<identifier> or <quoted-string>

Arguments

<file-spec>
The input file name. The file name is expressed either as an identifier or as a quoted string.

Usage Notes

- –i and –in are synonyms for the –input command.
- The SQLSRV_MANAGE utility does not prompt for input, and exits when the specified file is completely read.
- You cannot enter the –input switch at the SQLSRV prompt.

Examples

Example 1: Specify an input file from which the SQLSRV_MANAGE utility reads input.

```
$ sqlsrv_manage -input sqlsrv_create.sqs
```

–output Switch

Specifies the name of the output file to which the SQLSRV_MANAGE utility writes output.

Format

–o[ut[put]]	<file-spec>;
	<file-spec> ::=<identifier> or <quoted-string>

Arguments

<file-spec>
The output file name. The file name is expressed either as an identifier or as a quoted string.

Usage Notes

- –o and –out are synonyms for the –output switch.
- The SQLSRV_MANAGE utility writes all output to the specified file until a CLOSE or OPEN command is executed. If a CLOSE command is issued, subsequent output is sent to standard output. If an OPEN command is issued, output is sent to the new output file.
- You cannot enter the –output switch at the SQLSRV prompt.

Examples

Example 1: Specify an output file to which the SQLSRV_MANAGE utility writes output.

```
$ sqlsrv_manage -output out_testfile
```

@ Command

Runs an indirect command file in the SQLSRV_MANAGE environment.

Format

@	<file-spec>;
	<file-spec> ::=<identifier> or <quoted-string>

Arguments

<file-spec>
The indirect command file name. The file name is expressed as either an identifier or as a quoted string.

Usage Notes

When executed, the indirect command file is opened and input is taken from that file until either a syntax error occurs or there are no more characters in the file.

Examples

Example 1: Run an indirect script named test_file.sqs. Use a quoted string if it is important to preserve case.

```
SQLSRV> @ 'test_file.sqs';
```

ALTER DISPATCHER Command

Changes a dispatcher object definition for the current server only. Changes to a dispatcher definition are stored in the configuration file. Offline dispatcher changes do not affect a running server. Online dispatcher changes affect the running server if the change is to a dynamic attribute; otherwise, the dispatcher must be shut down and started again or the server restarted for dispatcher changes to take effect.

Format

```
ALTER DISPATCHER    <disp-name>
                    –>[ AUTOSTART { ON | OFF} ]
                    –>[ MAX_CONNECTIONS <conn-num> ]
                    –>[ IDLE_USER_TIMEOUT <timeout-num> ]
                    –>[ MAX_CLIENT_BUFFER_SIZE <buf-num> ]
                    –>[ <network-port-spec> ] ...;

                    <disp-name> ::=<identifier>
                    <conn-num> ::=<number>
                    <timeout-num> ::=<number>
                    <buf-num> ::=<number>
                    <network-port-spec> ::=NETWORK_PORT <transport-spec>
                      PROTOCOL <message-protocol>
                    <transport-spec> ::={ <tcp-spec> | <decnet-spec> | <ipxspx-spec>
                      | sqlnet-spec> }
                    <tcp-spec> ::=TCPIP  [ <tcpip-port-spec> ]
                    <tcpip-port-spec> ::=PORT_ID <tcpip-port-num>
                    <tcpip-port-num> ::=<number>
                    <decnet-spec> ::=DECNET  [ <decnet-object-spec> ]
```

```
<decnet-object-spec> ::=[OBJECT { <number>
    | <identifier> | <quoted-string> } ]
<ipxspx-spec> ::=IPXSPX [ <ipxspx-port-spec> ]
<ipxspx-port-spec> ::=PORT_ID <ipxspx-port-num>
<ipxspx-port-num> ::=<number>
<sqlnet-spec> ::=SQLNET <sqlnet-port-spec>
<sqlnet-port-spec> ::=LISTENER_NAME <listener-name>
<listener-name> ::={ <identifier> | <quoted-string> }
<message-protocol> ::={ NATIVE | OCI | SQLSERVICES }
```

Arguments

<disp-name>
The dispatcher name. The dispatcher name is expressed as an identifier.

AUTOSTART {ON | OFF}
Determines whether or not the dispatcher object automatically starts up when you issue a START SERVER or RESTART SERVER command. If the argument is specified as ON, the dispatcher object automatically starts when you issue a START SERVER or RESTART SERVER command. The default is ON.

MAX_CONNECTIONS <conn-num>
Specifies the maximum number of network connections from clients that the dispatcher accepts. The maximum number of connections is expressed as an integer. The default is 100. There is no upper limit other than the operating system configuration, the network configuration, and shared server memory.

IDLE_USER_TIMEOUT <timeout-num>
Specifies the amount of time in seconds that a client (user) can remain idle before the dispatcher disconnects the client. The <timeout-num> value is expressed as an integer. The default value is 0, which displays as none in a SHOW DISPATCHER command and means that the idle timeout value is infinite. A value specified other than 0 is rounded to the next higher multiple of 90 seconds. This is a dynamic attribute that, when changed, takes effect immediately.

MAX_CLIENT_BUFFER_SIZE <buf-num>
Specifies the size of the maximum client buffer size permitted. The maximum allowed client buffer size is 32,000 bytes. If a client application specifies a buffer size

larger than the maximum, then the Oracle SQL/Services client API adjusts the buffer size to the maximum size specified for the dispatcher. The default and minimum value allowed for the MAX_CLIENT_BUFFER_SIZE attribute is 5000 bytes.

<network-port-spec>
Lists network ports that the dispatcher should use for communications with Oracle SQL/Services clients and Oracle ODBC Driver for Rdb clients. The network port specification is any one or any combination of the following: TCP/IP, DECnet, IPX/SPX, and SQL*Net. The default port ID for TCP/IP is 118, the default DECnet object is 81, and the default port ID for IPX/SPX is 33969 (0x84b1). If the network port is not specified, the dispatcher will use the default ports. The maximum number of times that the <network-port-spec> argument can be specified in the ALTER DISPATCHER command is 5. The <network-port-spec> argument can be repeated to include multiple SQL*Net Listener Names.

Also determines the message protocol that each dispatcher network port can support. A dispatcher network port can support only one message protocol. Specify a message protocol that matches the type of client you want a dispatcher network port to support:

- NATIVE

  Oracle RMU clients (Oracle Rdb Performance Monitor, Oracle RMUwin, Parallel Backup Monitor)
- OCI

  Oracle clients using the Oracle Call Interface (OCI) (Oracle Enterprise Manager clients)
- SQLSERVICES

  Oracle SQL/Services clients (Query Performance Tuner, Oracle ODBC Driver for Rdb, Oracle SQL/Services, Oracle Rdb Schema Manager)

Note: Ensure that you have a dispatcher network port defined with a dispatcher message protocol that supports each service API protocol you want to use. For example, if you define a service that supports the OCI API protocol and another service that supports the SQLSERVICES API protocol, you must define at least one dispatcher network port that supports the OCI dispatcher message protocol and the SQL*Net transport and another dispatcher network port that supports the SQLSERVICES dispatcher message protocol and any available transport, respectively.

Usage Notes

- In general, any specified clauses in the ALTER DISPATCHER definition replace the specification of items in the previous server definition. That is, if a clause is specified in the ALTER DISPATCHER command, then the specification of items for that clause is changed in the definition. If no clause is specified, the specification of items remains unchanged for that clause.
- If a network port is altered, the entire network port specification is replaced. Thus, you can add a network port to the existing list with the ALTER DISPATCHER command, but you must respecify all other network port specifications to retain them in the configuration file.
- To use the SQL*Net transport option, specify the SQL*Net transport option as <sqlnet-spec> in the <transport-spec> argument and specify the SQL*Net Listener Name as its <identifier> argument.
- The word LISTENER_NAME is a synonym for the keyword LISTENER.

Examples

Example 1: Dynamically alter the idle user timeout value.

```
SQLSRV> ALTER DISPATCHER tcpip_disp IDLE_USER_TIMEOUT 180;
```

Example 2: Alter a dispatcher to use the SQL*Net protocol. This command removes all other ports for this dispatcher. You must respecify all existing network ports to prevent the loss of previously defined network ports for this dispatcher.

```
SQLSRV> ALTER DISPATCHER sqlnet_disp NETWORK_PORT SQLNET LISTENER_NAME "LISTENER";
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
SQLSRV> SHUTDOWN DISPATCHER sqlnet_disp;
SQLSRV> START DISPATCHER sqlnet_disp;
```

ALTER SERVER Command

Changes a server object definition. Changes to a server definition are stored in the configuration file. Offline server changes do not affect a running server. Online server changes affect the running server if the change is to a dynamic attribute; otherwise, the server must be shut down and started again or restarted for changes to take effect.

Format

```
ALTER SERVER
    –>[ MAX_SHARED_MEMORY_SIZE <mem-size> ]
    –>[ PROCESS_STARTUP_TIMEOUT <process-startup-timeout> ]
    –>[ PROCESS_SHUTDOWN_TIMEOUT <process-shutdown-timeout> ]
    –>[ <network-port-spec> ]...;

    <mem-size> ::=<number>
    <log-path> ::=<quoted-string>
    <dump-path> ::=<quoted-string>
    <lock-path> ::=<quoted-string>
    <shared-memory-path> ::=<quoted-string>
    <process-startup-timeout> ::=<number>
    <process-shutdown-timeout> ::=<number>
    <network-port-spec> ::=NETWORK_PORT <transport-spec>
    <transport-spec> ::={ <tcp-spec> | <decnet-spec> }
    <tcp-spec> ::=TCPIP [ <tcpip-port-spec> ]
    <tcpip-port-spec> ::=PORT_ID <tcpip-port-num>
    <tcpip-port-num> ::=<number>
    <decnet-spec> ::=DECNET [ <decnet-object-spec> ]
    <decnet-object-spec> ::=[ OBJECT {<number> | <identifier>
| <quoted-string> } ]
```

Arguments

MAX_SHARED_MEMORY_SIZE <mem-size>
Specifies the size in kilobytes of the maximum shared memory the server should use. The <mem-size> argument is a number. If the value is changed, that value becomes the maximum shared memory size when the monitor starts up. The default value is 2000 kilobytes or 2 megabytes. Oracle SQL/Services allocates the maximum shared memory size when the monitor starts up. The practical upper limit is dependent on the amount of disk space available.

PROCESS_STARTUP_TIMEOUT <process-startup-timeout>
Specifies the length of time to wait before deciding that a dispatcher or executor process is not going to start up before the monitor takes action and terminates the process. The <process-startup-timeout> argument is a number expressed in seconds. The default value is 0 seconds, which means that no process startup timer value is set. This is a dynamic attribute that, when changed, takes effect immediately. See the Usage Notes for more information.

PROCESS_SHUTDOWN_TIMEOUT <process-shutdown-timeout>
Specifies the length of time to wait before deciding that a dispatcher or executor process is not going to shut down before the monitor takes action and terminates the process. The <process-shutdown-timeout> argument is a number expressed in seconds. The default value is 0 seconds, which means that no process shutdown timer value is set; the process shutdown timer value is infinite. This is a dynamic attribute that, when changed, takes effect immediately. See the Usage Notes for more information.

<network-port-spec>
Lists network ports that the monitor should use for communications with Oracle SQL/Services management clients (SQLSRV_MANAGE and the Oracle SQL/Services Manager GUI). The network port specification is TCP/IP or DECnet. The default port ID for TCP/IP is 2199 and the default DECnet object name is SQLSRV_SERVER. If no network ports are specified, the monitor of the server uses the default ports. The maximum number of times that the <network-port-spec> argument can be specified in the ALTER SERVER command is 5. If a network port is altered, the entire network port specification is replaced.

DECnet or TCP/IP must be available on the node for which the ALTER SERVER definition is used. If none of these are available, then the server will not run.

Usage Notes

- The server definition can be altered online using the CONNECT TO SERVER command or offline if you select its configuration file using the SET CONFIG_FILE command. Online changes for dynamic attributes take effect immediately. When you make an online change of a nondynamic attribute, a status message is returned indicating that you must restart the server to have altered settings take effect. Oracle Corporation recommends that you immediately restart the running server after you complete your management session to ensure the overall consistency of the Oracle SQL/Services server. (To restart the running server, issue the RESTART SERVER command.)
- In general, any specified clauses in the ALTER SERVER definition replace the specification of items in the previous server definition. That is, if a clause is specified in the ALTER SERVER command, then the specification of items for that clause is changed in the definition. If no clause is specified, the specification of items remains unchanged for that clause.
- If you want to set process startup and shutdown timers, follow these guidelines:
  - Usually dispatcher and executor processes start up and shut down in a reasonable period of time. Only during an unusual situation would you need to specify nonzero values for the PROCESS_STARTUP_TIMEOUT and PROCESS_SHUTDOWN_TIMEOUT arguments.
  - In heavily loaded systems, it often takes longer for a particular operation to complete. If either process startup or process shutdown is set to a value other than zero and fails for no apparent reason (you have checked other possible causes and have not isolated the problem), set a higher value for the PROCESS_STARTUP_TIMEOUT argument or the PROCESS_SHUTDOWN_TIMEOUT argument to see if that solves the problem.
- The SQLSRV_MANAGE utility attempts to connect to the monitor of the server using the default TCP/IP or DECnet ports. If you change the network port of the server, you must also specify that port explicitly when connecting from the SQLSRV_MANAGE utility.
- If a network port is altered, the entire network port specification is replaced. Thus, you can add a network port to the existing list with the ALTER SERVER command, but you must respecify all other network port specifications to retain them in the configuration file.
- If the same port ID is specified more than once, an error is returned.

Examples

Example 1: Alter a server definition online.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> ALTER SERVER MAX_SHARED_MEMORY_SIZE 4000;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
SQLSRV> RESTART SERVER;
Disconnected from Server
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
```

Example 2: Alter a server definition offline.

```
SQLSRV> SET CONFIG_FILE ‘my_config_file’;
SQLSRV> ALTER SERVER MAX_SHARED_MEMORY_SIZE 4000;
SQLSRV> RESTART SERVER;
Connecting to server ...
Connected
```

ALTER SERVICE Command

Changes a service object definition for the current server only. Changes to a service definition are stored in the configuration file. Offline service changes do not affect a running server. Online service changes affect the running server if the change is to a dynamic attribute; otherwise, the service must be shut down and started again or the server restarted for service changes to take effect.

Format

```
ALTER SERVICE       <service-name>
                    –> [ PROTOCOL { OCI | RMU | SQLSERVICES } ]
                    –>[ AUTOSTART { ON | OFF } ]
                    –>[ DEFAULT_CONNECT_USERNAME <user-string> ]
                    –>[ REUSE [ SCOPE] [IS] { SESSION | TRANSACTION } ]
                    –>[ SQL_VERSION { <ver-num> | S[TANDARD ] } ]
                    –>[ PROCESS_INITIALIZATION { <proc-init-file-string> | LOGIN } ]
                    –>[ ATTACH <attach-string> ]
                    –>OWNER <user-string>
                    –>[ SCHEMA <schema-string> ]
                    –>[ SQL_INIT_FILE <sql-init-file-string> ]
                    –>[ DATABASE_AUTHORIZATION { [ SERVICE] OWNER
                       | [ CONNECT ] USERNAME } ]
                    –>[ APPLICATION_TRANSACTION_USAGE  { SERIAL | CONCURRENT }]
                    –>[ IDLE_USER_TIMEOUT <timeout-num> ]
                    –>[ IDLE_EXECUTOR_TIMEOUT <timeout> ]
                    –>[ MIN_EXECUTORS <min> ]
                    –>[ MAX_EXECUTORS <max> ]
                    –>[ CLIENTS_PER_EXECUTOR <clients-per-executor> ] ;
```

```
<service-name> ::=<identifier>
<user-string> ::={ <quoted-string> | <identifier> }
<ver-num> ::=<version-data-type>
<proc-init-file-string> ::=<quoted-string>
<attach-string> ::=<quoted-string>
<schema-string> ::=<quoted-string>
<locale-string> ::=<quoted-string>
<rc-file-string> ::=<quoted-string>
<sql-init-file-string> ::=<quoted-string>
<timeout-num> ::=<number>
<timeout> ::=<number>
<min> ::=<number>
<max> ::=<number>
<clients-per-executor> ::=<number>
```

Arguments

<service-name>
The service name. The service name is expressed as an identifier.

PROTOCOL {OCI | RMU | SQLSERVICES}
Determines the application programming interface (API) protocol that each service can support. A service can support only one API protocol. Specify an API protocol that matches the type of client you want a service to support:

- OCI

  Oracle clients using the Oracle Call Interface (OCI) (Oracle Enterprise Manager clients)

- RMU

  Oracle RMU clients (Oracle Rdb Performance Monitor, Oracle RMUwin, Parallel Backup Monitor)

- SQLSERVICES

Oracle SQL / Services clients (Query Performance Tuner, Oracle ODBC Driver for Rdb, Oracle SQL / Services, Oracle Rdb Schema Manager)

Note: Ensure that you have a dispatcher network port defined with a dispatcher message protocol that supports each service API protocol you want to use. For example, if you define a service that supports the OCI API protocol and another service that supports the SQLSERVICES API protocol, you must define at least one dispatcher network port that supports the OCI dispatcher message protocol and the SQL*Net transport and another dispatcher network port that supports the SQLSERVICES dispatcher message protocol and any available transport, respectively.

AUTOSTART {ON | OFF}
Determines whether or not the service object automatically starts up when you issue a START SERVER or RESTART SERVER command. If the argument is specified as ON, the service object automatically starts when you issue a START SERVER or RESTART SERVER command. The default is ON.

DEFAULT_CONNECT_USERNAME <user-string>
The <user-string> argument is either a quoted string or an identifier containing the user name under which unknown users will be allowed to connect to the service. See Section 2.7 and Section 2.8 for more information about using this argument. This is a dynamic attribute that, when changed, takes effect immediately.

REUSE SCOPE IS {SESSION | TRANSACTION}

- SESSION

  An executor for a session reusable service processes requests for one client session at a time. A session begins when a client connects to the service and the connection is bound to an executor process. A session ends when a client disconnects from the service and the connection is unbound from the executor process.

- TRANSACTION

  An executor for a transaction reusable service processes requests for one transaction at a time; however, it supports many concurrent client sessions. A transaction begins when a client issues an SQL statement that either implicitly or explicitly starts a transaction. A transaction ends when a client issues a successful SQL COMMIT or ROLLBACK statement. The REUSE SCOPE IS TRANSACTION argument may be applied only to database services.

See Section 2.6 for more information.

SQL_VERSION {<ver-num>| STANDARD}
Specifies the version of SQL to use for that service. It is expressed as either a version number data type (for example, 7.1) for selecting a version of SQL in an Oracle Rdb multiversion environment or by the keyword STANDARD (or S) for running a standard version of SQL in an Oracle Rdb single version environment. Either value is used as the first parameter argument for the Oracle Rdb RDB$SETVER command procedure when it runs, as described in the installation information. The version number resolves to an "n.n" or "s" parameter argument and the word STANDARD or S resolves to an S parameter argument. When no value is specified, the default is the keyword STANDARD.

PROCESS_INITIALIZATION {<proc-init-file-string> | LOGIN}
The process initialization file can be either a special process initialization file specified as a <quoted-string> or the keyword LOGIN, which resolves to the default login in the user authorization file (UAF). The process initialization or login file is used to help define some of the attributes of the executor process for this service. If no argument is specified, the default is not to run any initialization file.

ATTACH <attach-string>
The SQL ATTACH statement.

If you do not specify an SQL ATTACH statement, you create a universal service that is not preattached to a specific database.

If you specify an SQL ATTACH statement, you create a database service that is preattached to the specified database.

The <attach-string> argument is a single-quoted string and is exactly the same format as the attach-string-literal used in dynamic SQL.

See the *Oracle Rdb7 SQL Reference Manual* for more information on the ATTACH statement.

OWNER <user-string>
Specifies the user name of the owner of the service. Every service has an owner user name. The user name must be specified; otherwise, an error message is returned.

If the service is a database service, then the service owner user name privileges are used for access checks when an executor attaches to the specified database. See Section 2.6 for more information on database services.

If the database access authorization is by the service owner, then the service owner user name privileges are used for all database access operations. See the

DATABASE_AUTHORIZATION argument, later in this argument list, for more information on database access authorization.

Executors are created with the privileges and quotas from the account of the service owner. See Section 2.9.1 for more information.

The <user-string> argument is either a quoted string or an identifier.

SCHEMA <schema-string>
Provides a way to specify the default schema that you want to use when an executor attaches to a multischema database. The default schema name is set (for all service types) as shown in Table 4–3.

Table 4–3 Default Schema Name Used When an Executor Is Bound to a Multischema Database

Schema Name Specified in Service Definition	Database Access Authorization	Default Schema Name Set by Using
Yes	Service owner Connect user name	Name specified in service definition Name specified in service definition
No	Service owner Connect user name	Service owner account name Connect user name (see Section 2.12)

The schema argument allows the default to be overridden. The <schema-string> argument is a quoted string.

SQL_INIT_FILE <sql-init-file-string>
Specifies a file containing SQL statements that tailor the SQL environment for a client connection. For example, you can set the SQL dialect and default character set by using an SQL initialization file. The statements in an SQL initialization file are executed every time a client connects to a service.

The <sql-init-file-string> argument is a quoted string. See Section 4.1 for more information about using an SQL initialization file.

DATABASE_AUTHORIZATION {[SERVICE] OWNER | [CONNECT] USERNAME}
Determines the user name under which access to the database is made. The default is CONNECT USERNAME.

- SERVICE OWNER

For a database service, access to the database is made by using the service owner user name.

- CONNECT USERNAME

  Access to the database is made by using the client-specified user name, the DECnet proxy user name, or the user name specified in the DEFAULT_CONNECT_USERNAME argument.

For more information on database access authorization, see Section 2.7 and Section 2.8.

APPLICATION_TRANSACTION_USAGE {SERIAL | CONCURRENT}
Applies only to transaction reusable database services. Some applications make only a single connection to a service to perform their work, while other applications make multiple connections to the same service. Connections created to transaction reusable database services are tied to the same executor for the life of the session.

If a client application makes multiple connections to a service and these are assigned to the same executor, a deadlock occurs if the client application attempts to start a new transaction on one connection before ending an existing transaction on another connection. When you specify the CONCURRENT keyword, Oracle SQL/Services ensures that multiple connections from the same client application on the same node are never assigned to the same executor process.

When you specify the SERIAL keyword, Oracle SQL/Services assumes that client applications do not start concurrent transactions on multiple connections. Oracle SQL/Services assigns connections to executor processes on a least busy basis (the executor process with the fewest client connections already assigned). Thus, if a client application made more than one connection to the same service and the keyword SERIAL was specified, the second connection may or may not have gone to the same executor process as the first connection, depending on how many connections that executor process was already assigned versus how many connections the other executor processes was assigned for that service.

The default for the APPLICATION_TRANSACTION_USAGE argument is SERIAL. This is a dynamic attribute that, when changed, takes effect immediately.

Some applications, such as Microsoft Access, make multiple connections to the same service to perform their work and require that you specify the CONCURRENT keyword. If set to CONCURRENT, Oracle SQL/Services considers the node, user name, and application name of the client when choosing an executor to which to tie the connection and ensures that multiple connections from the same client application are never assigned to the same executor process.

IDLE_USER_TIMEOUT <timeout-num>
Specifies the amount of time in seconds that a client (user) can remain idle before the server disconnects the client. The <timeout-num> value is expressed as an integer. The default value is 0, which displays as none in a SHOW SERVICE command and means that the idle timeout value is infinite. A specified value other than 0 is rounded to the next higher multiple of 90 seconds. This is a dynamic attribute that, when changed, takes effect immediately.

IDLE_EXECUTOR_TIMEOUT <timeout>
Specifies the amount of time in seconds that an executor process for a session reusable service can remain inactive (not bound to a client connection) before being deleted. The value is expressed as an integer. The default timeout value is 1800 seconds (30 minutes). This is a dynamic attribute that, when changed, takes effect immediately.

MIN_EXECUTORS <min>
Sets the minimum value to which the number of executor processes is allowed to decrease. This is also the number of executor processes started at startup using a START SERVICE or START SERVER command. The value is expressed as an integer. The default minimum number of executors for a session reusable service is 0. A service with MIN_EXECUTORS set to 0 will never show the Starting state when the service starts up. The state will either display as Running or Failed. This is a dynamic attribute that, when changed, takes effect immediately.

If you use transaction reusable executors, you must set the value for the minimum number of executors so that it is equal to the value for the maximum number of executors. The default value is 1 for a transaction reusable service.

MAX_EXECUTORS <max>
Sets the maximum value to which the number of executor processes is allowed to increase. The value is expressed as an integer. The default maximum number of executors is 1. This is a dynamic attribute that, when changed, takes effect immediately.

If you use transaction reusable executors, you must set the value for the minimum number of executors so that it is equal to the value for the maximum number of executors. The default value is 1 for a transaction reusable service.

CLIENTS_PER_EXECUTOR <clients-per-executor>
Specifies the number of clients allowed per executor. The number of clients allowed is dependent upon whether the service is session reusable or transaction reusable. The default number of clients per executor for session reusability is 1 and cannot be greater than 1. The default number of clients per executor for transaction reusability

is 1 but can be greater than 1. The CLIENTS_PER_EXECUTOR value is expressed as an integer. This is a dynamic attribute that, when changed, takes effect immediately.

Usage Notes

- When a service is created, only a privileged user with SYSPRV privilege is authorized to use the service.
- Values specified for parameters in an ALTER SERVICE command replace values defined in the configuration file and in the running server. However, changes to the running server are not immediate and are as described in the following items:
  - If the value for the minimum number of executors for a session reusable service is decreased, the actual number of executor processes does not decrease until individual executors time out using their current timeout settings.
  - If the value for the minimum number of executors for a transaction reusable service is decreased, the actual number of executor processes does not decrease until the service is shut down and started again.
  - If the value for the maximum number of executors is increased, newly created executor processes succeed where they might have previously reached the limit.
  - If the value for the minimum number of executors is increased, new executors are created until the new minimum number of executors is active.
  - If the value for the idle executor timeout parameter is changed, the new idle timeout value is used beginning with the next timeout cycle of a given executor.

Examples

Example 1: Alter a transaction reusable database service online to increase the number of clients per executor to 20 and raise the minimum and maximum number of executors to 10. Because these attributes are dynamic attributes, the service need not be shut down and started up again.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> ALTER SERVICE database4
_SQLSRV> MIN_EXECUTORS 10
_SQLSRV> MAX_EXECUTORS 10
_SQLSRV> CLIENTS_PER_EXECUTOR 20;
```

Example 2: Alter a service online to change the SQL_INIT_FILE attribute. Because this attribute is not a dynamic attribute, the service must be shut down and started up again for the change to take effect.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> ALTER SERVICE database4
_SQLSRV> SQL_INIT_FILE ’sql710’;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take affect
SQLSRV> SHUTDOWN SERVICE database4;
SQLSRV> START SERVICE database4;
```

CLOSE Command

Closes an output file in the SQLSRV_MANAGE environment.

Format

CLOSE;

Usage Notes

Upon closing an output file, output is directed to standard output.

Examples

Example 1: Close an output file.

```
SQLSRV> CLOSE;
```

CONNECT TO SERVER Command

Connects to a server online so that you can begin managing it.

Format

```
CONNECT [TO] SERVER     [ AS <connect-name> ]
                        –>[ USER <user-name> USING <password> ]
                        –>[ NODE <node-name> ]
                        –>[ <network-port-spec> ];

                        <connect-name> ::=<identifier>
                        <user-name> ::={ <quoted-string> | <identifier> }
                        <password> ::={ <quoted-string> | <identifier> }
                        <node-name> ::={ <quoted-string> | <identifier> }
                        <network-port-spec> ::=NETWORK_PORT <transport-spec>
                        <transport-spec> ::={ <tcp-spec> | <decnet-spec> }
                        <tcp-spec> ::=TCPIP [ <tcpip-port-spec> ]
                        <tcpip-port-spec> ::=PORT_ID <tcpip-port-num>
                        <tcpip-port-num> ::=<number>
                        <decnet-spec> ::=DECNET [ <decnet-object-spec> ]
                        <decnet-object-spec> ::=[ OBJECT { <number> | <identifier>
                            | <quoted-string> } ]
```

Arguments

<connect-name>
The connection name. The identifier that uniquely identifies the connection to a server on a particular node. The connection name is most useful when connecting to more than one server at a time. If you are going to manage only one server, a connection name is not needed. Whenever you create a new connection, it becomes the current connection. To switch to a server that you want to manage among those

that you are connected to, use the SET CONNECTION command and specify the connection name of the server.

The connection name is expressed as an identifier.

USER <user-name> USING <password>
Specifies the user name and password of an account that is authorized to manage the server. The user name and password are expressed as either a quoted string or an identifier.

If you are using DECnet or TCP/IP with sufficient privileges to manage a server on the local node, you do not need to enter a user name and password when connecting to the server on a local node. See the Usage Notes for more information on connecting to a server on a local node without specifying a user name and password.

NODE <node-name>
The local host name. By default the node name is the local host name. The node-name is expressed as a quoted string or identifier.

<network-port-spec>
Lists network ports that the monitor should use for communications with Oracle SQL/Services management clients (SQLSRV_MANAGE and the Oracle SQL/Services Manager GUI). The <network-port-spec> argument is TCPIP or DECNET. The network port specification defaults to TCP/IP with a default port ID of 2199. The default DECnet object is named SQLSRV_SERVER.

Usage Notes

- You must either connect to a server before you can begin managing it online or select the configuration file of the server (SET CONFIG_FILE command) to manage it offline.
- When you establish a new connection to a server using the CONNECT TO SERVER command, the new connection becomes the current connection. All subsequent online system management commands operate on the current connection. Use the SET CONNECT command to switch between connections to multiple servers. Use the DISCONNECT command to disconnect from a server.
- A local user can connect to a server using DECnet without specifying a user name or password. You must have either the SYSPRV or BYPASS privilege to omit the user name and password when connecting to a server using TCP/IP.

- If you are connecting to a local server using the configuration file you currently have open, SQLSRV_MANAGE attempts to connect to any network port defined for the server. It tries each network port in a round-robin fashion up to three times each to establish a management connection. The network port used for the management connection is the first one that is successful.

  In a server environment in which multiple servers are running on one or more nodes, you can choose either to manage each server singly as previously mentioned, by connecting to it, performing management tasks, and disconnecting from it, in that sequence, or you can connect to multiple servers at the same time. In the latter case, you must specify a connection name for each server connection and the last connection becomes the current connection. To switch to a server that you wish to manage among those that you are connected to, use the SET CONNECTION command and specify the server's connection name. All subsequent server management commands are performed on that (the current) server.

Examples

Example 1: Connect to a server on the local node as a privileged local user using TCP/IP.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
```

Example 2: Connect to a server (user name and password are quoted strings).

```
SQLSRV> CONNECT SERVER USER 'system' USING 'password';
Connecting to server ...
Connected
```

CREATE DISPATCHER Command

Creates a dispatcher object definition for the current server. The definition is stored in the configuration file. New dispatcher objects must be started online to be part of a running server. Each dispatcher defined must be listening on a unique set of network ports or objects.

Format

```
CREATE DISPATCHER      <disp-name>
                       –>[ AUTOSTART { ON | OFF } ]
                       –>[ MAX_CONNECTIONS <conn-num> ]
                       –>[ IDLE_USER_TIMEOUT <timeout-num> ]
                       –>[ MAX_CLIENT_BUFFER_SIZE <buf-num> ]
                       –>[ <network-port-spec> ] ... ;

                       <disp-name> ::=<identifier>
                       <conn-num> ::=<number>
                       <timeout-num> ::=<number>
                       <buf-num> ::=<number>
                       <network-port-spec> ::=NETWORK_PORT <transport-spec>
                          PROTOCOL <message-protocol>
                       <transport-spec> ::={ <tcp-spec> | <decnet-spec>
                          | <ipxspx-spec> | sqlnet-spec> }
                       <tcp-spec> ::=TCPIP [ <tcpip-port-spec> ]
                       <tcpip-port-spec> ::=PORT_ID <tcpip-port-num>
                       <tcpip-port-num> ::=<number>
                       <decnet-spec> ::=DECNET [ <decnet-object-spec> ]
                       <decnet-object-spec> ::=[OBJECT { <number> | <identifier>
                          | <quoted-string> } ]
```

<ipxspx-spec> ::=IPXSPX [<ipxspx-port-spec>]

<ipxspx-port-spec> ::=PORT_ID <ipxspx-port-num>

<ipxspx-port-num> ::=<number>

<sqlnet-spec> ::=SQLNET <sqlnet-port-spec>

<sqlnet-port-spec> ::=LISTENER_NAME <listener-name>

<listener-name> ::={ <identifier> | <quoted-string> }

<message-protocol> ::={ NATIVE | OCI | SQLSERVICES }

Arguments

<disp-name>
The dispatcher name. The dispatcher name is expressed as an identifier. The dispatcher name must be unique.

AUTOSTART {ON | OFF}
Determines whether or not the dispatcher object automatically starts up when you issue a RESTART SERVER command. If the argument is specified as ON, the dispatcher object automatically starts when you issue a RESTART SERVER command. The default is ON.

MAX_CONNECTIONS <conn-num>
Specifies the maximum number of network connections from clients that the dispatcher will accept. The maximum number of connections is expressed as an integer. The default is 100. There is no upper limit other than the operating system configuration, the network configuration, and shared server memory.

IDLE_USER_TIMEOUT <timeout-num>
Specifies the amount of time in seconds that a client (user) can remain idle before the dispatcher disconnects the client. The <timeout-num> value is expressed as an integer. The default value is 0, which displays as none in a SHOW DISPATCHER command and means that the idle timeout value is infinite. A specified value other than 0 is rounded to the next higher multiple of 90 seconds.

MAX_CLIENT_BUFFER_SIZE <buf-num>
Specifies the maximum client buffer size permitted. The maximum allowed client buffer size is 32,000 bytes. If a client application specifies a buffer size larger than the maximum, then the Oracle SQL/Services client API adjusts the buffer size to the maximum size specified for the dispatcher. The default and minimum value allowed for the MAX_CLIENT_BUFFER_SIZE attribute is 5000 bytes.

<network-port-spec>
Lists network ports that the dispatcher should use for communications with Oracle SQL/Services clients, Oracle ODBC Driver for Rdb clients, and OCI clients. The network port specification is any one or any combination of the following: TCP/IP, DECnet, IPX/SPX, and SQL*Net. The default port ID for TCP/IP is 118, the default DECnet object is 81, and the default port ID for IPX/SPX is 33969 (0x84b1). If no network port is specified, the dispatcher uses the default ports. The <network-port-spec> argument can be repeated to include multiple SQL*Net Listener Names. The maximum number of times that the <network-port-spec> argument can be specified in the CREATE DISPATCHER command is 5.

Also determines the message protocol that each dispatcher network port can support. A dispatcher network port can support only one message protocol. Specify a message protocol that matches the type of client you want a dispatcher network port to support:

- NATIVE

  Oracle RMU clients (Rdb Performance Monitor, RMUwin, Parallel Backup Monitor, Backup Manager, Instance Manager, Storage Manager, SQL Worksheet)

- OCI

  Oracle clients using the Oracle Call Interface (OCI) (Oracle Enterprise Manager clients)

- SQLSERVICES

  Oracle SQL/Services clients (Query Performance Tuner, Oracle ODBC Driver for Rdb, Oracle SQL/Services, Schema Manager)

Note: Ensure that you have a dispatcher network port defined with a dispatcher message protocol that supports each service API protocol you want to use. For example, if you define a service that supports the OCI API protocol and another service that supports the SQLSERVICES API protocol, you must define at least one dispatcher network port that supports the OCI dispatcher message protocol and the SQL*Net transport and another dispatcher network port that supports the SQLSERVICES dispatcher message protocol and any available transport, respectively.

Usage Notes

- To use the SQL*Net transport option, specify the SQL*Net transport option as <sqlnet-spec> in the <transport-spec> argument and specify the SQL*Net Listener Name as its <identifier> argument.
- The word LISTENER is a synonym for the keyword LISTENER_NAME.
- SQLSRV_MANAGE lets you create two or more dispatchers listening on the same port ID or object, but only the first dispatcher with a unique port ID or object is allowed to start. If you attempt to start a second dispatcher listening on the same port ID or object, it fails to start if it cannot listen on any of the specified network ports.

Examples

Example 1: Create a dispatcher that uses the TCP/IP protocol.

```
SQLSRV> CREATE DISPATCHER tcpip_disp NETWORK_PORT TCPIP;
SQLSRV> START DISPATCHER tcpip_disp;
```

Example 2: Create a dispatcher that uses the SQL*Net protocol.

```
SQLSRV> CREATE DISPATCHER sqlnet_disp
_SQLSRV> NETWORK_PORT SQLNET LISTENER_NAME LISTENER;
SQLSRV> START DISPATCHER sqlnet_disp;
```

CREATE SERVER Command

Creates the server object definition and the configuration file. The definition is stored in the configuration file. The new server must be started offline.

Format

```
CREATE SERVER
                  –>[ MAX_SHARED_MEMORY_SIZE <mem-size> ]
                  –>[ PROCESS_STARTUP_TIMEOUT <process-startup-timeout> ]
                  –>[ PROCESS_SHUTDOWN_TIMEOUT <process-shutdown-timeout> ]
                  –>[ <network-port-spec> ] ... ;

                  <mem-size> ::=<number>
                  <process-startup-timeout> ::=<number>
                  <process-shutdown-timeout> ::=<number>
                  <network-port-spec> ::=NETWORK_PORT <transport-spec>
                  <transport-spec> ::={ <tcp-spec> | <decnet-spec> }
                  <tcp-spec> ::=TCPIP [ <tcpip-port-spec> ]
                  <tcpip-port-spec> ::=PORT_ID <tcpip-port-num>
                  <tcpip-port-num> ::=<number>
                  <decnet-spec> ::=DECNET [ <decnet-object-spec> ]
                  <decnet-object-spec> ::=[OBJECT
                     { <number> | <identifier> | <quoted-string> } ]
```

Arguments

MAX_SHARED_MEMORY_SIZE <mem-size>
Sets the size in kilobytes of the maximum shared memory that the server should use. The <mem-size> argument is a number. The default is 2000 kilobytes (2 megabytes). The Oracle SQL/Services V7.0 and higher server allocates the

maximum shared memory size when the monitor starts up. The practical upper limit is dependent on the amount of disk space available.

PROCESS_STARTUP_TIMEOUT <process-startup-timeout>
Specifies the length of time to wait before deciding that a dispatcher or executor process is not going to start up before the monitor takes action and terminates the process. The <process-startup-timeout> argument is a number expressed in seconds. The default value is 0 seconds, which means that no process startup timer value is set. See the Usage Notes for more information.

PROCESS_SHUTDOWN_TIMEOUT <process-shutdown-timeout>
Specifies the length of time to wait before deciding that a dispatcher or executor process is not going to shut down before the monitor takes action and terminates the process. The <process-shutdown-timeout> argument is a number expressed in seconds. The default value is 0 seconds, which means that no process shutdown timer value is set; the process shutdown timer value is infinite. See the Usage Notes for more information.

<network-port-spec>
Lists network ports that the monitor should use for communications with Oracle SQL/Services management clients (SQLSRV_MANAGE and the Oracle SQL/Services Manager GUI). The network port specification is TCP/IP or DECnet. The default port ID for TCP/IP is 2199 and the default DECnet object name is SQLSRV_SERVER. If no network ports are specified, the monitor of the server uses the default ports. The maximum number of times that the <network-port-spec> argument can be specified in the CREATE SERVER command is 5.

DECnet or TCP/IP must be available on the node for which the create server definition is defined. If none of these are available, the server will not start.

Usage Notes

- The CREATE SERVER command is typically used only during an Oracle SQL/Services installation. The installation procedure uses the SQLSRV_CREATE71.COM procedure to create a configuration file containing a server and a default set of dispatchers and services, and to start the server.

  If you accidentally delete the configuration file or if the file becomes corrupted, you need to re-create the server if you do not have a backup. First, delete the original configuration file if it still exists. However, be sure to retain a copy of the file if it was corrupted by an Oracle SQL/Services component, so you can submit it with a software problem report. See Section 3.6 for information on how to report a software problem. There are two ways to re-create the server.

- Run the SQLSRV_CREATE71.COM procedure.

  Execute the SYS$MANAGER:SQLSRV_CREATE71.COM command procedure, which re-creates the server using the SYS$MANAGER:SQLSRV_CREATE71.SQS SQLSRV_MANAGE script.

  Note: This is the recommended method of re-creating a server and is the only supported method of re-creating the Oracle RMU dispatcher and Oracle RMU service objects.

- Issue the SET CONFIG_FILE command and specify a configuration file specification that does not exist. When you do this, you are prompted if you want to create one now; answer YES. The default is NO. If the SET CONFIRM command is set to OFF, then you are not prompted. A SHOW SETTINGS command displays the current settings and the file specification for this new configuration file. Issue a CREATE SERVER command to create a server using this configuration file.

■ If the configuration file already exists and you issue a CREATE SERVER command, an error message displays and the CREATE SERVER command fails.

■ The SQLSRV_MANAGE utility attempts to connect to the monitor of the server using the default TCP/IP or DECnet ports. If you change the network port of the server, you must also specify that port explicitly when connecting from the SQLSRV_MANAGE utility.

■ If you want to set process startup and shutdown timers, follow these guidelines:

- Usually dispatcher and executor processes start up and shut down in a reasonable period of time. Only during an unusual situation would you need to specify nonzero values for the PROCESS_STARTUP_TIMEOUT and PROCESS_SHUTDOWN_TIMEOUT arguments.
- In heavily loaded systems, it often takes longer for a particular operation to complete. If either process startup or process shutdown is set to a value other than zero and fails for no apparent reason (you have checked other possible causes and have not isolated the problem), set a higher value for the PROCESS_STARTUP_TIMEOUT argument or the PROCESS_SHUTDOWN_TIMEOUT argument to see if that solves the problem.

Examples

Example 1: Create a server definition for a local node on which there is currently no Oracle SQL / Services server.

```
SQLSRV> SET CONFIG_FILE 'my_config_file';
SQLSRV> CREATE SERVER MAX_SHARED_MEMORY_SIZE 3000;
SQLSRV> START SERVER;
Server started
Connecting to server ...
Connected
```

CREATE SERVICE Command

Creates a service object definition for the current server only. The definition is stored in the configuration file. New service objects must be started online to be part of a running server.

Format

```
CREATE SERVICE     <service-name>
                   –>[ PROTOCOL { OCI | RMU | SQLSERVICES } ]
                   –>[ AUTOSTART { ON | OFF } ]
                   –>[ DEFAULT_CONNECT_USERNAME <user-string> ]
                   –>[ REUSE [ SCOPE ] [ IS ] { SESSION | TRANSACTION } ]
                   –>[ SQL_VERSION { <ver-num> | S[TANDARD ] } ]
                   –>[ PROCESS_INITIALIZATION { <proc-init-file-string> | LOGIN } ]
                   –>[ ATTACH <attach-string> ]
                   –>[ OWNER <user-string> ]
                   –>[ SCHEMA <schema-string> ]
                   –>[ SQL_INIT_FILE <sql-init-file-string> ]
                   –>[ DATABASE_AUTHORIZATION { [ SERVICE ] OWNER
                       | [ CONNECT ] USERNAME } ]
                   –>[ APPLICATION_TRANSACTION_USAGE
                       { SERIAL | CONCURRENT } ]
                   –>[ IDLE_USER_TIMEOUT <timeout-num> ]
                   –>[ IDLE_EXECUTOR_TIMEOUT <timeout> ]
                   –>[ MIN_EXECUTORS <min> ]
                   –>[ MAX_EXECUTORS <max> ]
                   –>[ CLIENTS_PER_EXECUTOR <clients-per-executor> ] ;

                   <service-name> ::=<identifier>
```

```
<user-string> ::={ <quoted-string> | <identifier> }
<ver-num> ::=<version-data-type>
<proc-init-file-string> ::=<quoted-string>
<attach-string> ::=<quoted-string>
<schema-string> ::=<quoted-string>
<locale-string> ::=<quoted-string>
<rc-file-string> ::=<quoted-string>
<sql-init-file-string> ::=<quoted-string>
<timeout-num> ::=<number>
<timeout> ::=<number>
<min> ::=<number>
<max> ::=<number>
<clients-per-executor> ::=<number>
```

Arguments

<service-name>
The service name. The service name is expressed as an identifier. The service name must be unique.

PROTOCOL {OCI | RMU | SQLSERVICES}
Determines the application programming interface (API) protocol that each service can support. A service can support only one API protocol. Specify an API protocol that matches the type of client you want a service to support:

- OCI

  Oracle clients using the Oracle Call Interface (OCI) (Oracle Enterprise Manager clients)

- RMU

  Oracle RMU clients (Rdb Performance Monitor, RMUwin, Parallel Backup Monitor, Backup Manager, Instance Manager, Storage Manager, SQL Worksheet)

- SQLSERVICES

Oracle SQL/Services clients (Query Performance Tuner, Oracle ODBC Driver for Rdb, Oracle SQL/Services, Schema Manager)

Note: Ensure that you have a dispatcher network port defined with a dispatcher message protocol that supports each service API protocol you want to use. For example, if you define a service that supports the OCI API protocol and another service that supports the SQLSERVICES API protocol, you must define at least one dispatcher network port that supports the OCI dispatcher message protocol and the SQL*Net transport and another dispatcher network port that supports the SQLSERVICES dispatcher message protocol and any available transport, respectively.

AUTOSTART {ON | OFF}
Determines whether or not the service object automatically starts up when you issue a RESTART SERVER command. If the argument is specified as ON, the service object automatically starts when you issue a START SERVER or RESTART SERVER command. The default is ON.

DEFAULT_CONNECT_USERNAME <user-string>
The <user-string> argument is either a quoted string or an identifier containing the user name under which unknown users are allowed to connect to the service. See Section 2.7 and Section 2.8 for more information about using this argument.

REUSE SCOPE IS {SESSION | TRANSACTION}

- SESSION

  An executor for a session reusable service processes requests for one client session at a time. A session begins when a client connects to the service and the connection is bound to an executor process. A session ends when a client disconnects from the service and the connection is unbound from the executor process.

- TRANSACTION

  An executor for a transaction reusable service processes requests for one transaction at a time; however, it supports many concurrent client sessions. A transaction begins when a client issues an SQL statement that either implicitly or explicitly starts a transaction. A transaction ends when a client issues a successful SQL COMMIT or ROLLBACK statement. The REUSE SCOPE IS TRANSACTION argument can be applied only to database services.

See Section 2.6 for more information.

SQL_VERSION {<ver-num> | STANDARD}
Specifies the version of SQL to use for that service. It is expressed as either a version number data type (for example, 7.1) for selecting a version of SQL in an Oracle Rdb multiversion environment or by the keyword STANDARD (or S) for running a standard version of SQL in an Oracle Rdb single version environment. Either value is used as the first parameter argument for the Oracle Rdb RDB$SETVER command procedure when it runs, as described in the installation information. The version number resolves to an "n.n" or "s" parameter argument and the word STANDARD or S resolves to an S parameter argument. When no value is specified, the default is to use the keyword STANDARD.

PROCESS_INITIALIZATION {<proc-init-file-string> | LOGIN}
The process initialization file can be either a special process initialization file specified as a <quoted-string> or the keyword LOGIN, which resolves to the default login in the user authorization file (UAF). The process initialization or login file is used to help define some of the attributes of the executor process for this service. If no argument is specified, the default is not to run any initialization file.

ATTACH <attach-string>
The SQL ATTACH statement.

If you do not specify an SQL ATTACH statement, you create a universal service that is not preattached to a specific database.

If you do specify an SQL ATTACH statement, you create a database service that is preattached to the specified database.

The <attach-string> argument is a single-quoted string and is exactly the same format as the attach-string-literal used in dynamic SQL.

See the *Oracle Rdb7 SQL Reference Manual* for more information on the ATTACH statement.

OWNER <user-string>
Specifies the user name of the owner of the service. Every service has an owner user name. The user name must be specified; otherwise, an error message is returned.

If the service is a database service, then the service owner's user name privileges are used for access checks when an executor attaches to the specified database. See Section 2.6 for more information on database services.

If database access authorization is by service owner, then the service owner user name privileges are used for all database access operations. See the DATABASE_

AUTHORIZATION argument for more information on database access authorization.

Executors are created with the privileges and quotas from the service owner's account. See Section 2.9.1 for more information.

The <user-string> is a quoted string or an identifier.

SCHEMA <schema-string>
Provides a way to specify the default schema that you want to use when an executor attaches to a multischema database. The default schema name is set (for all service types) as shown in Table 4–4.

Table 4–4 Default Schema Name Used When an Executor Is Bound to a Multischema Database

Schema Name Specified in Service Definition	Database Access Authorization	Default Schema Name Set by Using
Yes	Service owner Connect user name	Name specified in service definition Name specified in service definition
No	Service owner Connect user name	Service owner account name Connect user name (see Section 2.12)

The schema argument allows the default to be overridden. The <schema-string> argument is a quoted string.

SQL_INIT_FILE <sql-init-file-string>
Specifies a file containing SQL statements that tailor the SQL environment for a client connection. For example, you can set the SQL dialect and default character set by using an SQL initialization file. The statements in an SQL initialization file are executed every time a client connects to a service.

The <sql-init-file-string> argument is a quoted string. See Section 4.1 for more information about using an SQL initialization file.

DATABASE_AUTHORIZATION {[SERVICE] OWNER | [CONNECT] USERNAME}
Determines the user name under which access to the database is made. The default is CONNECT USERNAME.

- SERVICE OWNER

  For a database service, access to the database is made by using the service owner user name.

- CONNECT USERNAME

  Access to the database is made by using the client-specified user name, the DECnet proxy user name, or the user name specified in the DEFAULT_ CONNECT_USERNAME argument.

For more information on database access authorization, see Section 2.7 and Section 2.8.

APPLICATION_TRANSACTION_USAGE {SERIAL | CONCURRENT}
The APPLICATION_TRANSACTION_USAGE argument is applicable only to transaction reusable database services. Some applications make only a single connection to a service to perform their work, while other applications make multiple connections to the same service. Connections created to transaction reusable database services are tied to the same executor for the life of the session.

If a client application makes multiple connections to a service and these are assigned to the same executor, a deadlock occurs if the client application attempts to start a new transaction on one connection before ending an existing transaction on another connection. When you specify the CONCURRENT keyword, Oracle SQL/Services ensures that multiple connections from the same client application on the same node are never assigned to the same executor process.

When you specify the SERIAL keyword, Oracle SQL/Services assumes that client applications do not start concurrent transactions on multiple connections. Oracle SQL/Services assigns connections to executor processes on a least busy basis (the executor process with the fewest client connections already assigned). Thus, if a client application made more than one connection to the same service and the keyword SERIAL was specified, the second connection may or may not have gone to the same executor process as the first connection, depending on how many connections that executor process was already assigned versus how many connections the other executor processes was assigned for that service.

The default for the APPLICATION_TRANSACTION_USAGE argument is SERIAL.

Some applications, such as Microsoft Access, make multiple connections to the same service to perform their work and require that you specify the CONCURRENT keyword. If set to CONCURRENT, Oracle SQL/Services considers the node, user name, and application name of the client when choosing an executor to which to tie the connection and ensures that multiple connections from the same client application are never assigned to the same executor process.

IDLE_USER_TIMEOUT <timeout-num>
Specifies the amount of specified time in seconds that a client (user) can remain idle before the server disconnects the client. The <timeout-num> value is expressed as

an integer. The default value is 0, which displays as none in a SHOW SERVICE command and means that the idle timeout value is infinite. A specified value other than 0 is rounded to the next higher multiple of 90 seconds.

IDLE_EXECUTOR_TIMEOUT <timeout>
Specifies the amount of time in seconds that an executor process for a session reusable service can remain inactive (not bound to a client connect) before being deleted. The value is expressed as an integer. The default timeout value is 1800 seconds (30 minutes).

MIN_EXECUTORS <min>
Sets the minimum value to which the number of executor processes is allowed to decrease. This is also the number of executor processes started at startup using a START SERVICE or START SERVER command. The value is expressed as an integer. The default minimum number of executors for a session reusable service is 0. A service with MIN_EXECUTORS set to 0 never shows the Starting state when the service starts up. The state displays as either Running or Failed.

If you use transaction reusable executors, you must set the value for the minimum number of executors so that it is equal to the value for the maximum number of executors. The default value is 1 for a transaction reusable service.

MAX_EXECUTORS <max>
Sets the maximum value to which the number of executor processes is allowed to increase. The value is expressed as an integer. The default maximum number of executors is 1.

If you use transaction reusable executors, you must set the value for the minimum number of executors so that it is equal to the value for the maximum number of executors. The default value is 1 for a transaction reusable service.

CLIENTS_PER_EXECUTOR <clients-per-executor>
Specifies the number of clients allowed per executor. The number of clients allowed is dependent upon whether the service is session reusable or transaction reusable. The default number of clients per executor for session reusability is 1 and cannot be greater than 1. The default number of clients per executor for transaction reusability is 1 but can be greater than 1. The CLIENTS_PER_EXECUTOR value is expressed as an integer.

Usage Notes

- When a service is created, only a privileged user with SYSPRV privilege is authorized to use the service.

- If you use the default minimum number of 0 executors, the default maximum number of executors is 1. If the minimum number of executors defined is greater than 0, the default maximum number of executors *equals* the defined minimum value. For example, if the defined minimum number of executors is 5, the default maximum number of executors is also 5.

Examples

Example 1: Create a universal service named V71.

```
SQLSRV> CREATE SERVICE V71 OWNER 'SQLSRV$DEFLT' SQL VERSION 7.1
_SQLSRV> MIN_EXECUTORS 5
_SQLSRV> MAX_EXECUTORS 10;
SQLSRV> START SERVICE V71;
```

DISCONNECT SERVER Command

Disconnects from a connection to a server.

Format

```
DISCONNECT SERVER        [ <connect-name> ] ;

                         <connect-name> ::=<identifier>
```

Arguments

<connect-name>
The connection name. This identifier uniquely identifies the connection to a server on a particular node. The connection name is expressed as an identifier.

Usage Notes

The DISCONNECT SERVER command works in the opposite way as the CONNECT TO SERVER command. It disconnects the named connection if a connection name is specified or disconnects the current connection if no connection name is specified.

Examples

Example 1: Disconnect from the server whose connection name is eagle.

```
SQLSRV> CONNECT TO SERVER AS eagle;
Connecting to server ...
Connected
SQLSRV> DISCONNECT SERVER eagle;
```

DROP Command

Deletes the specified object for the current server.

Format

```
DROP                <obj-type><obj-name>;

                    <obj-type> ::=DISPATCHER | SERVICE
                    <obj-name> ::= <identifier>
```

Arguments

<obj-type>
Specifies dispatcher or service using the keyword DISPATCHER or SERVICE object type, respectively.

<obj-name>
The name of the object to be deleted. The object name is expressed as an identifier.

Usage Notes

- For online deletions, the object to be deleted cannot be currently active or running; that is, the object must first be shut down online. You may want to issue a SHOW CLIENTS command to determine if there are any client applications using the service you are going to shut down and delete and to ensure that no clients are connected to a service using a network transport provided by the dispatcher that you are going to shut down and delete.
- The SQLSRV_MANAGE utility does not prevent you from deleting a dispatcher or service object online while the dispatcher or service is running on a different node in an environment where two or more nodes share the same configuration file. If this happens, then the SQLSRV_MANAGE utility displays a warning message if you show the dispatcher or service that has been deleted but is still running. For example, if you delete a service, the following message displays for the deleted service when you issue a SHOW SERVICE command.

```
**********************************************************
** This Service has been deleted from the config file.  **
** It will not exist after it is shut down.             **
**********************************************************
```

- Oracle Corporation recommends that you do not make offline modifications to a configuration file if there is a server running that is using the same file. In this situation, the SQLSRV_MANAGE utility, for example, does not prevent you from deleting a dispatcher or service object offline while the dispatcher or service is running.

  A client application using a service or dispatcher that has been deleted offline, continues to have use of that object until it disconnects from the server object. However, once the client application disconnects from the server, it cannot reconnect to the dispatcher or service that was deleted. Before the object that was deleted is shut down, a SHOW command displays a message for the deleted object as shown in the previous list item.
- The DROP command removes the specified object from the configuration file.
- Removing a service implicitly deletes the USE privilege descriptors granted to users for that service.

Examples

Example 1: Delete the database_3 service object.

```
SQLSRV> SHUTDOWN SERVICE database_3;
SQLSRV> DROP SERVICE database_3;
```

Example 2: Delete the disp_tcpip dispatcher object.

```
SQLSRV> SHUTDOWN DISPATCHER disp_tcpip;
SQLSRV> DROP DISPATCHER disp_tcpip;
```

DROP SERVER Command

Deletes the current server, including the configuration file.

Format

DROP SERVER;

Usage Notes

- The server to be deleted cannot currently be active; it must first be shut down online and then deleted offline.
- The DROP SERVER command is an offline operation; you cannot be connected to the server.
- The DROP SERVER command deletes the configuration file.

Examples

Example 1: Delete the current server object.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHUTDOWN SERVER;
SQLSRV> DISCONNECT SERVER;
SQLSRV> SET CONFIG_FILE 'my_config_file';
SQLSRV> DROP SERVER;
Configuration file will be deleted, continue? (N) y
Configuration file deleted
```

EXIT Command

Exits the SQLSRV_MANAGE environment.

Format

<u>EXIT</u>[;]

Usage Notes

- You can exit the SQLSRV_MANAGE environment or utility in the following two ways:
  - Using the EXIT command
  - When an end-of-file is encountered on the last input source

    If you are using the SQLSRV_MANAGE utility interactively, you can enter Ctrl/Z to exit the SQLSRV_MANAGE utility.

    If you specify an input file on the command line with the –input file switch, and the file is the last input source, and an end-of-file is reached, SQLSRV_MANAGE exits.
- Use of the terminating semicolon (;) is optional.

Examples

Example 1: Exit the SQLSRV_MANAGE environment.

```
SQLSRV> EXIT
```

GRANT USE ON SERVICE Command

Grant the USE privilege descriptor for a service to a user. You can grant USE to an identifier and permit access to the specified service to a user who holds that specific identifier.

Format

```
GRANT USE ON SERVICE          <service-name-list> TO <grant-element-list> ;

                              <service-name-list> ::=<service-name> [ , <service-name> ] ...
                              <service-name> ::=<identifier>
                              <grant-element-list> ::=<grant-element> [ , <grant-element> ] ...
                              <grant-element> ::={ PUBLIC|PRIVILEGED_USER
                                  | [ USER[S] ] <user-name>
                                  | IDENTIFIER[S] <identifier-name>
                                  | GROUP[S] <group-name> }
                              <user-name> ::={ <quoted-string> | <identifier> }
                              <identifier-name> ::={ <quoted-string> | <identifier> }
                              <group-name> ::={ <quoted-string> | <identifier> }
```

Arguments

<service-name-list>
Lists service names on which the GRANT USE ON SERVICE command operates. The service name is expressed as an identifier.

<grant-element-list>
Lists grant elements on which the GRANT USE ON SERVICE command acts. A grant list element can be the keyword PUBLIC or PRIVILEGED_USER, a list of user names, or a list of identifier names. A PRIVILEGED_USER is defined as a user with SYSPRV privilege (either default or granted privilege). A user name is expressed as either a quoted string or an identifier. An identifier name is expressed as either a quoted string or an identifier.

Usage Notes

- For V7.0 and higher, Oracle SQL / Services grants a single privilege descriptor, USE.
- Granting a new user the USE privilege descriptor takes effect upon the user's next attempt to use Oracle SQL / Services *after* the privilege change is complete. For example, a new user, once granted the USE privilege descriptor, can use Oracle SQL / Services on the next attempt.
- If you use the keyword IDENTIFIER[S], the specified identifier is added to the list of granted identifiers and permits a user who holds that specific identifier to access the specified service. If the IDENTIFIER keyword is omitted, then the specified user name is granted access to use the service.

Examples

Example 1: Grant the USE privilege descriptor for the general service to PUBLIC.

```
SQLSRV> GRANT USE ON SERVICE general TO PUBLIC;
```

Example 2: Grant the USE privilege descriptor for the database_2 service to fred and wilma.

```
SQLSRV> GRANT USE ON SERVICE database_2 TO fred,wilma;
```

Example 3: Grant the USE privilege descriptor for the system management SQLSRV_MANAGE service to fred and wilma.

```
SQLSRV> GRANT USE ON SERVICE sqlsrv_manage TO fred,wilma;
```

Example 4: Grant the USE privilege descriptor for the system management SQLSRV_MANAGE service to the identifiers payroll_dba and operator.

```
SQLSRV> GRANT USE ON SERVICE sqlsrv_manage
_SQLSRV> TO IDENTIFIERS payroll_dba,operator;
```

HELP Command

Gets help on a topic within the SQLSRV_MANAGE environment.

Format

```
HELP          [ <help-keyword> ] ... [ ; ]

              <help-keyword> ::=<identifier>
```

Arguments

<help-keyword>
A help keyword. The help keyword is expressed as an identifier.

Usage Notes

Use of the terminating semicolon (;) is optional.

Examples

Example 1: Get help on a topic within the SQLSRV_MANAGE environment.

```
SQLSRV> HELP
```

KILL EXECUTOR Command

Kills the specified executor.

Format

```
KILL EXECUTOR        { PID <process-id> | <executor-name> } ;

                     <process-id> ::=<number>
                     <executor-name> ::=<identifier>
```

Arguments

{PID <process-id> | <executor-name>}
The process ID or executor name. The process ID is expressed as an integer and can be represented either in decimal or hexadecimal format. The executor name is expressed as an identifier. To determine the executor name, perform a SHOW CLIENTS FULL command.

Usage Notes

- The process ID can be represented in either decimal or hexadecimal format. To represent a process ID in hexadecimal format, precede the process ID value with the value '0x' or '0X' (for example, 0x0000088a).
- You can kill an executor only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to kill an executor running on that server.

Examples

Example 1: Kill an executor by process ID (represented in hexadecimal format).

```
SQLSRV> KILL EXECUTOR PID 0x0000072a;
```

Example 2: Kill an executor by process ID (represented in decimal format).

```
SQLSRV> KILL EXECUTOR PID 324693;
```

Example 3: Kill an executor by name.

```
SQLSRV> KILL EXECUTOR generi004000280;
```

OPEN Command

Opens an output file in the SQLSRV_MANAGE environment. Subsequent output by SQLSRV_MANAGE, including error messages, is written to this file.

Format

```
OPEN            <file-spec>;

                <file-spec> ::={ <identifier> | <quoted-string> }
```

Arguments

<file-spec>
The output file name. The file name is expressed either as an identifier or as a quoted string.

Usage Notes

The OPEN command creates the specified file and writes all subsequent output to that file. If you enter the OPEN command, the OPEN command does an implicit close of the current output file if an output file was already open.

Examples

Example 1: Open an output file.

```
SQLSRV> OPEN test_file;
```

RESTART SERVER Command

Restarts the current server.

Format

RESTART SERVER [AUTOSTART { ON | OFF }] ;

Arguments

AUTOSTART {ON | OFF}
Determines whether or not other server objects (dispatchers and services) automatically start up again when you issue a RESTART SERVER command. ON is the default. If the argument is specified as ON, other server objects automatically restart (shut down and start again) if each object's AUTOSTART argument value is also set as ON. If you do not want to restart other server objects, specify the AUTOSTART attribute value as OFF in the RESTART SERVER command. The AUTOSTART OFF attribute setting overrides each object's AUTOSTART attribute setting and allows you to individually start each object after restarting just the server object.

Usage Notes

- You can restart a server only as an online operation; that is, you must be connected to the server (CONNECT TO SERVER command) to restart it.
- Use the RESTART SERVER command to restart the server. By default, all server components (dispatchers and services) for the current server will also restart unless these server objects have the AUTOSTART argument specified as OFF in their definitions.

Examples

Example 1: Restart the current server.

```
SQLSRV> CONNECT TO SERVER;
Connecting to server ...
Connected
SQLSRV> ALTER SERVER MAX_SHARED_MEMORY_SIZE 4000;
%DBS-S-ALTER_RESTART, Restart object to have altered settings take effect
SQLSRV> RESTART SERVER;
```

```
Disconnected from Server
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
```

REVOKE USE ON SERVICE Command

Revoke the USE privilege descriptor for a service from a user. You can revoke USE from an identifier to remove access to the specified service by users who hold that identifier.

Format

```
REVOKE USE ON SERVICE     <service-name-list> FROM <grant-element-list> ;


                          <service-name-list> ::=<service-name> [ ,<service-name> ] ...
                          <service-name> ::=<identifier>
                          <grant-element-list> ::=<grant-element> [ , <grant-element> ] ...
                          <grant-element> ::={ PUBLIC | PRIVILEGED_USER
                            | [USER[S] ] <user-name>
                            | IDENTIFIER[S] <identifier-name>
                            | GROUP[S] <group-name> }
                          <user-name> ::={ <quoted-string> | <identifier> }
                          <identifier-name> ::={ <quoted-string> | <identifier> }
                          <group-name> ::={ <quoted-string> | <identifier> }
```

Arguments

<service-name-list>
Lists service names on which the REVOKE USE ON SERVICE command operates. The service name is expressed as an identifier.

<grant-element-list>
Lists grant elements on which the REVOKE USE ON SERVICE command acts. A grant list element can be the keyword PUBLIC or PRIVILEGED_USER, a list of user names, or a list of identifier names. A PRIVILEGED_USER is defined as a user with SYSPRV privilege (either default or granted privilege). A user name is expressed as either a quoted string or an identifier. An identifier name is expressed as either a quoted string or an identifier.

Usage Notes

- For V7.0 and higher, Oracle SQL / Services revokes a single privilege descriptor, USE.
- Revoking the USE privilege descriptor from an existing user takes effect upon the user's next attempt to use Oracle SQL / Services *after* the privilege change is complete. For example, a user whose USE privilege descriptor is revoked but who is still using Oracle SQL / Services, will not be able to use Oracle SQL / Services after disconnecting and then attempting to reconnect to the service.
- If you use the keyword IDENTIFIER[S], any specified identifier is removed from the service's list of granted identifiers. If you omit the IDENTIFIER[S] keyword, the specified user name is removed from the service's list of granted user names.

  If you revoke use of a service by a specific user name, that user is still able to access the service if the user holds an identifier that has been granted use of the service. Likewise, if you revoke use of a service by a specific identifier, a user who holds that identifier is still able to access the service if the user's name has been granted use of the service.

Examples

Example 1: Remove the USE privilege descriptor for the general service from PUBLIC.

```
SQLSRV> REVOKE USE ON SERVICE general FROM PUBLIC;
```

Example 2: Remove the USE privilege descriptor for the database_3 service from fred and wilma.

```
SQLSRV> REVOKE USE ON SERVICE database_3 FROM fred,wilma;
```

Example 3: Remove the USE privilege descriptor for the system management SQLSRV_MANAGE service from fred and wilma.

```
SQLSRV> REVOKE USE ON SERVICE sqlsrv_manage FROM fred,wilma;
```

Example 4: Remove the USE privilege descriptor for the system management SQLSRV_MANAGE service from the identifier names payroll_dba and operator.

```
SQLSRV> REVOKE USE ON SERVICE sqlsrv_manage
_SQLSRV> FROM IDENTIFIERS payroll_dba,operator;
```

SET CONFIGURATION_FILE Command

Enables you to select a server configuration file with which to start a server or to make server changes offline. Any subsequent management commands are written to the configuration file only and do not affect the running server except for GRANT USE and REVOKE USE commands and any restarted dispatchers and services.

Format

```
SET CONFIG[URATION]_FILE      <file-name>;


                              <file-name> ::=<quoted-string>
```

Arguments

<file-name>
The configuration file name. The file name is not required to be a quoted string.

Usage Notes

- CONFIG_FILE is a synonym for the keyword CONFIGURATION_FILE.
- When the SQLSRV_MANAGE utility starts up, it establishes a default configuration file name, as follows:

  ```
  SYS$MANAGER:SQLSRV_CONFIG_FILE71.DAT
  ```

  To override the default, set the SQLSRV_CONFIG_FILE71 logical name or supply a different file name to the SET CONFIGURATION_FILE command.
- The SHOW SETTINGS command shows the configuration file that offline modifications act upon. The SHOW SERVER command also shows the configuration file that online modifications act upon.
- If you issue the SET_CONFIG_FILE command and specify a configuration file specification that does not exist, you are prompted whether or not you want to create one now. The default is NO. If the SET CONFIRM command is set to OFF, then you are not prompted. A SHOW SETTINGS command displays the current settings and file specification for this new configuration file. If you issue a CREATE SERVER command, a server using this configuration file is created.

- When you make modifications to a configuration file using the SET CONFIG_ FILE command, all changes are made offline and do not affect the running server, except GRANT and REVOKE command changes. Changes made to a server's configuration file can be applied to the running server by restarting the object changed.

Examples

Example 1: Set the configuration file.

```
SQLSRV> SET CONFIG_FILE 'my_config_file';
```

SET CONFIRM Command

Echoes a confirmation prompt to the default output device when it is set as ON in the SQLSRV_MANAGE environment that requires confirmation for certain management operations.

Format

SET CONFIRM { ON | OFF } ;

Arguments

{ON | OFF}
When confirm is set as ON, a confirmation prompt echoes to the default output device requiring confirmation for certain management operations. When confirm is set as OFF, a confirmation prompt no longer echoes and no longer requires confirmation for certain management operations. ON is the default.

Usage Notes

- If the SET CONFIRM command is set as ON (the default) and you issue an SQLSRV_MANAGE command that in turn presents a confirmation prompt, this prompt is displayed on the default output device. For example, if you shut down and delete a server and then issue a SET_CONFIG_FILE command, and specify a configuration file that does not exist, you are prompted whether or not you want to create one now. The default is NO or not to create one now. If the SET CONFIRM command is set as OFF, you are not prompted to confirm this operation.
- A SHOW SETTINGS command displays, among other things, the current setting for the SET CONFIRM command.

Examples

Example 1: No longer echo a confirmation prompt to the default output device.

```
SQLSRV> SET CONFIRM OFF;
```

SET CONNECTION Command

Enables you to establish the specified connection as the current connection so that you can manage that server.

Format

```
SET CONNECT[ION]    [ <connect-name> ] ;

                    <connect-name> ::=<identifier>
```

Arguments

<connect-name>
The name of the connection. The identifier that uniquely identifies the connection to a server on a particular node. The connection name is expressed as an identifier.

Usage Notes

The SET CONNECT command allows you to manage multiple servers from a single SQLSRV_MANAGE session by switching between connections to the servers you are managing.

To manage a server online, you must first connect to the server using the CONNECT TO SERVER command. When you establish a new connection to a server using the CONNECT TO SERVER command, the new connection becomes the current connection. All online system management commands operate on the current connection. You can establish connections to multiple servers by issuing multiple CONNECT TO SERVER commands. You then use the SET CONNECT command to select the server that you wish to manage. Use the DISCONNECT SERVER command to disconnect from a server, at which time one of the remaining connections, if any, becomes the current connection.

Examples

Example 1: Manage two servers on nodes EAGLE and FALCON from node EAGLE.

```
SQLSRV> CONNECT SERVER AS EAGLE;
Connecting to server ...
Connected
```

```
SQLSRV> CONNECT SERVER AS FALCON NODE FALCON
_SQLSRV> USER ’dbsmgr’ USING ’password’;
Connecting to server ...
Connected
SQLSRV> SHOW CONNECTS;
Active connections:
CURRENT: FALCON
            Service: SQLSRV_MANAGE
            User: dbsmgr  Node: FALCON  Local: No
            Transport: DECNET  Object: SQLSRV_SERVER
            Request bufsize: 1024  Response bufsize: 1024

         EAGLE
            Service: SQLSRV_MANAGE
            User: <unknown>  Node: EAGLE  Local: Yes
            Transport: DECNET  Object: SQLSRV_SERVER
            Request bufsize: 1024  Response bufsize: 1024

SQLSRV> SHOW SERVICES;
                               C l i e n t s          E x e c u t o r s
Name            State    Per-Exec     Max   Active    Min    Max Running
RMU_SERVICE     RUNNING         1     100        0      4    100       4
GENERIC         RUNNING         1      10        0      2     10       2
SQLSRV_MANAGE   RUNNING       100       0        1      0      0       0

SQLSRV> SET CONNECT EAGLE;
SQLSRV> SHOW CONNECTS;
Active connections:
         FALCON
            Service: SQLSRV_MANAGE
            User: dbsmgr  Node: FALCON  Local: No
            Transport: DECNET  Object: SQLSRV_SERVER
            Request bufsize: 1024  Response bufsize: 1024

CURRENT: EAGLE
            Service: SQLSRV_MANAGE
            User: <unknown>  Node: EAGLE  Local: Yes
            Transport: DECNET  Object: SQLSRV_SERVER
            Request bufsize: 1024  Response bufsize: 1024

SQLSRV> SHOW SERVICES;
                               C l i e n t s          E x e c u t o r s
Name            State    Per-Exec     Max   Active    Min    Max Running
V61             RUNNING         1      20        2      5     20       5
RMU_SERVICE     RUNNING         1     100        3      4    100       4
```

```
GENERIC         RUNNING          1       50        5      20      50       20
SQLSRV_MANAGE   RUNNING        100        0        1       0       0        0
```

SET OUTPUT Command

Directs output to the default device if set as ON in the SQLSRV_MANAGE environment.

Format

SET OUTPUT { ON | OFF } ;

Arguments

{ON | OFF}
When output is set as ON it is directed to the default output device. When output is set as OFF it is no longer directed to the default output device but is directed to the current device, which is the terminal.

Usage Notes

None.

Examples

Example 1: Set the output to the default device.

```
SQLSRV> SET OUTPUT ON;
```

SET VERIFY Command

Echoes command file input to the default output device as it is read in the SQLSRV_MANAGE environment.

Format

SET VERIFY { ON | OFF } ;

Arguments

{ON | OFF}
When verify is set as ON, command file input echoes to the default output device. When verify is set as OFF, command file input no longer echoes to the default output device.

Usage Notes

None.

Examples

Example 1: Echo command file input to the default output device as it is read.

```
SQLSRV> SET VERIFY ON;
```

SHOW CLIENTS Command

Shows the active users for services for a configuration.

Format

```
SHOW CLIENTS        [ FOR ] <name-spec> [ FULL ] ;

                    <name-spec> ::={ * | <name-list> }
                    <name-list> ::={ [ SERVICE ] <service-name-list>
                       | [ USERNAME ] <user-name-list> | [ PID ] <executor-pid> }

                    <service-name-list> ::=<service-name> [ , <service-name> ] ...
                    <user-name-list> ::=<user-name> [ , <user-name> ] ...
                    <service-name> ::=<identifier>
                    <user-name> ::=<identifier>
                    <executor-pid> ::=<number>
```

Arguments

<name-spec>
The name specification. You can show:

- The clients connected to one or more services
- All clients connected to a server using a specific user name or list of user names
- Clients connected to a particular executor

<name-list>
Specifies the name list, which can be the service name list or user name list or the executor PID; depending on the keyword specified. If two or more service names or two or more user names are specified, each name *must* be separated by a comma. If one or more service names or user names are specified, then only information for those items is displayed.

<service-name>
The service object name is expressed as an identifier.

<user-name>
The user name is expressed as an identifier.

<exec-pid>
The executor PID is expressed as a number and can be represented either in decimal or hexadecimal format.

FULL
Displays a full description of information for each client. The default is to display brief information (one line of output) for each client. When no service name is specified, SQLSRV_MANAGE displays clients grouped by service name.

Usage Notes

- If no service object is specified, then information for *all* service objects is displayed.
- To show all clients for all services, you can either use the SHOW CLIENTS command and not specify the [FOR] [SERVICE] keywords, or specify an asterisk (*). Either method displays all clients for all services. For example:

  ```
  SQLSRV> SHOW CLIENTS;
  SQLSRV> SHOW CLIENTS *;
  ```

- To show all clients for all user names for all services, specify the SHOW CLIENTS FOR USERNAME command and do not specify user names.
- To show all clients for a specific executor PID, specify the SHOW CLIENTS FOR PID command and specify the executor PID.
- The executor PID can be represented in either decimal or hexadecimal format. To represent an executor PID in hexadecimal format, precede the executor PID value with the value ’0x’ or ’0X’ (for example, 0x0000088a).
- Client connections serviced by a session reusable service can be in one of three possible states (see Section 3.2 for more information):
  - Running Binding – The client is running and in the process of binding to an executor.
  - Running Bound – The client is running and is bound to an executor.
  - Canceling – The client connection is in the process of being disconnected.

- Client connections serviced by a transaction reusable database service can be in one of five possible states (see Section 3.2 for more information):
  - Running Binding – The client is running and in the process of binding to an executor.
  - Running Bound – The client is running and is bound to an executor.
  - Running Unbound – The client is not submitting requests, therefore is not bound to an executor, but it is still connected to its executor.
  - Canceling Binding – The client is in the process of informing the executor that the bound connect is going away (this operation precedes the Canceling operation).
  - Canceling – The client connection is in the process of being disconnected.
- This command also shows the management clients that are using the management service. The SHOW CLIENTS command allows server system managers to determine if other server system managers are connected to the server and using the management service.
- This command shows the actual location of executor log and error files and the location of an executor dump file should one be created.

Examples

Example 1: Show the clients for the universal service named generic and display a brief description.

```
SQLSRV> SHOW CLIENTS FOR SERVICE generic;
Service: GENERIC

  Connect         Client                           Executor
  Username        Node             State           PID       Application
  User1           123.0.0.1        RUNNING BOUND   28c0c4e6  Personnel
  User2           121.0.0.1        RUNNING BOUND   30b0a4d5  Personnel
```

Example 2: Show the clients for user names for all services and display a brief description.

```
SQLSRV> SHOW CLIENTS;
Service: SQLSRV_MANAGE

  Connect         Client                           Executor
  Username        Node             State           PID       Application
  User1           123.0.0.1        RUNNING BOUND   28c0c4e6  Personnel
  User1           123.0.0.1        RUNNING BOUND    29d0c4c7  SQLSRV_MANAGE
  User2           121.0.0.1        RUNNING BOUND    32b0b4c3  SQLSRV_MANAGE GUI
```

Example 3: Show the clients for the universal service named generic and display a full description.

```
SQLSRV> SHOW CLIENTS FOR SERVICE generic FULL;
Client Connect Username sqsapim1
    Service:      GENERIC
    Application:  Personnel
    State:        RUNNING BOUND
    Node:         12.34.567.89
    Executor:     GENERI0050002
    Executor PID: 543173877  0X20602cf5
    Log File:     SYS$SYSROOT:[SYSMGR]SQS_EAGLES_GENERI0050002.LOG
    Dump File:    SYS$SYSROOT:[SYSMGR]SQS_EAGLES_GENERI0050002.DMP

Client Connect Username sqsapim2
    Service:      GENERIC
    Application:  Personnel
    State:        RUNNING BOUND
    Node:         LOCAL:.mypc
    Executor:     GENERI0080004
    Executor PID: 543173877  0X20602cf5
    Log File:     SYS$SYSROOT:[SYSMGR]SQS_EAGLES_GENERI0080004.LOG
    Dump File:    SYS$SYSROOT:[SYSMGR]SQS_EAGLES_GENERI0080004.DMP
```

SHOW CONNECTIONS Command

Shows information about the current server.

Format

SHOW CONNECT[ION]S;

Usage Notes

- CONNECTS is a synonym for the keyword CONNECTIONS.
- The SHOW CONNECTS command shows you information about all of the active management connections that the SQLSRV_MANAGE utility has to each server. Use the SHOW CONNECTS command first to determine the current connection before issuing additional server management commands.

Examples

Example 1: Show information about the current server and connections to other servers.

```
SQLSRV> SHOW CONNECTS;
Active connections:
CURRENT: SQLSRV_MANAGE
            Service: SQLSRV_MANAGE
            User: system  Node: hawk  Local: No
            Transport: TCP/IP  Port-id: 2199
            Request bufsize: 1024  Response bufsize: 1024

         SQLSRV_MANAGE
            Service: SQLSRV_MANAGE
            User: system  Node: falcon  Local: Yes
            Transport: TCP/IP  Port-id: 2199
            Request bufsize: 1024  Response bufsize: 1024
```

SHOW DISPATCHER Command

Shows the static definition of all dispatcher objects and their operational state for the current server.

Format

```
SHOW DISPATCHER    [ <dispatcher-spec> ] ;

                   <dispatcher-spec> ::={ * | <dispatcher-name-list> }
                   <dispatcher-name-list> ::=<dispatcher-name> [ , <dispatcher-name> ] ...
                   <dispatcher-name> ::=<identifier>
```

Arguments

<dispatcher-spec>
The dispatcher object specification. This can be one or more dispatcher object names or can be specified with an asterisk (*). If an * is specified, information for all dispatcher object names is displayed.

<dispatcher-name-list>
The dispatcher object name list. If two or more dispatcher object names are specified, each dispatcher object name *must* be separated by a comma. If one or more dispatcher object names are specified, then only information for those named dispatcher objects is displayed.

<dispatcher-name>
The dispatcher object name is expressed as an identifier.

Usage Notes

- If no dispatcher object is specified, then information for *all* dispatcher objects is displayed.
- The dispatcher state and network port states can be one of three possible states:
  - Running – The dispatcher or dispatcher network port is running.
  - Inactive – The dispatcher or dispatcher network port is shut down.

  - Unknown – The management client is not connected to the server online so it cannot determine the state of the dispatcher and its network ports. The management client used the SET CONFIG_FILE command to manage the server offline. Use the CONNECT TO SERVER command to connect to the server online to determine the dispatcher state and the state of its network ports.

- When a difference exists for an attribute between the running server and its configuration file, the SHOW DISPATCHER command displays this difference at the end of the show output and indicates that when the server is restarted, the running server's dispatcher is updated to match the server's dispatcher definition in the configuration file.
- This command shows the actual location of dispatcher log and error files and the location of a dispatcher dump file should one be created. These file specifications are for informational purposes only and are not alterable attributes.

Examples

Example 1: Show information for the dispatcher for two different servers.

```
SQLSRV> CONNECT TO SERVER NODE hawk USER system USING password;
Connecting to server ...
Connected
SQLSRV> SHOW DISPATCHER sqlsrv_disp;
Dispatcher SQLSRV_DISP
    State:                     RUNNING
    Autostart:                 on
    Max connects:              100 clients
    Idle user Timeout:         <none>
    Max client buffer size:    5000 bytes
    Network Ports:                                   (State)   (Protocol)
      SPX/IPX port     0x84b1                        Inactive  SQL/Services
      DECnet  object   81                            Running   SQL/Services
      TCP/IP  port     118                           Running   SQL/Services
      SQL*Net listener FUBAR                         Running   SQL/Services
    Log File:                  SYS$MANAGER:SQS_EAGLE_SQLSRV_DIS100380.LOG
    Dump File:                 SYS$MANAGER:SQS_EAGLE_SQLSRV_DIS1003.DMP
```

SHOW SERVER Command

Shows the static definition of the server object defined and its operational state.

Format

SHOW SERVER;

Usage Notes

- The server network port state can be one of three possible states:
  - Running – The server network port is running.
  - Inactive – The server network port is shut down.
  - Unknown – The management client is not connected to the server online so it cannot determine the state of the server network ports. The management client used the SET CONFIG_FILE command to manage the server offline. Use the CONNECT TO SERVER command to connect to the server online to determine the state of its network ports.
- When a difference exists for an attribute between the running server and its configuration file, the SHOW SERVER command displays these differences at the end of the show output and indicates that when the server is restarted, the running server is updated to match the server's definition in the configuration file.
- Shows the actual location of server log and error files and the location of the server dump file should one be created. These file specifications are for informational purposes only and are not alterable attributes.

Examples

Example 1: Show information for the server defined in the configuration file.

```
SQLSRV> SHOW SERVER;
    Server Version:      7.1
    Server Platform:     Digital OpenVMS Alpha
    Max Shared Mem Size: 2000 Kb
    Config file:         SYS$SYSROOT:[SYSMGR]SQLSRV_CONFIG_FILE71.DAT;1
    Log path:            SYS$MANAGER:
    Dump path:           SYS$MANAGER:
```

```
Proc start time:     <none>
Proc shut time:      <none>
Network Ports:                                      (State)   (Protocol)
  DECnet  object   SQLSRV_SERVER                    Running   Native
  TCP/IP  port     2199                             Running   Native
Current shared memory usage:
  Allocation unit:      65536 bytes
  Total memory:       2031616 bytes ( 31 units)
  Free memory:        1769472 bytes ( 27 units)
  Partly allocated:    196608 bytes (  3 units)
Log File:            SYS$SYSROOT:[SYSMGR]SQS_CRANES_SQLSRV_MON_0071.LOG;
Dump File:           SYS$SYSROOT:[SYSMGR]SQS_CRANES_SQLSRV_71.DMP;
```

SHOW SERVICE Command

Shows the static definition of a service object or all service objects currently defined in the configuration file.

Format

```
SHOW SERVICE[S]    [ <service-spec> ] [ FULL ] ;


                   <service-spec> ::={ * | <service-name-list> }
                   <service-name-list> ::=<service-name> [ , <service-name> ] ...
                   <service-name> ::=<identifier>
```

Arguments

<service-spec>
The service object specification. This can be one or more service object names or can be specified with an asterisk (*). If one or more service object names are specified, then only information for those named service objects is displayed. If an * is specified, information for all service object names is displayed.

<service-name-list>
The service object name list. If two or more service object names are specified, each service object name *must* be separated by a comma. If one or more service object names are specified, then only information for those named service objects is displayed.

<service-name>
The service object name is expressed as an identifier.

FULL
Displays a full description of information for each service. The default is to display brief information (one line of output) for each service.

Usage Notes

- SERVICES is a synonym for the keyword SERVICE.

- If no service object is specified, then information for *all* service object names is displayed.
- The SHOW SERVICE * command and the SHOW SERVICE command both show you all of the services currently defined.
- The service state can be one of five possible states:
  - Starting – The service is starting.

    A service with MIN_EXECUTORS set to 0 never shows the Starting state when the service starts up. The state displays as either Running or Failed.
  - Failed – The service failed to start.
  - Running – The service is running.
  - Inactive – The service is shut down.
  - Unknown – The management client is not connected to the server online so it cannot determine the state of the service. The management client used the SET CONFIG_FILE command to manage the server offline. Use the CONNECT TO SERVER command to connect to the server online to determine the service state.
- When a difference exists for an attribute between the running server and its configuration file, the SHOW SERVICE command displays these differences at the end of the show output and indicates that when the server is restarted, the running server's service is updated to match the server's service definition in the configuration file.
- This command shows the list of user names granted access to the specified services.
- This command shows the list of identifiers granted access to the specified services.

Examples

Example 1: Show the services defined for a configuration and display a brief description.

```
SQLSRV> SHOW SERVICES;
                               C l i e n t s         E x e c u t o r s
Name            State    Per-Exec     Max   Active    Min     Max Running
SQLSRV_MANAGE   UNKNOWN       100     100        1      1       1       0
GENERIC         UNKNOWN         1      10        0      2      10       1
RMU_SERVICE     UNKNOWN         1     100        0      0     100       0
```

Example 2: Show the services defined for a configuration and display a full description.

```
SQLSRV> SHOW SERVICE payroll FULL;
Service PAYROLL
    State:                     RUNNING
    Owner:                     PAYROLLACCNT
    Protocol:                  SQL/Services
    Default Connect Username: <not specified>
    SQL version:               7.1
    Autostart:                 on
    Process init:              <not specified>
    Attach:                    ATTACH ’FILENAME PAYROLL_DISK:PAYROLL_DB’
    Schema:                    <not specified>
    Reuse:                     SESSION
    Database Authorization:   CONNECT USERNAME
    dbsrc file:                <not specified>
    SQL init file:             <not specified>
    Appl Transaction Usage:   SERIAL
    Idle User Timeout:         <none>
    Idle Exec Timeout:         1800 seconds
    Min Executors:             5
    Max Executors:             10
    Clients Per Executor:      1
    Active Clients:            0

Access to service PAYROLL
    Granted to users:
        PRIVILEGED_USER ’PAYROLLACCNT’
    Granted to identifiers:
        ’PAYROLL_DBA’  ’PAYROLLDEPT’
```

SHOW SETTINGS Command

Shows information about the current SQLSRV_MANAGE settings.

Format

SHOW SETTINGS;

Usage Notes

After starting the server, use the SHOW SETTINGS command to determine the current settings for the SQLSRV_MANAGE environment. Modify these SQLSRV_MANAGE environment settings for your own use.

Examples

Example 1: Show information about the current settings for the SQLSRV_MANAGE environment.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHOW SETTINGS;
Settings:
    version:    v7.1
    verify:     off
    output:     on
    config-file:        SYS$SYSROOT:[SYSMGR]SQLSRV_CONFIG_FILE.DAT;1
    confirm:    on
```

SHOW VERSION Command

Shows the version of the SQLSRV_MANAGE management client.

Format

SHOW VERSION;

Usage Notes

Use the SHOW VERSION command to determine the version of the SQLSRV_MANAGE management client.

Examples

Example 1: Show the version of the SQLSRV_MANAGE management client.

```
SQLSRV> SHOW VERSION;
Version:        v7.1
```

SHUTDOWN DISPATCHER Command

Shuts down the specified dispatcher.

Format

```
SHUTDOWN DISPATCHER          <dispatcher-name> ;

                             <dispatcher-name> ::=<identifier>
```

Arguments

<dispatcher-name>
Specifies the dispatcher object name. The dispatcher object name is expressed as an identifier.

Usage Notes

- Use the SHOW CLIENTS command to ensure no clients are connected to a service using a network transport being provided by the dispatcher that you are shutting down.
- You can shut down a dispatcher only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to shut down a dispatcher defined and running for that server.
- A dispatcher that has failed to start is left in a failed state and must be shut down. Correct the problem (usually an argument value is incorrectly specified), then start the dispatcher again.

Examples

Example 1: Shut down the dispatcher named disp_tcpip.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHOW CLIENTS;
SQLSRV> SHUTDOWN DISPATCHER disp_tcpip;
```

SHUTDOWN SERVER Command

Shuts down the current server.

Format

SHUTDOWN SERVER;

Usage Notes

- Use the SHOW CONNECTS command to ensure that you are shutting down the correct server.
- You can shut down a server only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to shut it down.

Examples

Example 1: Shut down the current server.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHOW CONNECTS;
Active connections:
CURRENT: SQLSRV_MANAGE
            Service: SQLSRV_MANAGE
            User: run_username  Node: EAGLE  Local: Yes
            Transport: DECNET  Object: SQLSRV_SERVER
            Request bufsize: 1024  Response bufsize: 1024
SQLSRV> SHUTDOWN SERVER;
Disconnected from Server
```

SHUTDOWN SERVICE Command

Shuts down the specified service.

Format

```
SHUTDOWN SERVICE          <service-name> ;

                          <service-name> ::=<identifier>
```

Arguments

<service-name>
Specifies the service object name. The service object name is expressed as an identifier.

Usage Notes

- Use the SHOW CLIENTS command to ensure that no clients are connected to the service that you are shutting down.
- You can shut down a service only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to shut down a service defined and running for that server.
- A service that has failed to start is left in a failed state and must be shut down. Correct the problem (usually an argument value is incorrectly specified), then start the service again.

Examples

Example 1: Shut down the universal service named generic.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> SHUTDOWN SERVICE generic;
```

START DISPATCHER Command

Starts a dispatcher process for the defined dispatcher object with the specified name for the current server.

Format

START DISPATCHER   <disp-name> ;

<disp-name> ::=<identifier>

Arguments

<disp-name>
Specifies the dispatcher name. The dispatcher name is expressed as an identifier.

Usage Notes

You can start a dispatcher only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to start a dispatcher defined for that server.

Examples

Example 1: Start the tcpip_disp dispatcher.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> START DISPATCHER tcpip_disp;
```

START SERVER Command

Starts the current server and optionally any defined dispatcher and service objects for the current server when AUTOSTART is set as ON and then connects to the server after starting it when AUTOCONNECT is set as ON.

Format

```
START SERVER
                  –>[ USER <user-name> USING <password> ]
                  –>[ AUTOCONNECT { ON | OFF } ]
                  –>[ AUTOSTART { ON | OFF } ] ;

                  <user-name> ::={ <quoted-string> | <identifier> }
                  <password> ::={ <quoted-string> | <identifier> }
```

Arguments

USER <user-name> USING <password>
Specifies a user name and password with which to connect to the server after the server has started. The user name and password are expressed as either a quoted-string or an identifier. You need not specify a user name and password if you are starting a server running DECnet or if you are starting a server running TCP/IP and you have SYSPRV or BYPASS privilege.

AUTOCONNECT {ON | OFF}
Determines whether or not to automatically connect to the server after you issue a START SERVER command. If the argument is specified as ON, SQLSRV_MANAGE automatically connects to the server after it starts the server when you issue a START SERVER command. A value of OFF starts the server but does not attempt to connect to the server after it has started. The default is ON.

AUTOSTART {ON | OFF}
Determines whether all other server objects (dispatchers and services) automatically start up when you issue a START SERVER command. If the argument is specified as ON, the default, all other server objects automatically start if each object's AUTOSTART argument value is also set as ON. If you do not want to start all other

server objects, specify the AUTOSTART attribute value as OFF in the START SERVER command. The AUTOSTART OFF attribute setting overrides each object’s AUTOSTART attribute setting and allows you to individually shut down and start each object after starting just the server object. The default is ON.

Usage Notes

- You can start a server only as an offline operation; that is, you must use the SET CONFIG_FILE command to select the configuration file of the server you want to start or use the default.
- After the server starts up with the AUTOCONNECT argument specified as ON, SQLSRV_MANAGE attempts to connect to any network port defined for the server. It tries each network port in a round-robin fashion up to three times each to establish the connection.
- You must have the SETPRV privilege or all privileges.
- When SQLSRV_MANAGE starts up, it establishes a default configuration file name:
  - The default configuration file is:

    ```
    SYS$MANAGER:SQLSRV_CONFIG_FILE71.DAT
    ```

  - To override the default, set the SQLSRV_CONFIG_FILE71 logical name or supply a different file name to the SET CONFIGURATION_FILE command.

Examples

Example 1: Start the current server.

```
SQLSRV> SET CONFIG_FILE ’my_config_file’;
SQLSRV> START SERVER;
Server started
Connecting to server ...
Connected
```

START SERVICE Command

Starts the specified, defined service object for the current server.

Format

START SERVICE   <service-name> ;

<service-name> ::=<identifier>

Arguments

<service-name>
Specifies the service name. The service name is expressed as an identifier.

Usage Notes

You can start a service only as an online operation; that is, you must be connected to a running server (CONNECT TO SERVER command) to start a service defined for that server.

Examples

Example 1: Start the universal service named V71.

```
SQLSRV> CONNECT SERVER;
Connecting to server ...
Connected
SQLSRV> START SERVICE v71;
```

Index