# Oracle®

Heterogenous Services

Release 8.1.7

January 2001

Part No. A88714-01

ORACLE®

Oracle Heterogeneous Services, Release 8.1.7

Part No. A88714-01



Primary Author: Ted Burroughs

Contributing Authors: Lance Ashdown and Pavna Jain

Contributors: John Bellemore, Jacco Draaijer, Diana Lorentz, Cynthia Kibbe, Nina Lewis, Raghu Mani, Basab Maulik, Kishan Peyetti, Paul Raveling, Katia Tarkhanov, Randy Urbano, and Sandy Venning

Graphic Designer: Valarie Moore

# Contents

**Send Us Your Comments** ..... ix

**Preface** ..... xi

## 1 Heterogeneous Services Concepts

**What is Heterogeneous Services?** ..... 1-2
**Heterogeneous Services Process Architecture** ..... 1-2
What is an Agent? ..... 1-3
Oracle Transparent Gateways ..... 1-3
Generic Connectivity ..... 1-4
**Heterogeneous Services Components** ..... 1-4
Transaction Service ..... 1-4
SQL Service ..... 1-5
Database Links to a Non-Oracle System ..... 1-5
**Architecture of the Heterogenous Services Data Dictionary** ..... 1-6
Classes and Instances ..... 1-7
Configuration Data ..... 1-8
Data Dictionary Views ..... 1-8

## 2 Managing Heterogeneous Services

**Setting Up Access to Non-Oracle Systems** ..... 2-2
Step 1: Install the Heterogeneous Services Data Dictionary ..... 2-2
Step 2: Set Up the Environment to Access Heterogeneous Services Agents ..... 2-2
Step 3: Create the Database Link to the Non-Oracle System ..... 2-4

Step 4: Test the Connection ..... 2-4
**Registering Agents** ..... 2-5
Enabling Agent Self-Registration ..... 2-5
Disabling Agent Self-Registration ..... 2-6
Using Agent Self-Registration to Avoid Configuration Mismatches ..... 2-6
Understanding Agent Self-Registration ..... 2-7
Specifying HS_AUTOREGISTER ..... 2-9
**Using Heterogeneous Services Data Dictionary Views** ..... 2-9
Understanding Types of Views ..... 2-10
Understanding Sources of Data Dictionary Information ..... 2-12
Using General Views ..... 2-12
Using Transaction Service Views ..... 2-13
Using SQL Service Views ..... 2-14
**Using the Heterogeneous Services Dynamic Performance Views** ..... 2-16
Determining Which Agents Are Running on a Host ..... 2-16
Determining the Open Heterogeneous Services Sessions ..... 2-17
Determining the Heterogeneous Services Parameters ..... 2-18
**Using the DBMS_HS Package** ..... 2-18
Specifying Initialization Parameters ..... 2-19
Unspecifying Initialization Parameters ..... 2-20

## 3 Generic Connectivity

**What Is Generic Connectivity?** ..... 3-2
Types of Agents ..... 3-2
Generic Connectivity Architecture ..... 3-3
SQL Execution ..... 3-6
Datatype Mapping ..... 3-6
Generic Connectivity Restrictions ..... 3-6
**Supported Oracle SQL Statements** ..... 3-7
Functions Supported by Generic Connectivity ..... 3-7
**Configuring Generic Connectivity Agents** ..... 3-8
Creating the Initialization File ..... 3-8
Editing the Initialization File ..... 3-9
Setting Initialization Parameters for an ODBC-based Data Source ..... 3-10
Setting Initialization Parameters for an OLE DB-based Data Source ..... 3-12

**ODBC Connectivity Requirements** ..... 3-13
**OLE DB (SQL) Connectivity Requirements** ..... 3-15
**OLE DB (FS) Connectivity Requirements** ..... 3-16
Data Source Properties ..... 3-18

## 4 Developing Applications with Heterogeneous Services

**Developing Applications with Heterogeneous Services: Overview** ..... 4-2
**Developing Applications Using Pass-Through SQL** ..... 4-2
Using the DBMS_HS_PASSTHROUGH package ..... 4-2
Considering the Implications of Using Pass-Through SQL ..... 4-3
Executing Pass-Through SQL Statements ..... 4-3
**Optimizing Data Transfers Using Bulk Fetch** ..... 4-9
Using OCI, an Oracle Precompiler, or Another Tool for Array Fetches ..... 4-10
Controlling the Array Fetch Between Oracle Database Server and Agent ..... 4-11
Controlling the Array Fetch Between Agent and Non-Oracle Server ..... 4-11
Controlling the Reblocking of Array Fetches ..... 4-11
**Researching the Locking Behavior of Non-Oracle Systems** ..... 4-12
**Limitations to Heterogeneous Services** ..... 4-12

## A Heterogeneous Services Initialization Parameters

**HS_COMMIT_POINT_STRENGTH** ..... A-3
**HS_DB_DOMAIN** ..... A-3
**HS_DB_INTERNAL_NAME** ..... A-4
**HS_DB_NAME** ..... A-4
**HS_DESCRIBE_CACHE_HWM** ..... A-4
**HS_FDS_CONNECT_INFO** ..... A-5
**HS_FDS_SHAREABLE_NAME** ..... A-6
**HS_FDS_TRACE_LEVEL** ..... A-6
**HS_FDS_TRACE_FILE_NAME** ..... A-6
**HS_LANGUAGE** ..... A-7
**HS_NLS_DATE_FORMAT** ..... A-8
**HS_NLS_DATE_LANGUAGE** ..... A-8
**HS_NLS_NCHAR** ..... A-9
**HS_OPEN_CURSORS** ..... A-9
**HS_ROWID_CACHE_SIZE** ..... A-10

**HS_RPC_FETCH_REBLOCKING** ..... A-10
**HS_RPC_FETCH_SIZE** ..... A-11

## B Heterogeneous Services Data Dictionary Views

## C DBMS_HS_PASSTHROUGH for Pass-Through SQL

**Summary of Subprograms** ..... C-2
BIND_VARIABLE procedure ..... C-3
BIND_VARIABLE_RAW procedure ..... C-4
BIND_OUT_VARIABLE procedure ..... C-6
BIND_OUT_VARIABLE_RAW procedure ..... C-8
BIND_INOUT_VARIABLE procedure ..... C-10
BIND_INOUT_VARIABLE_RAW procedure ..... C-12
CLOSE_CURSOR function ..... C-14
EXECUTE_IMMEDIATE function ..... C-15
EXECUTE_NON_QUERY function ..... C-16
FETCH_ROW function ..... C-17
GET_VALUE procedure ..... C-19
GET_VALUE_RAW procedure ..... C-20
OPEN_CURSOR function ..... C-22
PARSE procedure ..... C-23

## D Data Dictionary Translation for Generic Connectivity

**Data Dictionary Translation Support** ..... D-2
Accessing the Non-Oracle Data Dictionary ..... D-2
Supported Views and Tables ..... D-3
**Data Dictionary Mapping** ..... D-4
Default Column Values ..... D-5
**Generic Connectivity Data Dictionary Descriptions** ..... D-6
**ALL_CATALOG** ..... D-6
**ALL_COL_COMMENTS** ..... D-6
**ALL_CONS_COLUMNS** ..... D-6
**ALL_CONSTRAINTS** ..... D-7
**ALL_IND_COLUMNS** ..... D-7

**ALL_INDEXES** ..... D-8
**ALL_OBJECTS** ..... D-10
**ALL_TAB_COLUMNS** ..... D-11
**ALL_TAB_COMMENTS** ..... D-12
**ALL_TABLES** ..... D-12
**ALL_USERS** ..... D-14
**ALL_VIEWS** ..... D-14
**DICTIONARY** ..... D-14
**USER_CATALOG** ..... D-15
**USER_COL_COMMENTS** ..... D-15
**USER_CONS_COLUMNS** ..... D-15
**USER_CONSTRAINTS** ..... D-15
**USER_IND_COLUMNS** ..... D-16
**USER_INDEXES** ..... D-17
**USER_OBJECTS** ..... D-19
**USER_TAB_COLUMNS** ..... D-19
**USER_TAB_COMMENTS** ..... D-20
**USER_TABLES** ..... D-21
**USER_USERS** ..... D-22
**USER_VIEWS** ..... D-23

## E Datatype Mapping

**Mapping ODBC Datatypes to Oracle Datatypes** ..... E-2
**Mapping OLE DB Datatypes to Oracle Datatypes** ..... E-3

## Index

# Send Us Your Comments

**Oracle Heterogeneous Services, Release 8.1.7**

**Part No. A88714-01**

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: infodev@us.oracle.com
- FAX: (650) 506-7228 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op12
  Redwood Shores, CA 94065
  USA

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

.

# Preface

*Oracle Heterogeneous Services* describes implementation issues for Oracle8*i* Heterogeneous Services. It also introduces the tools and utilities available to assist you in implementing and using this feature.

This preface contains these topics:

- Audience
- Organization
- Related Documentation
- Conventions

> **See Also:** *Getting to Know Oracle8i* for information about the differences between Oracle8*i* and the Oracle8*i* Enterprise Edition.

## Audience

*Oracle Heterogeneous Services* is intended for database administrators who administer or plan to administer a distributed database system involving either Oracle to Oracle database links or Oracle to non-Oracle database links.

To use this document, you need to be familiar with:

- Relational database concepts and basic database administration as described in *Oracle8i Concepts* and the *Oracle8i Administrator's Guide*.
- The operating system environment under which database administrators are running Oracle.

## Organization

This document contains:

Chapter 1, "Heterogeneous Services Concepts" provides an overview of Oracle Heterogeneous Services.

Chapter 2, "Managing Heterogeneous Services" explains how to implement and maintain Heterogeneous Services using an Oracle Transparent Gateway.

Chapter 3, "Generic Connectivity" provides the information you need to connect to non-Oracle datastores through ODBC or OLE DB.

Chapter 4, "Developing Applications with Heterogeneous Services" provides the information you will need to develop applications that use Oracle Heterogeneous Services.

Appendix A, "Heterogeneous Services Initialization Parameters" lists all Heterogeneous Services-specific initialization parameters and their values.

Appendix B, "Heterogeneous Services Data Dictionary Views" lists the data dictionary views that are available through Heterogeneous Services mapping.

Appendix C, "DBMS_HS_PASSTHROUGH for Pass-Through SQL" describes the procedures and functions in the package DBMS_HS_PASSTHROUGH for pass-through SQL of Heterogeneous Services.

Appendix D, "Data Dictionary Translation for Generic Connectivity" explains and lists data dictionary translations for generic connectivity.

Appendix E, "Datatype Mapping" explains how datatypes are mapped for ODBC and OLE DB compliant data sources.

## Related Documentation

For more information, see these Oracle resources:

- *Oracle8i Concepts*
- *Oracle8i Administrator's Guide*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://technet.oracle.com/membership/index.htm`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://technet.oracle.com/docs/index.htm`

## Conventions

This section describes the conventions used in the text and code examples of the this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

### Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

| Convention | Meaning | Example |
|---|---|---|
| **Bold** | Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both. | The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid.<br>When you specify this clause, you create an **index-organized table**. |
| *Italics* | Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders. | *Oracle8i Concepts*<br>You can specify the *parallel_clause*.<br>Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading. |
| UPPERCASE monospace (fixed-width font) | Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles. | You can specify this clause only for a NUMBER column.<br>You can back up the database using the BACKUP command.<br>Query the TABLE_NAME column in the USER_TABLES data dictionary view.<br>Specify the ROLLBACK_SEGMENTS parameter.<br>Use the DBMS_STATS.GENERATE_STATS procedure. |
| lowercase monospace (fixed-width font) | Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values. | Enter sqlplus to open SQL*Plus.<br>The department_id, department_name, and location_id columns are in the hr.departments table.<br>Set the QUERY_REWRITE_ENABLED initialization parameter to true.<br>Connect as oe user. |

## Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

| Convention | Meaning | Example |
|---|---|---|
| [ ] | Brackets enclose one or more optional items. Do not enter the brackets. | `DECIMAL (digits [ , precision ])` |
| { } | Braces enclose two or more items, one of which is required. Do not enter the braces. | `{ENABLE \| DISABLE}` |
| \| | A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar. | `{ENABLE \| DISABLE}`<br>`[COMPRESS \| NOCOMPRESS]` |
| . . . | Horizontal ellipsis points indicate either:<br>■ That we have omitted parts of the code that are not directly related to the example<br>■ That you can repeat a portion of the code | `CREATE TABLE ... AS subquery;`<br>`SELECT col1, col2, ... , col`*`n`* `FROM employees;` |
| .<br>.<br>. | Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example. | |
| Other notation | You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown. | `acctbal NUMBER(11,2);`<br>`acct CONSTANT NUMBER(4) := 3;` |
| *Italics* | Italicized text indicates variables for which you must supply particular values. | `CONNECT SYSTEM/`*`system_password`* |
| UPPERCASE | Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase. | `SELECT last_name, employee_id FROM employees;`<br>`SELECT * FROM USER_TABLES;`<br>`DROP TABLE hr.employees;` |
| lowercase | Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. | `SELECT last_name, employee_id FROM employees;`<br>`sqlplus hr/hr` |

# 1

# Heterogeneous Services Concepts

This chapter describes the basic concepts of Heterogeneous Services.

This chapter contains these topics:

- What is Heterogeneous Services?
- Heterogeneous Services Process Architecture
- Heterogeneous Services Components
- Architecture of the Heterogenous Services Data Dictionary

> **See Also:** *Getting to Know Oracle8i* about features that are new to this release.

# What is Heterogeneous Services?

Heterogeneous Services is a component within the Oracle database server that is required to access a non-Oracle database system.

The term "non-Oracle database system" refers to the following:

- Any system accessed by PL/SQL procedures written in C (that is, by external procedures)
- Any system accessed through SQL (that is, by Oracle Transparent Gateways or generic connectivity)
- Any system accessed procedurally (that is, by procedural gateways)

> **Note:** This manual documents Heterogeneous Services as it relates to gateways. For more information on external procedures, see *Oracle8i SQL Reference* and *Oracle8i Application Developer's Guide - Fundamentals*.

Heterogeneous Services makes it possible for Oracle database server users to do the following:

- Use Oracle SQL statements to retrieve data stored in non-Oracle systems.
- Use Oracle procedure calls to access non-Oracle systems, services, or application programming interfaces (APIs) from within an Oracle distributed environment.

Heterogeneous Services is generally applied in one of two ways:

- Users use an Oracle Transparent Gateway in conjunction with Heterogeneous Services to access a particular non-Oracle system for which the Oracle Transparent Gateway has been designed. (For example, you use the Oracle Transparent Gateway for Sybase on Solaris to access a Sybase database system operating on a Sun Solaris platform.)
- Users use Heterogeneous Services' generic connectivity to access non-Oracle databases through ODBC or OLE DB interfaces.

# Heterogeneous Services Process Architecture

Heterogenous Services is composed of two basic components:

- Generic code that is part of the database server and which performs most of the processing for Heterogeneous Services
- Agent generic code that is not part of the database server but which is necessary in order that the Oracle database server be able to communicate with a non-Oracle database system

## What is an Agent?

An agent is the Heterogeneous Services process that links the Oracle database server into the code of the non-Oracle system. Agent generic code in Heterogeneous Services in combination with a driver becomes an agent. Drivers are specific to the type of non-Oracle system you want to access and provide the systems interface between the non-Oracle system and the agent generic code of Oracle Heterogeneous Services.

An agent can reside in the following places:

- On the same machine as the non-Oracle system
- On the same machine as the Oracle server
- On a machine different from either of these two

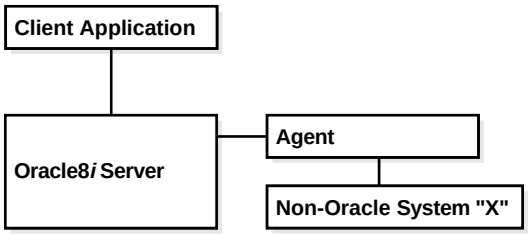
Agent processes are activated when a user session accesses a non-Oracle system through a database link on an Oracle database server. These connections are made using Oracle's remote data access software, Net8, that enables both client-server and server-server communication. The agent process continues to run until the user session is disconnected or the database link is explicitly closed.

## Oracle Transparent Gateways

An Oracle Transparent Gateway is a particular kind of agent that is designed by Oracle Corporation to access commercially produced brands and versions of database systems not marketed by Oracle Corporation. For example, an Oracle Transparent Gateway for Sybase on Solaris is designed to access Sybase database systems that are running on Solaris platforms.

With Oracle Transparent Gateways, you can use an Oracle database server to access data anywhere in a distributed database system without needing to know the location of the data or how it is stored. Also, when the results of your queries are returned to you by the Oracle database server, they are presented to you as if the datastores from which they were taken all resided within a remote Oracle database. This functionality is called transparency; when you are using it, you are transparently accessing a non-Oracle database system.

*Figure 1–1 Accessing Heterogeneous Non-Oracle Systems*




## Generic Connectivity

Generic connectivity is a feature of the Oracle database server that enables users to use ODBC and OLE DB drivers to access non-Oracle systems having an ODBC or an OLE DB interface.

A gateway using generic connectivity must have an additional ODBC or OLE DB driver to provide an interface between generic connectivity and the non-Oracle system. These drivers are not provided by Oracle Corporation. However, as long as Oracle Corporation supports the ODBC and OLE DB protocols, you can use these gateways to access their respective non-Oracle systems.

The ODBC and OLE DB drivers using generic connectivity are installed in the same Oracle Home directory as the Oracle database server. Connecting to one of these gateways from another Oracle database server is not supported.

**See Also:** Chapter 3, "Generic Connectivity"

# Heterogeneous Services Components

Heterogeneous Services provides the following components:

- Transaction Service
- SQL Service

## Transaction Service

The transaction service allows non-Oracle systems to be integrated into Oracle database server transactions and sessions. Users transparently set up an authenticated session in the non-Oracle system when it is accessed for the first time over a database link within an Oracle user session. At the end of the Oracle user

session, the authenticated session in the non-Oracle system is transparently closed at the non-Oracle system.

Additionally, one or more non-Oracle systems can participate in an Oracle distributed transaction. When an application commits a transaction, Oracle's two-phase commit protocol accesses the non-Oracle system to transparently coordinate the distributed transaction. Even if the non-Oracle system does not support all aspects of Oracle's two-phase commit protocol, the Oracle database server usually supports distributed transactions with the non-Oracle system.

The SQL service uses the transaction service while Oracle's object transaction service uses agents that implement only the transaction service.

> **See Also:** "Using Transaction Service Views" on page 2-13 for more information on heterogeneous distributed transactions.

## SQL Service

The SQL service uses SQL to access the non-Oracle system transparently. If an application's SQL request requires data from a non-Oracle system, Heterogeneous Services does the following:

1. Translates the Oracle SQL request into an equivalent SQL request for the non-Oracle system.
2. Accesses the non-Oracle data.
3. Makes the data from the non-Oracle system available to the Oracle database server for post-processing.

The SQL service provides the following capabilities:

- Translates Oracle's SQL into a SQL dialect understood by the non-Oracle system.
- Translates specific SQL requests that are made on Oracle data dictionary tables (which tell the Oracle database server how to organize the data in its datastore) into equivalent requests on the non-Oracle system's data dictionary tables.
- Maps non-Oracle system datatypes to Oracle datatypes.

## Database Links to a Non-Oracle System

With Heterogeneous Services, a non-Oracle system appears to the user as a remote Oracle database server. To access or manipulate tables or to execute procedures in the non-Oracle system, you must create a database link that specifies the connect

descriptor for the non-Oracle database. Use the following syntax to create a link to a non-Oracle system (variables in italics):

```
CREATE DATABASE LINK link_name
  CONNECT TO user IDENTIFIED BY password
  USING 'non_oracle_system';
```

If a non-Oracle system is referenced, then Heterogeneous Services translates the SQL statement or PL/SQL remote procedure call into the appropriate statement at the non-Oracle system.

You can access tables and procedures at the non-Oracle system by qualifying the tables and procedures with the database link. This operation is identical to accessing tables and procedures at a remote Oracle database server.

Consider the following example, which accesses a non-Oracle system through a database link:

```
SELECT * FROM EMP@non_oracle_system;
```

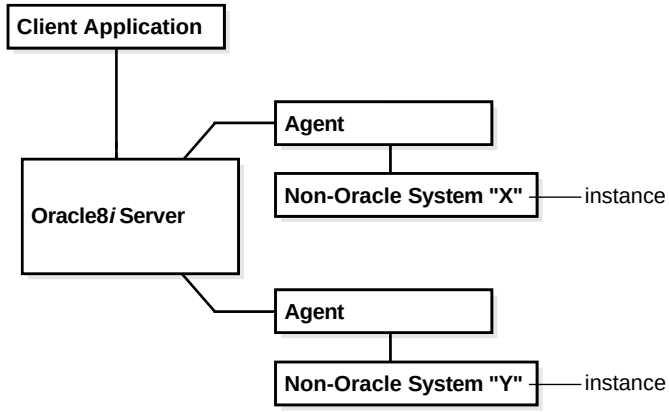
Heterogeneous Services translates the Oracle SQL statement into the SQL dialect of the target system and then executes the translated SQL statement at the non-Oracle system.

**See Also:** Chapter 2, "Managing Heterogeneous Services"

## Architecture of the Heterogenous Services Data Dictionary

You can access multiple non-Oracle systems from the same Oracle database server, as illustrated in Figure 1–2.

***Figure 1–2 Accessing Multiple Non-Oracle Instances***




For Heterogeneous Services generic code to correctly generate SQL, map datatypes, and interact effectively with the code of the non-Oracle system, Heterogeneous Services needs information about that system. This information is uploaded from the agent for that system and is stored in the Heterogeneous Services data dictionary.

## Classes and Instances

Oracle organizes information about the non-Oracle system by two levels of granularity in the Heterogeneous Services data dictionary. These two levels of granularity are class and instance. A class pertains to a specific type of non-Oracle system. For example, you might want to access the class of Sybase database systems with your Oracle database server. An instance defines specializations within a class. For example, you might want to access several separate instances within a Sybase database system. Instance information takes precedence over class information, and class information takes precedence over server-supplied defaults.

Although it is possible to store data dictionary information at one level of granularity by having completely separate definitions in the data dictionary for each individual instance, this could lead the amount of stored data dictionary information to become unnecessarily large and redundant. To avoid this, Oracle organizes the data dictionary by two levels of granularity, in which each class definition (one level of granularity) is shared by all the particular instances (a second level of granularity) under that class.

For example, consider a case where the Oracle database server accesses three instances of Sybase and two instances of Ingres II. Sybase and Ingres II each have their own code, which requires separate class definitions for the Oracle database server to be able to access them. The Heterogeneous Services data dictionary therefore would contain two class definitions, one for Sybase and one for Ingres II, with five instance definitions, one for each instance being accessed by the Oracle database server.

## Configuration Data

The Heterogeneous Services data dictionary also contains the following types of configuration data:

- Heterogeneous Services initialization parameters to provide control over various things, including language and date formats, domain names, and Heterogeneous Services tuning
- Capability definitions to identify details such as the SQL language features supported by the non-Oracle data source
- Data dictionary translations to map references to Oracle data dictionary tables and views into equivalents specific to the non-Oracle data source

## Data Dictionary Views

The Heterogeneous Services data dictionary views contain information about:

- Names of instances and classes uploaded into the Oracle data dictionary
- Capabilities, including SQL translations, defined for each class or instance
- Data Dictionary translations defined for each class or instance
- Initialization parameters defined for each class or instance
- Distributed external procedures accessible from the Oracle database server

You can access information from the Oracle data dictionary by using fixed views. The views can be divided into three main types:

- General views
- Views used for the transaction service
- Views used for the SQL service

**See Also:**

- "Using Heterogeneous Services Data Dictionary Views" on page 2-9 to learn how to use these views
- Appendix B, "Heterogeneous Services Data Dictionary Views" for a list of data dictionary views that Heterogeneous Services supports

# 2

# Managing Heterogeneous Services

This chapter describes how to maintain a heterogeneous distributed environment when using a transparent gateway.

This chapter contains these topics:


- Setting Up Access to Non-Oracle Systems
- Registering Agents
- Using Heterogeneous Services Data Dictionary Views
- Using the Heterogeneous Services Dynamic Performance Views
- Using the DBMS_HS Package

# Setting Up Access to Non-Oracle Systems

This section explains the generic steps to configure access to a non-Oracle system. Please see the Installation and User's Guide for your agent for more installation information. The instructions for configuring your agent may slightly differ from the following.

The steps for setting up access to a non-Oracle system are:

- Step 1: Install the Heterogeneous Services Data Dictionary
- Step 2: Set Up the Environment to Access Heterogeneous Services Agents
- Step 3: Create the Database Link to the Non-Oracle System
- Step 4: Test the Connection

## Step 1: Install the Heterogeneous Services Data Dictionary

For most users, the script to install data dictionary tables and views for Heterogeneous Services is automatically run at the time of installation.

In case you need to install these tables and views manually, you must run a script that creates the Heterogeneous Services data dictionary tables, views, and packages. On most systems this script is called `caths.sql` and resides in `$ORACLE_HOME/rdbms/admin`.

---

**Note:** Data dictionary tables, views, and packages might already be installed on your Oracle database server. Check for the existence of Heterogeneous Services data dictionary views, such as SYS.HS_FDS_CLASS.

---

## Step 2: Set Up the Environment to Access Heterogeneous Services Agents

To initiate a connection to the non-Oracle system, the Oracle database server starts an agent process through the Net8 listener. For the Oracle database server to be able to connect to the agent, you must:

1. Set up a Net8 service name for the agent that can be used by the Oracle database server. The Net8 service name descriptor includes protocol-specific information needed to access the Net8 listener. The service name descriptor must include the (HS=OK) clause to ensure that the connection uses Oracle Heterogeneous Services.
2. Set up the listener to listen for incoming request from the Oracle database server and spawn Heterogeneous Services agents. Modify the `listener.ora`

file so that the listener can start Heterogeneous Services agents, and then restart the listener.

### Sample Entry for a Net8 Service Name

The following is a sample entry for the service name in the `tnsnames.ora` file:

```
Sybase_sales= (DESCRIPTION=
                    (ADDRESS=(PROTOCOL=tcp)
                             (HOST=dlsun206)
                             (PORT=1521))
                    (CONNECT_DATA = (SID=SalesDB))
                    (HS = OK))
```

The description of this service name is defined in `tnsnames.ora`, the Oracle Names server, or in third-party name servers using the Oracle naming adapter. See the installation documentation for your agent for more information about how to define the Net8 service name.

### A Sample Listener Entry

The following is a sample entry for the listener in the `listener.ora` file:

```
LISTENER =
   (ADDRESS_LIST =
      (ADDRESS= (PROTOCOL=tcp)
                (HOST = dlsun206)
                (PORT = 1521)
      )
  )
...
SID_LIST_LISTENER =
  (SID_LIST =
      (SID_DESC = (SID_NAME=SalesDB)
                  (ORACLE_HOME=/home/oracle/tg4sybs/8.1.6)
                  (PROGRAM=tg4sybs)
      )
  )
```

The value associated with PROGRAM keyword defines the name of the agent executable. The agent executable must reside in the `$ORACLE_HOME/bin` directory. Typically, you use SID_NAME to define the initialization parameter file for the agent.

## Step 3: Create the Database Link to the Non-Oracle System

To create a database link to the non-Oracle system, use the CREATE DATABASE LINK statement. The service name that is used in the USING clause of the CREATE DATABASE LINK command is the Net8 service name.

For example, to create a database link to the SALES database on Sybase, enter:

```
CREATE DATABASE LINK sales
USING ‘Sybase_sales’;
```

## Step 4: Test the Connection

To test the connection to the non-Oracle system, use the database link in a SQL or PL/SQL statement. If the non-Oracle system is a SQL-based database, you can execute a SELECT statement from an existing table or view using the database link. For example, enter:

```
SELECT * FROM product@sales
WHERE product_name like '%pencil%';
```

When you try to access the non-Oracle system for the first time, the Heterogeneous Services agent uploads information into the Heterogeneous Services data dictionary. The uploaded information includes:

| Type of Data | Explanation |
|---|---|
| Capabilities of the non-Oracle system | For example, the agent specifies whether it can perform a join or a GROUP BY. |
| SQL translation information | The agent specifies how to translate Oracle functions and operators into functions and operators of the non-Oracle system. |
| Data dictionary translations | The agent specifies how to translate Oracle data dictionary tables into tables and views of the non-Oracle system, to make the data dictionary information of the non-Oracle system available just as if it were an Oracle data dictionary, |

**Note:** Most agents upload information into the Oracle data dictionary automatically the first time they are accessed. However, some Oracle Transparent Gateway products provide scripts that you must run on the Oracle database server.

**See Also:** "Using Heterogeneous Services Data Dictionary Views" on page 2-9.

# Registering Agents

Registration is an operation through which Oracle stores information about an agent in the Heterogeneous Services data dictionary. Agents do not have to be registered. If an agent is not registered, Oracle stores information about the agent in memory instead of in the data dictionary. When a session involving an agent terminates, this information ceases to be available.

Self-registration is an operation in which a database administrator sets an initialization parameter that lets the agent automatically upload information into the data dictionary. In release 8.0 of the Oracle database server, an agent could determine whether to self-register. In this release, self-registration occurs only when the HS_AUTOREGISTER initialization parameter is set to TRUE (default).

This section contains the following topics:

- Enabling Agent Self-Registration
- Using Heterogeneous Services Data Dictionary Views
- Using Agent Self-Registration to Avoid Configuration Mismatches
- Understanding Agent Self-Registration

## Enabling Agent Self-Registration

To ensure correct operation over heterogeneous database links, agent self-registration automates updates to Heterogeneous Services configuration data that describe agents on remote hosts. Agent self-registration is the default behavior. If you do not want to use the agent self-registration feature, then you must set the HS_AUTOREGISTER initialization parameter to FALSE.

Both the server and the agent rely on three types of information to configure and control operation of the Heterogeneous Services connection. These three sets of information are collectively called Heterogeneous Services configuration data.

| Heterogeneous Services Configuration Data | Description |
|---|---|
| Heterogeneous Services initialization parameters | Provide control over various things, including language and date formats, domain names, and Heterogeneous Services tuning. |

| Heterogeneous Services Configuration Data | Description |
|---|---|
| Capability definitions | Identify details such as SQL language features supported by the non-Oracle datasource. |
| Data dictionary translations | Map references to Oracle data dictionary tables and views into equivalents specific to the non-Oracle data source. |

**See Also:** "Specifying HS_AUTOREGISTER" on page 2-9.

## Disabling Agent Self-Registration

To disable agent self-registration, set the HS_AUTOREGISTER initialization parameter as follows:

```
HS_AUTOREGISTER = FALSE
```

If you disable agent self-registration, then agent information is not stored in the data dictionary. Consequently, the Heterogeneous Services data dictionary views cease to be useful sources of information. However, the Oracle database server still requires information about the class and instance of each agent. To meet this requirement when agent self-registration is disabled, the Oracle database server stores this information in local memory.

## Using Agent Self-Registration to Avoid Configuration Mismatches

Heterogeneous Services configuration data is stored in the Oracle database server's data dictionary. Because the agent is possibly remote, and can therefore be administered separately, several circumstances can lead to configuration mismatches between servers and agents:

- An agent can be newly installed on a separate machine so that the server has no Heterogeneous Services data dictionary content to represent the agent's Heterogeneous Services configuration data.
- A server can be newly installed and lack the necessary Heterogeneous Services configuration data for existing agents and non-Oracle data stores.
- A non-Oracle instance can be upgraded from an older version to a newer version, requiring modification of the Heterogeneous Services configuration data.

- A Heterogeneous Services agent at a remote site can be upgraded to a new version or patched, requiring modification of the Heterogeneous Services configuration data.
- A database administrator at the non-Oracle site can change the agent setup, possibly for tuning or testing purposes, in a manner that affects Heterogeneous Services configuration data.

Agent self-registration permits successful operation of Heterogeneous Services in all these scenarios. Specifically, agent self-registration enhances interoperability between any Oracle database server and any Heterogeneous Services agent, provided that each is at least as recent as Version 8.0.3. The basic mechanism for this functionality is the ability to upload Heterogeneous Services configuration data from agents to servers.

Self-registration provides automatic updating of Heterogeneous Services configuration data residing in the Oracle database server data dictionary. The update ensures that the agent self-registration uploads need to be done only once, during the initial use of a previously unregistered agent. Instance information is uploaded on each connection and is not stored in the database server data dictionary.

## Understanding Agent Self-Registration

The Heterogeneous Services agent self-registration feature can:

- Identify the agent and the non-Oracle data store to the Oracle database server
- Permit agents to define Heterogeneous Services initialization parameters for use both by the agent and connected Oracle database servers
- Upload capability definitions and data dictionary translations, if available, from an Heterogeneous Services agent during connection initialization

> **Note:** When both the server and the agent are release 8.1 or higher, the upload of class information occurs only when the class is undefined in the database server data dictionary. Similarly, instance information is uploaded only if the instance is undefined in the server data dictionary.

The information required to accomplish this is accessed in the database server data dictionary by using the following agent-supplied names:

- FDS_CLASS
- FDS_CLASS_VERSION

> **See Also:** "Using Heterogeneous Services Data Dictionary Views" on page 2-9 to learn how to use the Heterogeneous Services data dictionary views.

#### FDS_CLASS and FDS_CLASS_VERSION

FDS_CLASS and FDS_CLASS_VERSION are defined by Oracle or by third-party vendors for each individual Heterogeneous Services agent and version. Oracle Heterogeneous Services concatenates these names to form FDS_CLASS_NAME, which is used as a primary key to access class information in the server data dictionary.

FDS_CLASS should specify the type of non-Oracle data store to be accessed and FDS_CLASS_VERSION should specify a version number for both the non-Oracle data store and the agent to which it connects. Note that when any component of an agent changes, FDS_CLASS_VERSION must also change to uniquely identify the new release.

---

**Note:** This information is uploaded when you initialize each connection.

---

#### FDS_INST_NAME

Instance-specific information can be stored in the database server data dictionary. The instance name, FDS_INST_NAME, is configured by the database administrator who administers the agent; how the database administrator performs this configuration depends on the specific agent in use.

The Oracle database server uses FDS_INST_NAME to look up instance-specific configuration information in its data dictionary. Oracle uses the value as a primary key for columns of the same name in these views:

- FDS_INST_INIT
- FDS_INST_CAPS
- FDS_INST_DD

Server data dictionary accesses that use FDS_INST_NAME also use FDS_CLASS_NAME to uniquely identify configuration information rows. Instances of the same

name but that occur under different classes have separate sets of configuration information. For example, if your database contains a Sybase816 and a Sybase817 class, but both of these classes have an instance called SALES, then each SALES instance has a separate set of configuration information.

Unlike class information, instance information is not automatically self-registered in the server data dictionary.

- If the database server data dictionary contains instance information, it represents setup details defined by the database administrator that fully define the instance configuration. No instance information is uploaded from the agent to the server.
- If the database server data dictionary does not contain instance information, any instance information made available by a connected agent is uploaded to the database server for use in that connection. The uploaded instance data is not stored in the database server data dictionary.

### Specifying HS_AUTOREGISTER

The Oracle database server initialization parameter HS_AUTOREGISTER enables or disables automatic self-registration of Heterogeneous Services agents. This parameter is specified in the Oracle initialization parameter file, not the agent initialization file.

For example, you can set the parameter as follows:

```
HS_AUTOREGISTER = TRUE
```

When set to TRUE, the agent uploads information describing a previously unknown agent class or a new agent version into the server's data dictionary.

Oracle Corporation recommends that you use the default value for this parameter (TRUE), which ensures that the server's data dictionary content always correctly represents definitions of class capabilities and data dictionary translations as used in Heterogeneous Services connections.

**See Also:** *Oracle8i Reference* for a description of this parameter.

## Using Heterogeneous Services Data Dictionary Views

You can use the Heterogeneous Services data dictionary views to access information about Heterogeneous Services. This section addresses the following topics:

- Understanding Types of Views

- Understanding Sources of Data Dictionary Information
- Using General Views
- Using Transaction Service Views
- Using SQL Service Views


## Understanding Types of Views

The Heterogeneous Services data dictionary views, which all begin with the prefix *HS_*, can be divided into three main types:

- General views
- Views used for the transaction service
- Views used for the SQL service

Most of the data dictionary views are defined for both classes and instances. Consequently, for most types of data there is a *_CLASS and an *_INST view.

***Table 2–1 Data Dictionary Views for Heterogeneous Services***

| View | Type | Identifies |
|---|---|---|
| HS_BASE_CAPS | SQL service | All capabilities supported by Heterogeneous Services |
| HS_BASE_DD | SQL service | All data dictionary translation table names supported by Heterogeneous Services |
| HS_CLASS_CAPS | Transaction service, SQL service | Capabilities for each class |
| HS_CLASS_DD | SQL service | Data dictionary translations for each class |
| HS_CLASS_INIT | General | Initialization parameters for each class |
| HS_FDS_CLASS | General | Classes accessible from this Oracle database server |
| HS_FDS_INST | General | Instances accessible from this Oracle database server |
| HS_INST_CAPS | Transaction service, SQL service | Capabilities for each instance |
| HS_INST_DD | SQL service | Data dictionary translations for each instance |
| HS_INST_INIT | General | Initialization parameters for each instance |

Like all Oracle data dictionary tables, the views are read-only. Do not use SQL to change the content of any of the underlying tables. To make changes to any of the underlying tables, use the procedures available in the DBMS_HS package.

**See Also:**

- "Heterogeneous Services Process Architecture" on page 1-2 for more information about classes and instances
- *Oracle8i Reference* for information about Heterogeneous Services views
- "Using the DBMS_HS Package" on page 2-18 for more information about the DBMS_HS package

## Understanding Sources of Data Dictionary Information

The values used for data dictionary content in any particular connection on a Heterogeneous Services database link can come from any of the following sources, in order of precedence:

- Instance information uploaded by the connected Heterogeneous Services agent at the start of the session. This information overrides corresponding content in the Oracle data dictionary, but is never stored in the Oracle data dictionary.
- Instance information stored in the Oracle data dictionary. This data overrides any corresponding content for the connected class.
- Class information stored in the Oracle data dictionary

If the Oracle database server runs with the HS_AUTOREGISTER server initialization parameter set to FALSE, then no information is stored automatically in the Oracle data dictionary. The equivalent data is uploaded by the Heterogeneous Services agent on a connection-specific basis each time a connection is made, with any instance-specific information taking precedence over class information.

---

**Note:** Because an agent can upload instance information, it is not possible to determine positively what capabilities and what data dictionary translations are in use for a given session.

---

You can determine the values of Heterogeneous Services initialization parameters by querying the VALUE column of the V$HS_PARAMETER view. Note that the VALUE column of V$HS_PARAMETER truncates the actual initialization parameter value from a maximum of 255 characters to a maximum of 64 characters, and it truncates the parameter name from a maximum of 64 characters to a maximum of 30 characters.

## Using General Views

The views that are common for all services are as follows:

| View | Contains |
|---|---|
| HS_FDS_CLASS<br>HS_FDS_INST | Names of the instances and classes that are uploaded into the Oracle8*i* data dictionary |
| HS_CLASS_INIT<br>HS_INST_INIT | Information about the Heterogeneous Services initialization parameters |

For example, you can access multiple Sybase gateways from an Oracle database server. After accessing the gateways for the first time, the information uploaded into the Oracle database server could appear as follows:

```
SQL> SELECT * FROM hs_fds_class;

FDS_CLASS_NAME        FDS_CLASS_COMMENTS             FDS_CLASS_ID
--------------------- ------------------------------ ------------
Sybase816             Uses Sybase driver, R1.1                1
Sybase817             Uses Sybase driver, R1.2               21
```

Two classes are uploaded: a class that accesses Sybase816 and a class that accesses Sybase817. The data dictionary in the Oracle database server now contains capability information, SQL translations, and data dictionary translations for both Sybase816 and Sybase817.

In addition to this information, the Oracle database server data dictionary also contains instance information in the HS_FDS_INST view for each non-Oracle system instance that is accessed.

## Using Transaction Service Views

When a non-Oracle system is involved in a distributed transaction, the transaction capabilities of the non-Oracle system and the agent control whether it can participate in distributed transactions. Transaction capabilities are stored in the HS_CLASS_CAPS and HS_INST_CAPS capability tables.

The ability of the non-Oracle system and agent to support two-phase commit protocols is specified by the 2PC type capability, which can specify one of the following five types.

| | |
|---|---|
| Read-only (RO) | The non-Oracle system can only be queried with SQL SELECT statements. Procedure calls are not allowed because procedure calls are assumed to write data. |
| Single-Site (SS) | The non-Oracle system can handle remote transactions but not distributed transactions. That is, it *cannot* participate in the two-phase commit protocol. |
| Commit Confirm (CC) | The non-Oracle system can participate in distributed transactions. It can participate in Oracle's two-phase commit protocol but only as the Commit Point Site. That is, it *cannot* prepare data, but it *can* remember the outcome of a particular transaction if asked by the global coordinator. |

| | |
|---|---|
| Two-Phase Commit | The non-Oracle system can participate in distributed transactions. It can participate in Oracle's two-phase commit protocol, as a regular two-phase commit node, but not as a Commit Point Site. That is, it *can* prepare data, but it *cannot* remember the outcome of a particular transaction if asked to by the global coordinator. |
| Two-Phase Commit Confirm | The non-Oracle system can participate in distributed transactions. It can participate in Oracle's two-phase commit protocol as a regular two-phase commit node or as the Commit Point Site. That is, it *can* prepare data and it *can* remember the outcome of a particular transaction if asked by the global coordinator. |

The transaction model supported by the driver and non-Oracle system can be queried from Heterogeneous Services data dictionary views HS_CLASS_CAPS and HS_INST_CAPS.

An example of the two-phase commit capability follows:

```
SELECT cap_description, translation
FROM   hs_class_caps
WHERE  cap_description LIKE '2PC%'
AND    fds_class_name='Sybase';

CAP_DESCRIPTION                          TRANSLATION
---------------------------------------- -----------
2PC type (RO-SS-CC-PREP/2P-2PCC)                  CC
```

When the non-Oracle system and agent support distributed transactions, the non-Oracle system is treated like any other Oracle database server. When a failure occurs during the two-phase commit protocol, the transaction is recovered automatically. If the failure persists, the in-doubt transaction may need to be manually overridden by the database administrator.

## Using SQL Service Views

Data dictionary views that are specific for the SQL service contain information about:

- SQL capabilities and SQL translations of the non-Oracle data source
- Data dictionary translations to map Oracle data dictionary views to the data dictionary of the non-Oracle system.

**Note:** This section describes only a portion of the SQL Service-related capabilities. Because you should never need to alter these settings for administrative purposes, these capabilities are not discussed here.

### Using Views for Capabilities and Translations

The HS_*_CAPS data dictionary tables contain information about the SQL capabilities of the non-Oracle data source and required SQL translations. These views specify whether the non-Oracle data store or the Oracle database server implements certain SQL language features. If a capability is turned off, then the Oracle database server does not send any SQL statements to the non-Oracle data source that require this particular capability, but it still performs post-processing.

### Using Views for Data Dictionary Translations

In order to make the non-Oracle system appear similar to an Oracle database server, Heterogeneous Services connections map a limited set of Oracle data dictionary views onto the non-Oracle system's data dictionary. This mapping permits applications to issue queries as if these views belonged to an Oracle data dictionary. Data dictionary translations make this access possible. These translations are stored in Heterogeneous Services views whose names are suffixed with _DD.

For example, the following SELECT statement transforms into a Sybase query that retrieves information about EMP tables from the Sybase data dictionary table:

```
SELECT * FROM USER_TABLES@salesdb
WHERE UPPER(TABLE_NAME)='EMP';
```

Data dictionary tables can be mimicked instead of translated. If a data dictionary translation is not possible because the non-Oracle data source does not have the required information in its data dictionary, Heterogeneous Services causes it to appear as if the data dictionary table is available, but the table contains no information.

To retrieve information for which Oracle data dictionary views or tables are translated or mimicked for the non-Oracle system, you can issue the following query on the HS_CLASS_DD or HS_INST_DD views:

```
SELECT DD_TABLE_NAME, TRANSLATION_TYPE
FROM   HS_CLASS_DD
WHERE  FDS_CLASS_NAME='Sybase';
```

```
DD_TABLE_NAME                  T
-----------------------------  -
ALL_ARGUMENTS                  M
ALL_CATALOG                    T
ALL_CLUSTERS                   T
ALL_CLUSTER_HASH_EXPRESSIONS   M
ALL_COLL_TYPES                 M
ALL_COL_COMMENTS               T
ALL_COL_PRIVS                  M
ALL_COL_PRIVS_MADE             M
ALL_COL_PRIVS_RECD             M
...
```

The translation type 'T' specifies that a translation exists. When the translation type is 'M', the data dictionary table is mimicked.

> **See Also:** Appendix B, "Heterogeneous Services Data Dictionary Views" for a list of data dictionary views that are supported through Heterogeneous Services mapping.

# Using the Heterogeneous Services Dynamic Performance Views

The Oracle database server stores information about agents, sessions, and parameters. You can use the V$ dynamic performance views to access this information. This section contains the following topics:

- Determining Which Agents Are Running on a Host
- Determining the Open Heterogeneous Services Sessions

## Determining Which Agents Are Running on a Host

The following view shows generation information about agents:

| View | Purpose |
|---|---|
| V$HS_AGENT | Identifies the set of Heterogeneous Services agents currently running on a given host, using one row per agent process. |

Use this view to determine general information about the agents running on a specified host. The following table describes the most relevant columns

**See also:** *Oracle8i Reference* for a description of all the columns in the view.

***Table 2–2 V$HS_AGENT***

| Column | Description |
|---|---|
| AGENT_ID | Net8 session identifier used for connections to agent (`listener.ora` SID) |
| MACHINE | Operating system machine name |
| PROGRAM | Program name of agent |
| AGENT_TYPE | Type of agent |
| FDS_CLASS_ID | The ID of the foreign data store class |
| FDS_INST_ID | The instance name of the foreign data store |

## Determining the Open Heterogeneous Services Sessions

The following view shows which Heterogeneous Services sessions are open for the Oracle database server:

| View | Purpose |
|---|---|
| V$HS_SESSION | Lists the sessions for each agent, specifying the database link used. |

The following table shows the most relevant columns

**See also:** *Oracle8i Reference* for a description of all the columns in the view.

***Table 2–3 V$HS_SESSION***

| Column | Description |
|---|---|
| HS_SESSION_ID | Unique Heterogeneous Services session identifier |
| AGENT_ID | Net8 session identifier used for connections to agent (`listener.ora` SID) |
| DB_LINK | Database link name used to access the agent NULL means that no database link is used (such as, when using external procedures) |

*Table 2–3 V$HS_SESSION*

| Column | Description |
|---|---|
| DB_LINK_OWNER | Owner of the database link in DB_LINK |

## Determining the Heterogeneous Services Parameters

The following view shows which Heterogeneous Services parameters are set in the Oracle database server:

| View | Purpose |
|---|---|
| V$HS_PARAMETER | Lists Heterogeneous Services parameters and values registered in the Oracle database server. |

The following table describes the most relevant columns.

> **See Also:** *Oracle8i Reference* for a description of all the columns in the view.

*Table 2–4 V$HS_SESSION*

| Column | Description |
|---|---|
| HS_SESSION_ID | Unique Heterogeneous Services session identifier |
| PARAMETER | The name of the Heterogeneous Services parameter |
| VALUE | The value of the Heterogeneous Services parameter |

# Using the DBMS_HS Package

The DBMS_HS package contains functions and procedures that allow you to specify and unspecify Heterogeneous Services initialization parameters, capabilities, instance names, and class names. These parameters are configured in the gateway initialization file, not the Oracle initialization parameter file. The only exception is HS_AUTOREGISTER, which is set in the Oracle initialization parameter file.

> **See Also:** *Oracle8i Supplied PL/SQL Packages Reference* for a reference listing of all DBMS_HS package interface information for Heterogeneous Services administration.

## Specifying Initialization Parameters

Set initialization parameters either in the Oracle database server or in the Heterogeneous Services agent.

To set initialization parameters in the Oracle database server, use the DBMS_HS package. See the agent's installation documentation for more information. If the same initialization parameter is set both in the agent and the Oracle database server, then the value of the initialization parameter set in the Oracle database server takes precedence.

Many, although not all, Oracle gateways allow initialization parameters to be set inside the initialization files. The name of the initialization file is usually init*agent_sid*.ora and it is usually located in $ORACLE_HOME/*product_name*/admin. Parameters set in the server override those set in the initialization files.

**See Also:** Chapter 3, "Generic Connectivity"

There are two types of initialization parameters to consider when setting up your gateway:

| Type | Description |
|---|---|
| Generic | Defined by Heterogeneous Services. See Appendix A, "Heterogeneous Services Initialization Parameters" for more information on generic initialization parameters. |
| Non-Oracle class-specific | Defined by the Oracle transparent gateway product. Some non-Oracle data store class-specific parameters may be mandatory. For example, a parameter may include connection information required to connect to a non-Oracle system. These parameters are documented in the installation documentation for your agent. |

You can set both generic and non-Oracle data store class-specific Heterogeneous Services initialization parameters in the Oracle database server using the CREATE_INST_INIT procedure in the DBMS_HS package.

For example, set the HS_DB_DOMAIN initialization parameter as follows

```
DBMS_HS.CREATE_INST_INIT
          (FDS_INST_NAME   => 'SalesDB',
           FDS_CLASS_NAME  => 'Sybase',
           INIT_VALUE_NAME => 'HS_DB_DOMAIN',
           INIT_VALUE      => 'US.SALES.COM');
```

**See Also:** Appendix A, "Heterogeneous Services Initialization Parameters" for more information about initialization parameters.

## Unspecifying Initialization Parameters

To unspecify an Heterogeneous Services initialization parameter in the Oracle database server, use the DROP_INST_INIT procedure. For example, to delete the HS_DB_DOMAIN entry, enter:

```
DBMS_HS.DROP_INST_INIT
          (FDS_INST_NAME   => 'SalesDB',
           FDS_CLASS_NAME  => 'Sybase',
           INIT_VALUE_NAME => 'HS_DB_DOMAIN');
```

**See Also:** *Oracle8i Supplied PL/SQL Packages Reference* for a full description of the DBMS_HS package.

# 3

# Generic Connectivity

This chapter describes the configuration and usage of generic connectivity agents.

This chapter contains these topics:


- What Is Generic Connectivity?
- Supported Oracle SQL Statements
- Configuring Generic Connectivity Agents
- ODBC Connectivity Requirements
- OLE DB (SQL) Connectivity Requirements
- OLE DB (FS) Connectivity Requirements

# What Is Generic Connectivity?

Generic connectivity is intended for low-end data integration solutions requiring the ad hoc query capability to connect from an Oracle database server to non-Oracle database systems. Generic connectivity is enabled by Oracle Heterogeneous Services, allowing you to connect to non-Oracle systems with improved performance and throughput.

Generic connectivity is implemented as either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent. An ODBC agent and OLE DB agent are included as part of your Oracle system. Be sure to use the agents shipped with your particular Oracle system, installed in the same `$ORACLE_HOME`.

Any data source compatible with the ODBC or OLE DB standards described in this chapter can be accessed using a generic connectivity agent.

This section contains the following topics:


- Types of Agents
- Generic Connectivity Architecture
- SQL Execution
- Datatype Mapping
- Generic Connectivity Restrictions


## Types of Agents

Generic connectivity is implemented as one of the following types of Heterogeneous Services agents:

- ODBC agent for accessing ODBC data providers
- OLE DB agent for accessing OLE DB data providers that support SQL processing—sometimes referred to as *OLE DB (SQL)*
- OLE DB agent for accessing OLE DB data providers without SQL processing support—sometimes referred to as *OLE DB (FS)*

Each user session receives its own dedicated agent process spawned by the first use in that user session of the database link to the non-Oracle system. The agent process ends when the user session ends.

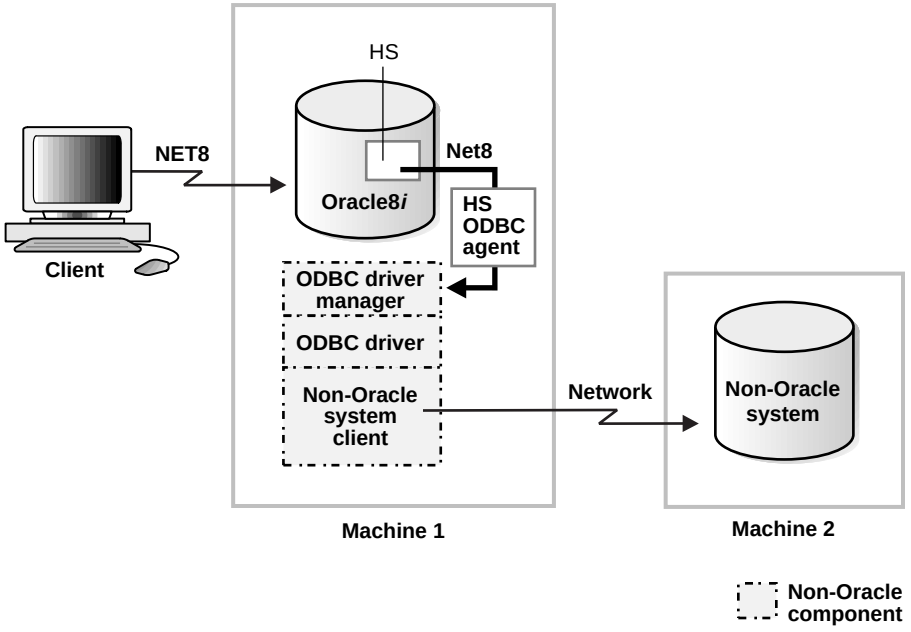
## Generic Connectivity Architecture

To access the non-Oracle data store using generic connectivity, the agents work with an ODBC or OLE DB driver. The Oracle database server provides support for the ODBC or OLE DB driver interface. The driver that you use must be on the same platform as the agent. The non-Oracle data stores can reside on the same machine as the Oracle database server or a different machine.

### Oracle and Non-Oracle Systems on Separate Machines

Figure 3–1 shows an example of a configuration in which an Oracle and non-Oracle database are on separate machines, communicating through an Heterogeneous Services ODBC agent.

***Figure 3–1 Non-Oracle System on Separate Computer***




In this configuration:

1. A client connects to the Oracle database server through Net8

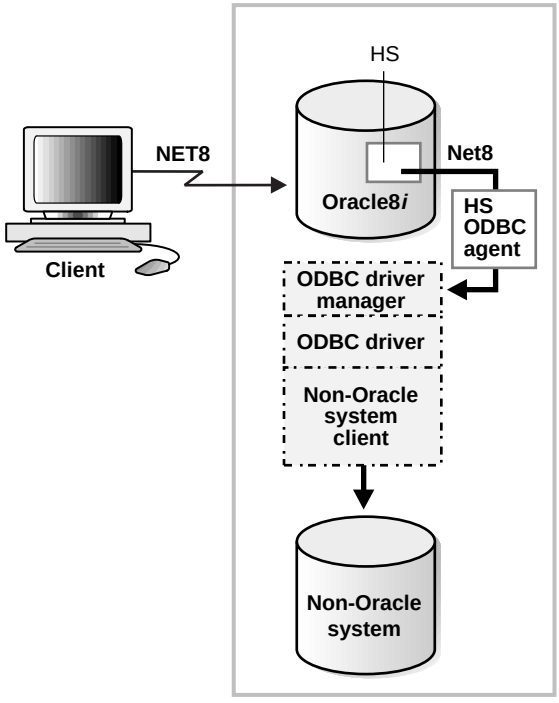
2. The Heterogeneous Services component of the Oracle database server connects through Net8 to the Heterogeneous Services ODBC agent
3. The agent communicates with the following non-Oracle components:
   - An ODBC driver manager
   - An ODBC driver
   - A non-Oracle client application

This client connects to the non-Oracle data store through a network.

### Oracle and Non-Oracle Systems on Same Machine

Figure 3–2 shows an example of a different configuration in which an Oracle and non-Oracle database are on the same machine, again communicating through an Heterogeneous Services ODBC agent.

***Figure 3–2 Accessing Heterogeneous Non-Oracle Systems***




In this configuration:

1. A client connects to the Oracle database server through Net8
2. The Heterogeneous Services component of the Oracle database server connects through Net8 to the Heterogeneous Services ODBC agent
3. The agent communicates with the following non-Oracle components:
   - An ODBC driver manager
   - An ODBC driver

The driver then allows access to the non-Oracle data store.

> **Note:** The ODBC driver may require non-Oracle client libraries even if the non-Oracle database is located on the same machine.

## SQL Execution

SQL statements sent using a generic connectivity agent are executed differently depending on the type of agent you are using: ODBC, OLE DB (SQL), or OLE DB (FS). For example, if a SQL statement involving tables is sent using an ODBC agent for a file-based storage system, the file can be manipulated as if it were a table in a relational database. The naming conventions used at the non-Oracle system can also depend on whether you are using an ODBC or OLE DB agent.

## Datatype Mapping

The Oracle database server maps the datatypes used in ODBC and OLE DB compliant data sources to supported Oracle datatypes. When the results of a query are returned, the Oracle database server converts the ODBC or OLE DB datatypes to Oracle datatypes. For example, the ODBC datatype SQL_TIMESTAMP and the OLE DB datatype DBTYPE_DBTIMESTAMP are converted to Oracle's DATE datatype.

## Generic Connectivity Restrictions

Generic connectivity restrictions include:

- A table including a BLOB column must have a separate column that serves as a primary key
- BLOB/CLOB data cannot be read through passthrough queries
- Updates or deletes that include unsupported functions within a WHERE clause are not allowed
- Stored procedures are not supported
- Generic connectivity agents cannot participate in distributed transactions; they support single-site transactions only

# Supported Oracle SQL Statements

Generic connectivity supports the following statements, but only if the ODBC or OLE DB driver and non-Oracle system can execute them *and* the statements contain supported Oracle SQL functions:

- DELETE
- INSERT
- SELECT
- UPDATE

Only a limited set of functions are assumed to be supported by the non-Oracle system. Most Oracle functions have no equivalent function in this limited set. Consequently, many Oracle functions are not supported by generic connectivity, although post-processing is performed by the Oracle database server, possibly impacting performance.

If an Oracle SQL function is not supported by generic connectivity, then this function is not supported in DELETE, INSERT, or UPDATE statements. In SELECT statements, these functions are evaluated by the Oracle database server and post-processed after they are returned from the non-Oracle system.

If an unsupported function is used in a DELETE, INSERT, or UPDATE statement, it generates this Oracle error:

```
ORA-02070: database db_link_name does not support function in this context
```

## Functions Supported by Generic Connectivity

Generic connectivity assumes that the following minimum set of SQL functions is supported:

- AVG(*exp*)
- LIKE(*exp*)
- COUNT(*)
- MAX(*exp*)
- MIN(*exp*)
- NOT

# Configuring Generic Connectivity Agents

To implement generic connectivity on a non-Oracle data source, you must set the agent parameters.

This section contains the following topics:


- Creating the Initialization File
- Editing the Initialization File
- Setting Initialization Parameters for an ODBC-based Data Source
- Setting Initialization Parameters for an OLE DB-based Data Source


## Creating the Initialization File

You must create and customize an initialization file for your generic connectivity agent. Oracle Corporation supplies sample initialization files named `init`*`agent`*`.ora`, where *`agent`* is `odbc` or `oledb`, indicating which agent the sample file can be used for, as in the following:

```
initodbc.ora
initoledb.ora
```

The sample files are stored in the `$ORACLE_HOME/hs/admin directory`.

To create an initialization file for an ODBC or OLE DB agent, copy the applicable sample initialization file and rename the file to `init`*`HS_SID`*`.ora`, where *HS_SID* is the system identifier you want to use for the instance of the non-Oracle system to which the agent connects.

The *HS_SID* is also used to identify how to connect to the agent when you configure the listener by modifying the `listener.ora` file. The *HS_SID* you add to the `listener.ora` file must match the *HS_SID* in an `init`*`HS_SID`*`.ora` file, because the agent spawned by the listener searches for a matching `init`*`HS_SID`*`.ora` file. That is how each agent process gets its initialization information. When you copy and rename your `init`*`HS_SID`*`.ora` file, ensure it remains in the `$ORACLE_HOME/hs/admin` directory.

> **See Also:** "Step 2: Set Up the Environment to Access Heterogeneous Services Agents" for more information on configuring the listener.

## Editing the Initialization File

Customize the `init`*`HS_SID`*`.ora` file by setting the parameter values used for generic connectivity agents to values appropriate for your system, agent, and drivers. You must edit the `init`*`HS_SID`*`.ora` file to change the HS_FDS_CONNECT_INFO initialization parameter. HS_FDS_CONNECT_INFO specifies the information required for connecting to the non-Oracle system.

> **See Also:** Appendix A, "Heterogeneous Services Initialization Parameters" for more information on parameters.

Set the parameter values as follows:

`[SET][PRIVATE]` *`parameter=value`*

where:

`[SET][PRIVATE]` are optional keywords. If you do not specify either SET or PRIVATE, the parameter and value are simply used as an initialization parameter for the agent.

SET specifies that in addition to being used as an initialization parameter, the parameter value is set as an environment variable for the agent process.

PRIVATE specifies that the parameter value is private and not transferred to the Oracle database server and does not appear in V$ tables or in an graphical user interfaces.

SET PRIVATE specifies that the parameter value is set as an environment variable for the agent process and is also private (not transferred to the Oracle database server, not appearing in V$ tables or graphical user interfaces).

*parameter* is the Heterogeneous Services initialization parameter that you are specifying. See Appendix A, "Heterogeneous Services Initialization Parameters" for a description of all Heterogeneous Services parameters and their possible values. The parameter is case-sensitive.

*value* is the value you want to specify for the Heterogeneous Services parameter. The value is case-sensitive.

For example, to enable tracing for an agent, set the HS_FDS_TRACE_LEVEL parameter as follows:

```
HS_FDS_TRACE_LEVEL=ON
```

Typically, most parameters are only needed as initialization parameters, so you do not need to use SET or PRIVATE. Use SET for parameter values that the drivers or non-Oracle system need as environment variables.

PRIVATE is only supported for the follow Heterogeneous Services parameters:

- HS_FDS_CONNECT_INFO
- HS_FDS_SHAREABLE_NAME
- HS_FDS_TRACE_LEVEL
- HS_FDS_TRACE_FILE_NAME

You should only use PRIVATE for these parameters if the parameter value includes sensitive information such as a username or password.

## Setting Initialization Parameters for an ODBC-based Data Source

The settings for the initialization parameters vary depending on the type of operating system.

### Setting Agent Parameters on Windows NT

Specify a File DSN or a System DSN which has previously been defined using the ODBC Driver Manager.

When connecting using a File DSN, specify the value as follows:

```
HS_FDS_CONNECT_INFO=FILEDSN=file_dsn
```

When connecting using a System DSN, specify the value as follows:

```
HS_FDS_CONNECT_INFO=system_dsn
```

If you are connecting to the data source through the driver for that data source, precede the DSN by the name of the driver, followed by a semi-colon (;).

**Setting Parameters on NT: Example** Assume a System DSN has been defined in the Windows ODBC Data Source Administrator. In order to connect to this SQL Server database through the gateway, the following line is required in `init`*`HS_SID`*`.ora`:

```
HS_FDS_CONNECT_INFO=sqlserver7
```

where *sqlserver7* is the name of the System DSN defined in the Windows ODBC Data Source Administrator.

The following procedure enables you to define a System DSN in the Windows ODBC Data Source Administrator:

1. From the **Start** menu, choose **Settings** > **Control Panel** and select the **ODBC** icon.
2. Select the **System DSN** tab to display the system data sources.
3. Click **Add**.
4. From the list of installed ODBC drivers, select the name of the driver that the data source will use. For example, select **SQL Server**.
5. Click **Finish**.
6. Enter a name for the DSN and an optional description. Enter other information depending on the ODBC driver. For example, for SQL Server enter the SQL Server machine.

   **Note:** The name entered for the DSN must match the value of the parameter HS_FDS_CONNECT_INFO that is specified in `init`*`HS_SID`*`.ora`.

7. Continue clicking **Next** and answering the prompts until you click **Finish**).
8. Click **OK** until you exit the ODBC Data Source Administrator.

### Setting Agent Parameters on UNIX platforms

Specify a DSN and the path of the ODBC shareable library, as follows:

```
HS_FDS_CONNECT_INFO=dsn_value
HS_FDS_SHAREABLE_NAME=full_odbc_library_path_of_odbc_driver
```

HS_FDS_CONNECT_INFO is required for all platforms for an ODBC agent. HS_FDS_SHAREABLE_NAME is required on UNIX platforms for an ODBC agent. Other initialization parameters have defaults or are optional. You can use the default values and omit the optional parameters, or you can specify the parameters with values tailored for your installation.

**Note:** Before deciding to accept the default values or change them, see Appendix A, "Heterogeneous Services Initialization Parameters" for detailed information on all the initialization parameters.

**Setting Parameters on UNIX: Example** Assume that the `odbc.ini` file for connecting to Informix using the Intersolve ODBC driver is located in `/opt/odbc` and includes the following information:

```
[ODBC Data Sources]
Informix=INTERSOLV 3.11 Informix Driver

[Informix]
Driver=/opt/odbc/lib/ivinf13.so
Description=Informix
Database=personnel@osf_inf72
HostName=osf
LogonID=uid
Password=pwd
```

In order to connect to this Informix database through the gateway, the following lines are required in `init`*`HS_SID`*`.ora`:

```
HS_FDS_CONNECT_INFO=Informix
HS_FDS_SHAREABLE_NAME=/opt/odbc/lib/libodbc.so
set INFORMIXDIR=/users/inf72
set INFORMIXSERVER=osf_inf72
set ODBCINI=/opt/odbc/odbc.ini
```

Note that the set statements are optional as long as they are specified in the working account. Each database has its own set statements.

The HS_FDS_CONNECT_INFO parameter value must match the ODBC data source name in the `odbc.ini` file.

## Setting Initialization Parameters for an OLE DB-based Data Source

You can only set these parameters on the Windows NT platform.

Specify a data link (UDL) that has previously been defined:

*`SET|PRIVATE|SET PRIVATE`* `HS_FDS_CONNECT_INFO="UDLFILE=`*`data_link`*`"`

Or, specify the connection details directly:

*SET|PRIVATE|SET PRIVATE* HS_FDS_CONNECT_INFO="*provider*;*db*[,CATALOG=*catalog*]"

where:

| | |
|---|---|
| *provider* | is the name of the provider as it appears in the registry. The value is case sensitive. |
| *db* | is the name of the database |
| *catalog* | is the name of the catalog |

**Note:** If the parameter value includes an equal sign (=), then it must be surrounded by quotation marks.

HS_FDS_CONNECT_INFO is required for an OLE DB agent. Other initialization parameters have defaults or are optional. You can use the default values and omit the optional parameters, or you can specify the parameters with values tailored for your installation.

**Note:** Before deciding to accept the default values or change them, see Appendix A, "Heterogeneous Services Initialization Parameters" for detailed information on all the initialization parameters.

## ODBC Connectivity Requirements

To use an ODBC agent, you must have an ODBC driver installed on the same machine as the Oracle database server. On Windows NT, you must have an ODBC driver manager also located on the same machine. The ODBC driver manager and driver must meet the following requirements:

- On Windows NT machines, a thread-safe, 32-bit ODBC driver Version 2.x or 3.x is required. You can use the native driver manager supplied with your Windows NT system.
- On UNIX machines, ODBC driver Version 2.5 is required. A driver manager is not required.

The ODBC driver and driver manager on Windows NT must conform to ODBC API conformance Level 1 or higher. If the ODBC driver or driver manager does not support multiple active ODBC cursors, then it restricts the complexity of SQL statements that you can execute using generic connectivity.

The ODBC driver you use must support all of the core SQL ODBC datatypes and should support SQL grammer level SQL_92. The ODBC driver should also expose the following ODBC APIs:

***Table 3–1 ODBC Functions*** **(Page 1 of 2)**

| ODBC Function | Comment |
|---|---|
| SQLAllocConnect | |
| SQLAllocEnv | |
| SQLAllocStmt | |
| SQLBindCol | |
| SQLBindParameter | |
| SQLColumns | |
| SQLConnect | |
| SQLDescribeCol | |
| SQLDisconnect | |
| SQLDriverConnect | |
| SQLError | |
| SQLExecDirect | |
| SQLExecute | |
| SQLExtendedFetch | Recommended if used by the non-Oracle system. |
| SQLFetch | |
| SQLForeignKeys | Recommended if used by the non-Oracle system. |
| SQLFreeConnect | |
| SQLFreeEnv | |
| SQLFreeStmt | |
| SQLGetConnectOption | |
| SQLGetData | |
| SQLGetFunctions | |
| SQLGetInfo | |
| SQLGetTypeInfo | |

*Table 3–1 ODBC Functions* **(Page 2 of 2)**

| ODBC Function | Comment |
|---|---|
| SQLNumParams | Recommended if used by the non-Oracle system. |
| SQLNumResultCols | |
| SQLParamData | |
| SQLPrepare | |
| SQLPrimaryKeys | Recommended if used by the non-Oracle system. |
| SQLProcedureColumns | Recommended if used by the non-Oracle system. |
| SQLProcedures | Recommended if used by the non-Oracle system. |
| SQLPutData | |
| SQLRowCount | |
| SQLSetConnectOption | |
| SQLSetStmtOption | |
| SQLStatistics | |
| SQLTables | |
| SQLTransact | Recommended if used by the non-Oracle system. |

# OLE DB (SQL) Connectivity Requirements

These requirements apply to OLE DB data providers that have an SQL processing capability and expose the OLE DB interfaces.

Generic connectivity passes the username and password to the provider when calling `IDBInitialize::Initialize()`.

OLE DB (SQL) connectivity requires that the data provider expose the following OLE DB interfaces:

*Table 3–2 OLE DB (SQL) Interfaces*

| Interface | Methods |
|---|---|
| IAccessor | CreateAccessor, ReleaseAccessor |
| IColumnsInfo | GetColumnsInfo (Command and Rowset objects) |
| ICommand | Execute |

*Table 3–2 OLE DB (SQL) Interfaces*

| Interface | Methods |
|---|---|
| ICommandPrepare | Prepare |
| ICommandProperties | SetProperties |
| ICommandText | SetCommandText |
| ICommandWithParameters | GetParameterInfo |
| IDBCreateCommand | CreateCommand |
| IDBCreateSession | CreateSession |
| IDBInitialize | Initialize |
| IDBSchemaRowset | GetRowset (tables, columns, indexes; optionally also procedures, procedure parameters) |
| IErrorInfo[1] | GetDescription, GetSource |
| IErrorRecords | GetErrorInfo |
| ILockBytes (OLE)[2] | Flush, ReadAt, SetSize, Stat, WriteAt |
| IRowset | GetData, GetNextRows, ReleaseRows, RestartPosition |
| IStream (OLE) | Read, Seek, SetSize, Stat, Write |
| ISupportErrorInfo | InterfaceSupportsErrorInfo |
| ITransactionLocal (optional) | StartTransaction, Commit, Abort |

[1] You can also use IErrorLookup with the GetErrorDescription method.

[2] Required only if BLOBs are used in the OLE DB provider.

# OLE DB (FS) Connectivity Requirements

These requirements apply to OLE DB data providers that do not have SQL processing capabilities. If the provider exposes them, then OLE DB (FS) connectivity uses OLE DB Index interfaces.

OLE DB (FS) connectivity requires that the data provider expose the following OLE DB interfaces:

***Table 3–3 OLE DB (FS) Interfaces***

| Interface | Methods |
|---|---|
| IAccessor | CreateAccessor, ReleaseAccessor |
| IColumnsInfo | GetColumnsInfo (Command and Rowset objects) |
| IOpenRowset | OpenRowset |
| IDBCreateSession | CreateSession |
| IRowsetChange | DeleteRows, SetData, InsertRow |
| IRowsetLocate | GetRowsByBookmark |
| IRowsetUpdate | Update (optional) |
| IDBInitialize | Initialize, Uninitialize |
| IDBSchemaRowset | GetRowset (tables, columns, indexes; optionally also procedures, procedure parameters) |
| ILockBytes (OLE)[1] | Flush, ReadAt, SetSize, Stat, WriteAt |
| IRowsetIndex[2] | SetRange |
| IErrorInfo[3] | GetDescription, GetSource |
| IErrorRecords | GetErrorInfo |
| IRowset | GetData, GetNextRows, ReleaseRows, RestartPosition |
| IStream (OLE) | Read, Seek, SetSize, Stat, Write |
| ITransactionLocal (optional) | StartTransaction, Commit, Abort |
| ISupportErrorInfo | InterfaceSupportsErrorInfo |
| ITableDefinition | CreateTable, DropTable |
| IDBProperties | SetProperties |

[1] Required only if BLOBs are used in the OLE DB provider.

[2] Required only if indexes are used in the OLE DB provider.

[3] You can use IErrorLookup with the GetErrorDescription method as well.

Because OLE DB (FS) connectivity is generic, it can connect to a number of different data providers that expose OLE DB interfaces. Every such data provider must meet the certain requirements.

> **Note:** The data provider must expose bookmarks. This enables tables to be updated. Without bookmarks being exposed, the tables are read-only.

## Data Source Properties

The OLE DB data source must support the following initialization properties:

- DBPROP_INIT_DATASOURCE
- DBPROP_AUTH_USERID

> **Note:** Required if the userid has been supplied in the security file

- DBPROP_AUTH_PASSWORD

> **Note:** Required if the userid and password have been supplied in the security file

The OLE DB data source must also support the following rowset properties:

- DBPROP_IRowsetChange = TRUE
- DBPROP_UPDATABILITY = CHANGE+DELETE+INSERT
- DBPROP_OWNUPDATEDELETE = TRUE
- DBPROP_OWNINSERT = TRUE
- DBPROP_OTHERUPDATEDELETE = TRUE
- DBPROP_CANSCROLLBACKWARDS = TRUE
- DBPROP_IRowsetLocate = TRUE
- DBPROP_OTHERINSERT = FALSE

# 4

# Developing Applications with Heterogeneous Services

This chapter provides information for application developers who want to use Heterogeneous Services.

This chapter contains these topics:


- Developing Applications with Heterogeneous Services: Overview
- Developing Applications Using Pass-Through SQL
- Optimizing Data Transfers Using Bulk Fetch
- Researching the Locking Behavior of Non-Oracle Systems
- Limitations to Heterogeneous Services

# Developing Applications with Heterogeneous Services: Overview

When you develop applications with Heterogeneous Services, Heterogeneous Services makes the non-Oracle system appear as if it were another Oracle database server.

However, you may sometimes need to access a non-Oracle system using the non-Oracle system's SQL dialect. To make access possible, Heterogeneous Services provides a pass-through SQL feature that allows you to directly execute a native SQL statement at the non-Oracle system.

Additionally, Heterogeneous Services supports bulk fetches to optimize the data transfers for large data sets between a non-Oracle system, agent and Oracle database server. This chapter also discusses how to tune such data transfers.

# Developing Applications Using Pass-Through SQL

The pass-through SQL feature allows you to send a statement directly to a non-Oracle system without being interpreted by the Oracle database server. This feature can be useful if the non-Oracle system allows for operations in statements for which there is no equivalent in Oracle.

This section contains the following topics:

- Using the DBMS_HS_PASSTHROUGH package
- Considering the Implications of Using Pass-Through SQL
- Executing Pass-Through SQL Statements

## Using the DBMS_HS_PASSTHROUGH package

You can execute these statements directly at the non-Oracle system using the PL/SQL package DBMS_HS_PASSTHROUGH. Any statement executed with the pass-through package is executed in the same transaction as standard SQL statements.

The DBMS_HS_PASSTHROUGH package conceptually resides at the non-Oracle system. You must invoke procedures and functions in the package by using the appropriate database link to the non-Oracle system.

> **See Also:** *Oracle8i Supplied PL/SQL Packages Reference* for more information about this package.

## Considering the Implications of Using Pass-Through SQL

When you execute a pass-through SQL statement that implicitly commits or rolls back a transaction in the non-Oracle system, the transaction is affected. For example, some systems implicitly commit the transaction containing a DDL statement. Because the Oracle database server is bypassed, the Oracle database server is unaware of the commit in the non-Oracle system. Consequently, the data at the non-Oracle system can be committed while the transaction in the Oracle database server is not.

If the transaction in the Oracle database server is rolled back, data inconsistencies between the Oracle database server and the non-Oracle server can occur. This situation results in global data inconsistency.

Note that if the application executes a regular COMMIT statement, the Oracle database server can coordinate the distributed transaction with the non-Oracle system. The statement executed with the pass-through facility is part of the distributed transaction.

## Executing Pass-Through SQL Statements

Table 4–1 shows the functions and procedures provided by the DBMS_HS_PASSTHROUGH package that allow you to execute pass-through SQL statements.

***Table 4–1 DBMS_HS_PASSTHROUGH Procedures and Functions***

| Procedure/Function | Description |
|---|---|
| OPEN_CURSOR | Opens a cursor |
| CLOSE_CURSOR | Closes a cursor |
| PARSE | Parses the statement |
| BIND_VARIABLE | Binds IN variables |
| BIND_OUT_VARIABLE | Binds OUT variables |
| BIND_INOUT_VARIABLE | Binds IN OUT variables |
| EXECUTE_NON_QUERY | Executes non-query |
| EXECUTE_IMMEDIATE | Executes non-query without bind variables |
| FETCH_ROW | Fetches rows from query |
| GET_VALUE | Retrieves column value from SELECT statement or retrieves OUT bind parameters |

This section contains these topics:


- Executing Non-Queries
- Executing Queries


## Executing Non-Queries

Non-queries include the following statements and types of statements:

- INSERT
- UPDATE
- DELETE
- DDL

To execute non-query statements, use the EXECUTE_IMMEDIATE function. For example, to execute a DDL statement at a non-Oracle system that you can access using the database link SalesDB, execute:

```
DECLARE
  num_rows INTEGER;

BEGIN
  num_rows := DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@SalesDB
            ('CREATE TABLE DEPT (n SMALLINT, loc CHARACTER(10))');
END;
```

The variable *num_rows* is assigned the number of rows affected by the execution. For DDL statements, zero is returned. Note that you cannot execute a query with EXECUTE_IMMEDIATE and you cannot use bind variables.

**Using Bind Variables: Overview** Bind variables allow you to use the same SQL statement multiple times with different values, reducing the number of times a SQL statement needs to be parsed. For example, if you need to insert four rows in a particular table, then you can parse the SQL statement once and bind and execute the SQL statement for each row. One SQL statement can have zero or more bind variables.

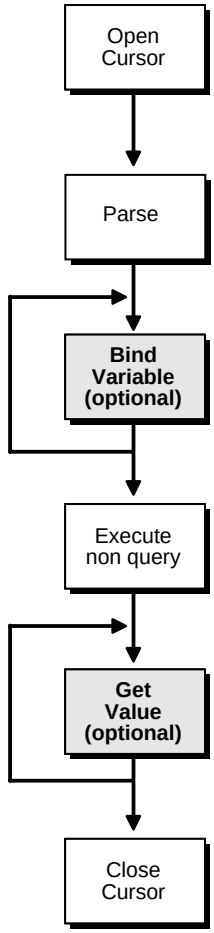
To execute pass-through SQL statements with bind variables, you must:

1. Open a cursor.
2. Parse the SQL statement at the non-Oracle system.
3. Bind the variables.

4. Execute the SQL statement at the non-Oracle system.
5. Close the cursor.

Figure 4–1 shows the flow diagram for executing non-queries with bind variables.

***Figure 4–1 Flow Diagram for Non-Query Pass-Through SQL***




**Using IN Bind Variables** The syntax of the non-Oracle system determines how a statement specifies a bind variable. For example, in Oracle you define bind variables with a preceding colon, as in:

```
UPDATE EMP
SET SAL=SAL*1.1
WHERE ENAME=:ename
```

In this statement `:ename` is the bind variable. In other non-Oracle systems you might need to specify bind variables with a question mark, as in:

```
UPDATE EMP
SET SAL=SAL*1.1
WHERE ENAME= ?
```

In the bind variable step, you must positionally associate host program variables (in this case, PL/SQL) with each of these bind variables.

For example, to execute the previous statement, you can use the following PL/SQL program:

```
DECLARE
  c INTEGER;
  nr INTEGER;
BEGIN
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@SalesDB;
  DBMS_HS_PASSTHROUGH.PARSE@SalesDB(c,
        'UPDATE EMP SET SAL=SAL*1.1 WHERE ENAME=?');
  DBMS_HS_PASSTHROUGH.BIND_VARIABLE(c,1,'JONES');
  nr:=DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY@SalesDB(c);
  DBMS_OUTPUT.PUT_LINE(nr||' rows updated');
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@salesDB(c);
END;
```

**Using OUT Bind Variables** In some cases, the non-Oracle system can also support OUT bind variables. With OUT bind variables, the value of the bind variable is not known until *after* the execution of the SQL statement.

Although OUT bind variables are populated after the SQL statement is executed, the non-Oracle system must know that the particular bind variable is an OUT bind variable *before* the SQL statement is executed. You must use the BIND_OUT_VARIABLE procedure to specify that the bind variable is an OUT bind variable.

After the SQL statement is executed, you can retrieve the value of the OUT bind variable using the GET_VALUE procedure.

**Using IN OUT Bind Variables** A bind variable can be both an IN and an OUT variable. This means that the value of the bind variable must be known before the SQL statement is executed but can be changed after the SQL statement is executed.

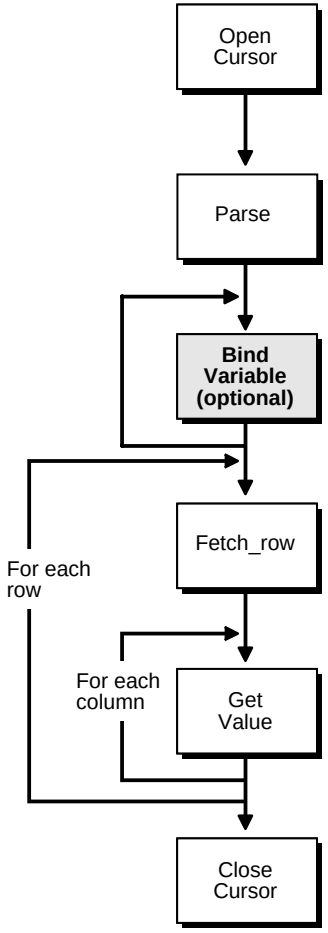
For IN OUT bind variables, you must use the BIND_INOUT_VARIABLE procedure to provide a value *before* the SQL statement is executed. *After* the SQL statement is executed, you must use the GET_VALUE procedure to retrieve the new value of the bind variable.

### Executing Queries

The difference between queries and non-queries is that queries retrieve a result set from a SELECT statement. The result set is retrieved by iterating over a cursor.

Figure 4–2 illustrates the steps in a pass-through SQL query. After the system parses the SELECT statement, each row of the result set can be fetched with the FETCH_ROW procedure. After the row is fetched, use the GET_VALUE procedure to retrieve the select list items into program variables. After all rows are fetched you can close the cursor.

***Figure 4–2 Pass-through SQL for Queries***




You do not have to fetch all the rows. You can close the cursor at any time after opening the cursor, for example, after fetching a few rows.

---

**Note:** Although you are fetching one row at a time, Heterogeneous Services optimizes the round trips between the Oracle database server and the non-Oracle system by buffering multiple rows and fetching from the non-Oracle data system in one round trip.

---

The following example executes a query:

```
DECLARE
   val  VARCHAR2(100);
   c    INTEGER;
   nr   INTEGER;
BEGIN
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@SalesDB;
  DBMS_HS_PASSTHROUGH.PARSE@SalesDB(c,
    'select ename
     from   emp
     where  deptno=10');
  LOOP
    nr := DBMS_HS_PASSTHROUGH.FETCH_ROW@SalesDB(c);
    EXIT WHEN nr = 0;
    DBMS_HS_PASSTHROUGH.GET_VALUE@SalesDB(c, 1, val);
    DBMS_OUTPUT.PUT_LINE(val);
  END LOOP;
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@SalesDB(c);
END;
```

After parsing the SELECT statement, the rows are fetched and printed in a loop until the function FETCH_ROW returns the value 0.

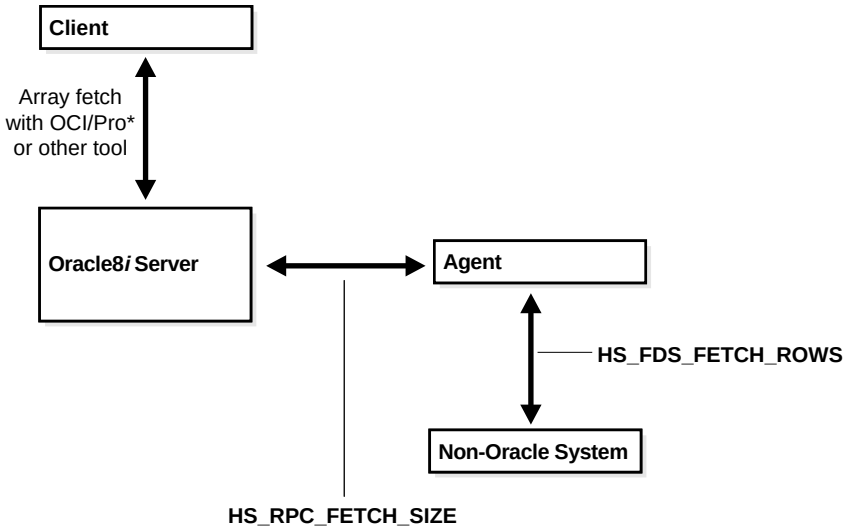
## Optimizing Data Transfers Using Bulk Fetch

When an application fetches data from a non-Oracle system using Heterogeneous Services, data is transferred:

- From the non-Oracle system to the agent process
- From the agent process to the Oracle database server
- From the Oracle database server to the application

Oracle allows you to optimize all three data transfers, as illustrated in Figure 4–3.

*Figure 4–3 Optimizing data transfers*




This section contains the following topics:

- Using OCI, an Oracle Precompiler, or Another Tool for Array Fetches
- Controlling the Array Fetch Between Oracle Database Server and Agent
- Controlling the Array Fetch Between Agent and Non-Oracle Server
- Controlling the Reblocking of Array Fetches

## Using OCI, an Oracle Precompiler, or Another Tool for Array Fetches

You can optimize data transfers between your application and the Oracle database server by using array fetches. See your application development tool documentation for information about array fetching and how to specify the amount of data to be transferred over the network.

## Controlling the Array Fetch Between Oracle Database Server and Agent

When Oracle retrieves data from a non-Oracle system, the Heterogeneous Services initialization parameter HS_RPC_FETCH_SIZE defines the number of bytes sent per fetch between the agent and the Oracle database server. The agent fetches data from the non-Oracle system until one of the following occurs:

- It has accumulated the specified number of bytes to send back to the Oracle database server
- The last row of the result set is fetched from the non-Oracle system

## Controlling the Array Fetch Between Agent and Non-Oracle Server

The initialization parameter HS_FDS_FETCH_ROWS determines the number of rows to be retrieved from a non-Oracle system. Note that the array fetch must be supported by the agent. See your agent-specific documentation to ensure that your agent supports array fetching.

## Controlling the Reblocking of Array Fetches

By default, an agent fetches data from the non-Oracle system until it has enough data retrieved to send back to the server. That is, it continues processing until the number of bytes fetched from the non-Oracle system is equal to or higher than the value of HS_RPC_FETCH_SIZE. In other words, the agent *reblocks* the data between the agent and the Oracle database server in sizes defined by the value of HS_RPC_FETCH_SIZE.

When the non-Oracle system supports array fetches, you can immediately send the data fetched from the non-Oracle system by the array fetch to the Oracle database server without waiting until the exact value of HS_RPC_FETCH_SIZE is reached. That is, you can stream the data from the non-Oracle system to the Oracle database server and disable reblocking by setting the value of initialization parameter HS_RPC_FETCH_REBLOCKING to OFF.

For example, assume that you set HS_RPC_FETCH_SIZE to 64K and HS_FDS_FETCH_ROWS to 100 rows. Assume that each row is approximately 600 bytes in size, so that the 100 rows are approximately 60K. When HS_RPC_FETCH_REBLOCKING is set to ON, the agent starts fetching 100 rows from the non-Oracle system.

Because there is only 60K bytes of data in the agent, the agent does not send the data back to the Oracle database server. Instead, the agent fetches the next 100 rows from the non-Oracle system. Because there is now 120K of data in the agent, the first 64K can be sent back to the Oracle database server.

Now there is 56K of data left in the agent. The agent fetches another 100 rows from the non-Oracle system before sending the next 64K of data to the Oracle database server. By setting the initialization parameter HS_RPC_FETCH_REBLOCKING to OFF, the first 100 rows are immediately sent back to the Oracle database server.

## Researching the Locking Behavior of Non-Oracle Systems

When designing applications with Heterogeneous Services, be aware that the Oracle database server and non-Oracle data sources can have different locking behaviors. For example, some non-Oracle data sources differ from the Oracle database server in how they set read and write locks on records in affected tables.

Oracle cannot change any aspect of the locking behavior of a non-Oracle data source. In order to avoid adverse effects on other users of the non-Oracle data source, all applications that access a non-Oracle data source must always adhere to the programming standards of that data source.

> **See Also:** Your non-Oracle system's documentation for information about locking behavior.

## Limitations to Heterogeneous Services

Limitations to Heterogeneous Services include the following:

- Long columns are limited to 4 MB in size
- There is no support for the SQL*Plus DESCRIBE command
- There is no support for LOBs and object types
- Shared database links cannot be used with Heterogeneous Services
- There is no support for PL/SQL records
- There is no support for ref cursors
- Each specific gateway might have its own limitations

# A

# Heterogeneous Services Initialization Parameters

Heterogeneous Services parameters are distinct from Oracle database server initialization parameters. Set Heterogeneous Services parameters by editing the Oracle Transparent Gateway initialization file, or by using the DBMS_HS package to set them in the data dictionary. String values for Heterogeneous Services parameters must be lowercase.

This appendix contains the following topics:


- HS_COMMIT_POINT_STRENGTH
- HS_DB_DOMAIN
- HS_DB_INTERNAL_NAME
- HS_DB_NAME
- HS_DESCRIBE_CACHE_HWM
- HS_FDS_CONNECT_INFO
- HS_FDS_SHAREABLE_NAME
- HS_FDS_TRACE_LEVEL
- HS_FDS_TRACE_FILE_NAME
- HS_LANGUAGE
- HS_NLS_DATE_FORMAT
- HS_NLS_DATE_LANGUAGE
- HS_NLS_NCHAR
- HS_OPEN_CURSORS
- HS_ROWID_CACHE_SIZE

- HS_RPC_FETCH_REBLOCKING
- HS_RPC_FETCH_SIZE

## HS_COMMIT_POINT_STRENGTH

| | |
|---|---|
| **Default value:** | 0 |
| **Range of values:** | 0 to 255 |

Specifies a value that determines the commit point site in a heterogeneous distributed transaction. HS_COMMIT_POINT_STRENGTH is similar to COMMIT_POINT_STRENGTH, described in the *Oracle8i Reference*.

Set HS_COMMIT_POINT_STRENGTH to a value relative to the importance of the site that is the commit point site in a distributed transaction. The Oracle database server or non-Oracle system with the highest commit point strength becomes the commit point site. To ensure that a non-Oracle system *never* becomes the commit point site, set the value of HS_COMMIT_POINT_STRENGTH to zero.

HS_COMMIT_POINT_STRENGTH is important only if the non-Oracle system can participate in the two-phase protocol as a regular two-phase commit partner and as the commit point site. This is only the case if the transaction model is two-phase commit confirm (2PCC).

> **See Also:** Chapter 2, "Managing Heterogeneous Services" for more information about heterogeneous distributed transactions.

## HS_DB_DOMAIN

| | |
|---|---|
| **Default value:** | WORLD |
| **Range of values:** | 1 to 119 characters |

Specifies a unique network sub-address for a non-Oracle system. HS_DB_DOMAIN is similar to DB_DOMAIN, described in the *Oracle8i Administrator's Guide* and the *Oracle8i Reference*. HS_DB_DOMAIN is required if you use the Oracle Names server. HS_DB_NAME and HS_DB_DOMAIN define the global name of the non-Oracle system.

> **Note:** HS_DB_NAME and HS_DB_DOMAIN must combine to form a unique address.

## HS_DB_INTERNAL_NAME

| | |
|---|---|
| **Default value:** | 01010101 |
| **Range of values:** | 1 to 16 hexadecimal characters |

Specifies a unique hexadecimal number identifying the instance to which the Heterogeneous Services agent is connected. This parameter's value is used as part of a transaction ID when global name services are activated. Specifying a non-unique number can cause problems when two-phase commit recovery actions are necessary for a transaction.

## HS_DB_NAME

| | |
|---|---|
| **Default value:** | HO |
| **Range of values:** | 1 to 8 lowercase characters |

Specifies a unique alphanumeric name for the data store given to the non-Oracle system. This name identifies the non-Oracle system within the cooperative server environment. HS_DB_NAME and HS_DB_DOMAIN define the global name of the non-Oracle system.

## HS_DESCRIBE_CACHE_HWM

| | |
|---|---|
| **Default value:** | 100 |
| **Range of values:** | 1 to 4000 |

Specifies the maximum number of entries in the describe cache used by Heterogeneous Services. This limit is known as the describe cache high water mark. The cache contains descriptions of the mapped tables that Heterogeneous Services reuses so that it does not have to re-access the non-Oracle data store.

If you are accessing many mapped tables, then increase the high water mark to improve performance. Note that increasing the high water mark improves performance at the cost of memory usage.

# HS_FDS_CONNECT_INFO

| | |
|---|---|
| **Default value:** | none |
| **Range of values:** | not applicable |

Specifies the information needed to bind to the data provider, that is, the non-Oracle system. For generic connectivity, you can bind to an ODBC-based data source or to an OLE DB-based data source. The information that you provide depends on the platform and whether the data source is ODBC or OLE DB-based.

This parameter is required if you are using generic connectivity.

**ODBC-based Data Source on Windows:** You can use either a File DSN or a System DSN as follows:

- When connecting using a File DSN the parameter format is:

  `HS_FDS_CONNECT_INFO=FILEDSN=`*`file_dsn`*

- When connecting using a System DSN the parameter format is:

  `HS_FDS_CONNECT_INFO=`*`system_dsn`*

If you are connecting to the data source through the driver for that data source, then precede the DSN by the name of the driver, followed by a semi-colon (;).

**ODBC-based Data Source on UNIX:** Use a DSN with the following format:

`HS_FDS_CONNECT_INFO=dsn`

**OLE DB-based Data Source (Windows NT Only):** Use a universal data link (UDL) with the following formats:

- HS_FDS_CONNECT_INFO="UDLFILE=*data_link*"
- HS_FDS_CONNECT_INFO="*data_link_provider;db*[,CATALOG=*catalog*]"

  which allows you to specify the connection details directly, and where:

  – *data_link_provider* is the case-sensitive name of the provider as it appears in the registry

  – *db* is the name of the database

– *catalog* is the name of the catalog

**Note:** Whenever the parameter value includes an equal sign (=), it must be enclosed in quotation marks.

## HS_FDS_SHAREABLE_NAME

**Default value:** none

**Range of values:** not applicable

Specifies the full path name to the ODBC library. This parameter is required when you are using generic connectivity to access data from an ODBC provider on a UNIX machine.

## HS_FDS_TRACE_LEVEL

**Default value:** OFF

**Range of values:** ON or OFF

Specifies whether error tracing is enabled or disabled for generic connectivity. Enable the tracing to see which error messages occur when you encounter problems. The results are written to a generic connectivity log file, in the `/log` directory under the `$ORACLE_HOME` directory.

## HS_FDS_TRACE_FILE_NAME

**Default value:** none

**Range of values:** not applicable

Specifies the name of the trace file to which generic connectivity error messages are written, if TRACE is enabled. The trace file is located in the `LOG` directory under the `$ORACLE_HOME` directory.

# HS_LANGUAGE

| | |
|---|---|
| **Default value:** | System-specific |
| **Range of values:** | Any valid language name (up to 255 characters) |

Provides Heterogeneous Services with character set, language, and territory information of the non-Oracle data source. The value must use the following format:

*language*[*_territory.character_set*]

> **Note:** The national language support initialization parameters affect error messages, the data for the SQL Service, and parameters in distributed external procedures.

## Character sets

Ideally, the character sets of the Oracle database server and the non-Oracle data source are the same. If they are not the same, Heterogeneous Services attempts to translate the character set of the non-Oracle data source to the Oracle database character set, and back again. The translation can degrade performance. In some cases, Heterogeneous Services cannot translate a character from one character set to another.

> **Note:** The specified character set must be a superset of the operating system character set on the platform where the agent is installed.

## Language

The language component of the HS_LANGUAGE initialization parameter determines:

- Day and month names of dates
- AD, BC, PM, and AM symbols for date and time
- Default sorting mechanism

Note that HS_LANGUAGE does not determine the language for error messages for the generic Heterogeneous Services messages (ORA-25000 through ORA-28000). These are controlled by the session settings in the Oracle database server.

---

**Note:** Use the HS_NLS_DATE_LANGUAGE initialization parameter to set the day and month names, and the AD, BC, PM, and AM symbols for dates and time independently from the language.

---

### Territory

The territory clause specifies the conventions for day and week numbering, default date format, decimal character and group separator, and ISO and local currency symbols. Note that:

- You can override the date format using the initialization parameter HS_NLS_DATE_FORMAT.
- The level of National Language Support between the Oracle database server and the non-Oracle data source depends on how the driver is implemented. See the installation documentation for your platform for more information about the level of National Language Support.

## HS_NLS_DATE_FORMAT

| | |
|---|---|
| **Default value:** | Value determined by HS_LANGUAGE parameter |
| **Range of values:** | Any valid date format mask (up to 255 characters) |

Defines the date format for dates used by the target system. This parameter has the same function as the NLS_DATE_FORMAT parameter for an Oracle database server. The value of can be any valid date mask listed in the *Oracle8i Reference*, but must match the date format of the target system. For example, if the target system stores the date February 14, 1995 as 1995/02/14, set the parameter to yyyy/mm/dd. Note that characters must be lowercase.

## HS_NLS_DATE_LANGUAGE

| | |
|---|---|
| **Default value:** | Value determined by HS_LANGUAGE parameter |

| | |
|---|---|
| **Range of values:** | Any valid NLS_LANGUAGE value (up to 255 characters) |

Specifies the language used in character date values coming from the non-Oracle system. Date formats can be language independent. For example, if the format is dd/mm/yyyy, all three components of the character date are numbers. In the format dd-mon-yyyy, however, the month component is the name abbreviated to three characters. The abbreviation is very much language dependent. For example, the abbreviation for the month April is "apr", which in French is "avr" (Avril).

Heterogeneous Services assumes that character date values fetched from the non-Oracle system are in this format. Also, Heterogeneous Services sends character date bind values in this format to the non-Oracle system.

## HS_NLS_NCHAR

| | |
|---|---|
| **Default value:** | Value determined by HS_LANGUAGE parameter |
| **Range of values:** | Any valid national character set (up to 255 characters) |

Informs Heterogeneous Services of the value of the national character set of the non-Oracle data source. This value is the non-Oracle equivalent to the NATIONAL CHARACTER SET parameter setting in the Oracle CREATE DATABASE statement. The HS_NLS_NCHAR value should be the character set ID of a character set supported by the Oracle NLSRTL library.

**See Also:** HS_LANGUAGE on page A-7.

## HS_OPEN_CURSORS

| | |
|---|---|
| **Default value:** | 50 |
| **Range of values:** | 1 - value of Oracle's OPEN_CURSORS initialization parameter |

Defines the maximum number of cursors that can be open on one connection to a non-Oracle system instance.

The value never exceeds the number of open cursors in the Oracle database server. Therefore, setting the same value as the OPEN_CURSORS initialization parameter in the Oracle database server is recommended.

## HS_ROWID_CACHE_SIZE

| | |
|---|---|
| **Default value:** | 3 |
| **Range of values:** | 1 to 32767 |

Specifies the size of the Heterogeneous Services cache containing the non-Oracle system equivalent of ROWIDs. The cache contains non-Oracle system ROWIDs needed to support the WHERE CURRENT OF clause in a SQL statement or a SELECT FOR UPDATE statement.

When the cache is full, the first slot in the cache is reused, then the second, and so on. Only the last HS_ROWID_CACHE_SIZE non-Oracle system ROWIDs are cached.

## HS_RPC_FETCH_REBLOCKING

| | |
|---|---|
| **Default value:** | ON |
| **Range of values:** | OFF, ON |

Controls whether Heterogeneous Services attempts to optimize performance of data transfer between the Oracle database server and the HS agent connected to the non-Oracle data store.

The following values are possible:

- OFF disables reblocking of fetched data so that data is immediately sent from agent to server
- ON enables reblocking, which means that data fetched from the non-Oracle system is buffered in the agent and is not sent to the Oracle database server until the amount of fetched data is equal or higher than HS_RPC_FETCH_SIZE. However, any buffered data is returned immediately when a fetch indicates that no more data exists or when the non-Oracle system reports an error.

# HS_RPC_FETCH_SIZE

| | |
|---|---|
| **Default value:** | 4000 |
| **Range of values:** | Decimal integer (byte count) |

Tunes internal data buffering to optimize the data transfer rate between the server and the agent process.

Increasing the value can reduce the number of network round trips needed to transfer a given amount of data, but also tends to increase data bandwidth and to reduce response time or *latency* as measured between issuing a query and completion of all fetches for the query. Nevertheless, increasing the fetch size can increase latency for the initial fetch results of a query, because the first fetch results are not transmitted until additional data is available.

**See Also:** Chapter 4, "Developing Applications with Heterogeneous Services" for more information.

# B

# Heterogeneous Services Data Dictionary Views

Heterogeneous Services mapping supports the following list of data dictionary views:

- ALL_CATALOG
- ALL_COL_COMMENTS
- ALL_COL_PRIVS
- ALL_COL_PRIVS_MADE
- ALL_COL_PRIVS_RECD
- ALL_CONSTRAINTS
- ALL_CONS_COLUMNS
- ALL_DB_LINKS
- ALL_DEF_AUDIT_OPTS
- ALL_DEPENDENCIES
- ALL_ERRORS
- ALL_INDEXES
- ALL_IND_COLUMNS
- ALL_OBJECTS
- ALL_SEQUENCES
- ALL_SNAPSHOTS
- ALL_SOURCE

- ALL_SYNONYMS
- ALL_TABLES
- ALL_TAB_COLUMNS
- ALL_TAB_COMMENTS
- ALL_TAB_PRIVS
- ALL_TAB_PRIVS_MADE
- ALL_TAB_PRIVS_RECD
- ALL_TRIGGERS
- ALL_USERS
- ALL_VIEWS
- AUDIT_ACTIONS
- COLUMN_PRIVILEGES
- DBA_CATALOG
- DBA_COL_COMMENTS
- DBA_COL_PRIVS
- DBA_OBJECTS
- DBA_ROLES
- DBA_ROLE_PRIVS
- DBA_SYS_PRIVS
- DBA_TABLES
- DBA_TAB_COLUMNS
- DBA_TAB_COMMENTS
- DBA_TAB_PRIVS
- DBA_USERS
- DICTIONARY
- DICT_COLUMNS
- DUAL
- INDEX_STATS

- PRODUCT_USER_PROFILE
- RESOURCE_COST
- ROLE_ROLE_PRIVS
- ROLE_SYS_PRIVS
- ROLE_TAB_PRIVS
- SESSION_PRIVS
- SESSION_ROLES
- TABLE_PRIVILEGES
- USER_AUDIT_OBJECT
- USER_AUDIT_SESSION
- USER_AUDIT_STATEMENT
- USER_AUDIT_TRAIL
- USER_CATALOG
- USER_CLUSTERS
- USER_CLU_COLUMNS
- USER_COL_COMMENTS
- USER_COL_PRIVS
- USER_COL_PRIVS_MADE
- USER_COL_PRIVS_RECD
- USER_CONSTRAINTS
- USER_CONS_COLUMNS
- USER_DB_LINKS
- USER_DEPENDENCIES
- USER_ERRORS
- USER_EXTENTS
- USER_FREE_SPACE
- USER_INDEXES
- USER_IND_COLUMNS

- USER_OBJECTS
- USER_OBJ_AUDIT_OPTS
- USER_RESOURCE_LIMITS
- USER_ROLE_PRIVS
- USER_SEGMENTS
- USER_SEQUENCES
- USER_SNAPSHOT_LOGS
- USER_SOURCE
- USER_SYNONYMS
- USER_SYS_PRIVS
- USER_TABLES
- USER_TABLESPACES
- USER_TAB_COLUMNS
- USER_TAB_COMMENTS
- USER_TAB_PRIVS
- USER_TAB_PRIVS_MADE
- USER_TAB_PRIVS_RECD
- USER_TRIGGERS
- USER_TS_QUOTAS
- USER_USERS
- USER_VIEWS

# C

# DBMS_HS_PASSTHROUGH for Pass-Through SQL

The package DBMS_HS_PASSTHROUGH contains the procedures and functions for pass-through SQL of Heterogeneous Services.

This appendix contains these topics:


- BIND_VARIABLE procedure
- BIND_VARIABLE_RAW procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_INOUT_VARIABLE procedure
- BIND_INOUT_VARIABLE_RAW procedure
- CLOSE_CURSOR function
- EXECUTE_IMMEDIATE function
- EXECUTE_NON_QUERY function
- FETCH_ROW function
- GET_VALUE procedure
- GET_VALUE_RAW procedure
- OPEN_CURSOR function
- PARSE procedure


**See Also:** Chapter 4, "Developing Applications with Heterogeneous Services" for more information about this package.

# Summary of Subprograms

*Table C–1 DBMS_HS Package Subprograms*

| Subprogram | Description |
|---|---|
| BIND_VARIABLE procedure | Binds an IN variable positionally with a PL/SQL program variable |
| BIND_VARIABLE_RAW procedure | Binds IN variables of type RAW |
| BIND_OUT_VARIABLE procedure | Binds an OUT variable with a PL/SQL program variable |
| BIND_OUT_VARIABLE_ RAW procedure | Binds an OUT variable of datatype RAW with a PL/SQL program variable |
| BIND_INOUT_VARIABLE procedure | Binds IN OUT bind variables |
| BIND_INOUT_VARIABLE_ RAW procedure | Binds IN OUT bind variables of datatype RAW |
| CLOSE_CURSOR function | Closes the cursor and releases associated memory after the SQL statement has been executed at the non-Oracle system |
| EXECUTE_IMMEDIATE function | Executes a SQL statement immediately |
| EXECUTE_NON_QUERY function | Executes any SQL statement other than a SELECT statement |
| FETCH_ROW function | Fetches rows from a result set |
| GET_VALUE procedure | Retrieves the select list items of SELECT statements after a row has been fetched, and retrieves the OUT bind values after the SQL statement has been executed |
| GET_VALUE_RAW procedure | Retrieves the select list items of SELECT statements after a row has been fetched, and retrieves the OUT bind values after the SQL statement has been executed. This procedure operates on the RAW datatype |
| OPEN_CURSOR function | Opens a cursor for executing a pass-through SQL statement at the non-Oracle system |
| PARSE procedure | Parses a SQL statement at non-Oracle system |

## BIND_VARIABLE procedure

This procedure binds an IN variable positionally with a PL/SQL program variable. Syntax.

> **See Also:** Chapter 4, "Developing Applications with Heterogeneous Services".

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
     c      IN    BINARY_INTEGER NOT NULL,
     pos    IN    BINARY_INTEGER NOT NULL,
     val    IN    dty
    [,name  IN    VARCHAR2]);
```

Where *dty* is one of

- DATE
- NUMBER
- VARCHAR2

> **See Also:** BIND_VARIABLE_RAW procedure

### Parameters

***Table C–2 BIND_VARIABLE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. The cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE. |
| pos | Position of the bind variable in the SQL statement. Starts from 1 |
| val | Value that must be passed to the bind variable |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

### Exceptions

*Table C–3 BIND_VARIABLE Procedure Exceptions*

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid |
| ORA-28552 | The procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter |

### Pragmas

Purity levels defined: WNDS, RNDS

## BIND_VARIABLE_RAW procedure

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_OUT_VARIABLE procedure
- BIND_VARIABLE_RAW procedure

This procedure binds IN variables of type RAW.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
     c      IN    BINARY_INTEGER NOT NULL,
     pos    IN    BINARY_INTEGER NOT NULL,
     val    IN    RAW
    [,name  IN    VARCHAR2]);
```

## Parameters

***Table C–4 BIND_VARIABLE_RAW Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE. |
| pos | Position of the bind variable in the SQL statement. Starts from 1. |
| val | Value that must be passed to the bind variable |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

## Exceptions

***Table C–5 BIND_VARIABLE_RAW Procedure Exceptions***

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter |

## Pragmas

Purity level defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_VARIABLE procedure
- BIND_OUT_VARIABLE procedure

## BIND_OUT_VARIABLE procedure

This procedure binds an OUT variable with a PL/SQL program variable.

**See Also:** See Chapter 4, "Developing Applications with Heterogeneous Services" for more information about binding OUT parameters.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
     c          IN    BINARY_INTEGER NOT NULL,
     pos        IN    BINARY_INTEGER NOT NULL,
     val        OUT   dty,
     [,name     IN    VARCHAR2]);
```

Where *dty* is one of

- DATE
- NUMBER
- VARCHAR2

**See Also:** BIND_OUT_VARIABLE_RAW procedure for more information about OUT variables of datatype RAW.

## Parameters

***Table C–6 BIND_OUT_VARIABLE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |
| pos | Position of the bind variable in the SQL statement. Starts from 1. |
| val | Variable in which the OUT bind variable will store its value. The package will remember only the size of the variable. After the SQL statement is executed, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE. |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

## Exceptions

***Table C–7 BIND_OUT_VARIABLE Procedure Exceptions***

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_VARIABLE procedure
- BIND_VARIABLE_RAW procedure
- GET_VALUE procedure

## BIND_OUT_VARIABLE_RAW procedure

This procedure binds an OUT variable of datatype RAW with a PL/SQL program variable.

**See Also:** Chapter 4, "Developing Applications with Heterogeneous Services" for more information on binding OUT parameters.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
     c       IN    BINARY_INTEGER NOT NULL,
     pos     IN    BINARY_INTEGER NOT NULL,
     val     OUT   RAW,
     [,name  IN    VARCHAR2]);
```

### Parameters

***Table C–8 BIND_OUT_VARIABLE_RAW Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |
| pos | Position of the bind variable in the SQL statement. Starts from 1. |

*Table C–8 BIND_OUT_VARIABLE_RAW Procedure Parameters*

| Parameter | Description |
|---|---|
| val | Variable in which the OUT bind variable will store its value. The package will remember only the size of the variable. After the SQL statement is executed, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW. |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

## Exceptions

*Table C–9 BIND_OUT_VARIABLE_RAW Parameter Exceptions*

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

## Pragmas

Pragmas defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_OUT_VARIABLE procedure
- BIND_VARIABLE procedure
- BIND_VARIABLE_RAW procedure
- GET_VALUE procedure

## BIND_INOUT_VARIABLE procedure

This procedure binds IN OUT bind variables.

**See Also:** Chapter 4, "Developing Applications with Heterogeneous Services" for more information on binding IN OUT parameters.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
     c       IN     BINARY_INTEGER NOT NULL,
     pos     IN     BINARY_INTEGER NOT NULL,
     val     IN OUT dty,
     [,name  IN     VARCHAR2]);
```

Where *dty* is one of

- DATE
- NUMBER
- VARCHAR2

For binding IN OUT variables of datatype RAW, see BIND_INOUT_VARIABLE_RAW.

## Parameters

***Table C–10 BIND_INOUT_VARIABLE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |
| pos | Position of the bind variable in the SQL statement. Starts from 1. |
| val | This value will be used for two purposes:<br>■ To provide the IN value before the SQL statement is executed<br>■ To determine the size of the OUT value |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

## Exceptions

***Table C–11 BIND_INOUT_VARIABLE Procedure Exceptions***

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_INOUT_VARIABLE_RAW procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_VARIABLE procedure
- BIND_VARIABLE_RAW procedure
- GET_VALUE procedure

## BIND_INOUT_VARIABLE_RAW procedure

This procedure binds IN OUT bind variables of datatype RAW. See Syntax

**See Also :** Chapter 4, "Developing Applications with Heterogeneous Services" for more information on binding IN OUT parameters.

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
     c        IN     BINARY_INTEGER NOT NULL,
     pos      IN     BINARY_INTEGER NOT NULL,
     val      IN OUT RAW,
     [,name   IN     VARCHAR2]);
```

### Parameters

***Table C–12 BIND_INOUT_VARIABLE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |
| pos | Position of the bind variable in the SQL statement. Starts from 1. |

*Table C–12 BIND_INOUT_VARIABLE Procedure Parameters*

| Parameter | Description |
|---|---|
| val | This value will be used for two purposes:<br>■ To provide the IN value before the SQL statement is executed<br>■ To determine the size of the out value |
| name | Optional parameter to name the bind variable. For example, consider the following statement:<br>`SELECT * FROM emp WHERE ename=:ename;`<br>The position of the bind variable *:ename* is 1 and the name is *:ename*. You can use this parameter if the non-Oracle system supports named binds instead of positional binds. Note that passing the position is still required. |

## Exceptions

*Table C–13 BIND_INOUT_VARIABLE Procedure Exceptions*

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

## Pragmas

Pragmas defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- BIND_INOUT_VARIABLE procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_VARIABLE procedure
- BIND_VARIABLE_RAW procedure
- GET_VALUE procedure

## CLOSE_CURSOR function

This function closes the cursor and releases associated memory after the SQL statement has been executed at the non-Oracle system. If the cursor was not open, the operation is a no operation.

### Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
     c   IN   BINARY_INTEGER NOT NULL);
```

### Parameter

*Table C–14 CLOSE_CURSOR Procedure Parameters*

| Parameter | Description |
|---|---|
| c | Cursor to be released. |

### Exceptions

*Table C–15 CLOSE_CURSOR Procedure Exceptions*

| Exception | Description |
|---|---|
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS, RNDS

**See Also:** OPEN_CURSOR function

## EXECUTE_IMMEDIATE function

This function executes a SQL statement immediately. Any valid SQL command except SELECT can be executed immediately, but the statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally, the SQL statement is executed using the PASSTHROUGH SQL protocol sequence of OPEN_CURSOR, PARSE, EXECUTE_NON_QUERY, CLOSE_CURSOR.

### Syntax

```
EXECUTE_IMMEDIATE ( s IN VARCHAR2 NOT NULL )
RETURN BINARY_INTEGER;
```

### Parameter Description

*Table C–16 EXECUTE_IMMEDIATE Procedure Parameters*

| Parameter | Description |
|---|---|
| s | VARCHAR2 variable with the statement to be executed immediately. |

### Returns

The number of rows affected by the execution of the SQL statement.

### Exceptions

*Table C–17 EXECUTE_IMMEDIATE Procedure Exceptions*

| Exception | Description |
|---|---|
| ORA-28544 | Max open cursors. |
| ORA-28551 | SQL statement is invalid. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: NONE

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- EXECUTE_NON_QUERY function
- CLOSE_CURSOR function

## EXECUTE_NON_QUERY function

This function executes any SQL statement other than a SELECT statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be executed.

### Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
     c   IN   BINARY_INTEGER NOT NULL)
     RETURN   BINARY_INTEGER;
```

### Parameter

***Table C–18 EXECUTE_NON_QUERY Function Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |

### Returns

The number of rows affected by the SQL statement in the non-Oracle system.

### Exceptions

*Table C–19 EXECUTE_NON_QUERY Function Exceptions*

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | BIND_VARIABLE procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: NONE

**See Also:**

- OPEN_CURSOR function
- PARSE procedure

## FETCH_ROW function

This function fetches rows from a result set. The result set is defined with a SQL SELECT statement.

Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed. When there are no more rows to be fetched, the function returns 0. After a 0 return, the NO_DATA_FOUND exception occurs when:

- A subsequent FETCH_ROW is attempted
- A GET_VALUE is attempted

### Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
     c        IN    BINARY_INTEGER NOT NULL
     [,first  IN    BOOLEAN])
     RETURN  BINARY_INTEGER;
```

### Parameters and Descriptions

*Table C–20 FETCH_ROW Function Parameters*

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively. |
| first | Optional parameter to re-execute a SELECT statement. Possible values:<br>■ TRUE: re-execute SELECT statement.<br>■ FALSE: fetch the next row, or if executed for the first time execute and fetch rows (default). |

### Returns

The returns the number of rows fetched. The function will return 0 if the last row was already fetched.

### Exceptions

*Table C–21 FETCH_ROW Function Exceptions*

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor and parse the SQL statement? |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure

## GET_VALUE procedure

This procedure has two purposes:

- To retrieve the select list items of SELECT statements after a row has been fetched.
- To retrieve the OUT bind values after the SQL statement has been executed.

### Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
     c     IN    BINARY_INTEGER NOT NULL,
     pos   IN    BINARY_INTEGER NOT NULL,
     val   OUT   dty);
```

Where *dty* is one of

- DATE
- NUMBER
- VARCHAR2

For retrieving values of datatype RAW, see GET_VALUE_RAW.

### Parameters

***Table C–22 GET_VALUE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_ CURSOR and PARSE respectively. |
| pos | Position of the bind variable or select list item in the SQL statement. Starts from 1. |
| val | Variable in which the OUT bind variable or select list item will store its value. |

### Exceptions

*Table C–23 GET_RAW Procedure Exceptions*

| Exception | Description |
|---|---|
| ORA-1403 | Returns NO_DATA_FOUND exception when executing the GET_VALUE after the last row was fetched (i.e. FETCH_ROW returned 0). |
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor, parse and execute (or fetch) the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- FETCH_ROW function
- GET_VALUE_RAW procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_INOUT_VARIABLE procedure
- BIND_INOUT_VARIABLE_RAW procedure

## GET_VALUE_RAW procedure

This procedure, which operates on RAW datatypes, has two purposes:

- To retrieve the select list items of SELECT statements after a row has been fetched.

- To retrieve the OUT bind values after the SQL statement has been executed.

### Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
     c      IN    BINARY_INTEGER NOT NULL,
     pos    IN    BINARY_INTEGER NOT NULL,
     val    OUT   RAW);
```

### Parameters

***Table C–24 GET_VALUE_RAW Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_ CURSOR and PARSE respectively. |
| pos | Position of the bind variable or select list item in the SQL statement. Starts from 1. |
| val | Variable in which the OUT bind variable or select list item will store its value. |

### Exceptions

***Table C–25 GET_VALUE_RAW Procedure Exceptions***

| Exception | Description |
|---|---|
| ORA-1403 | Returns NO_DATA_FOUND exception when executing the GET_VALUE after the last row was fetched (i.e. FETCH_ROW returned 0). |
| ORA-28550 | The cursor passed is invalid. |
| ORA-28552 | Procedure is not executed in right order. Did you first open the cursor, parse and execute (or fetch) the SQL statement? |
| ORA-28553 | The position of the bind variable is out of range. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- FETCH_ROW function
- GET_VALUE procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_INOUT_VARIABLE procedure
- BIND_INOUT_VARIABLE_RAW procedure

## OPEN_CURSOR function

This function opens a cursor for executing a pass-through SQL statement at the non-Oracle system. This function must be called for any type of SQL statement The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, you call the procedure DBMS_HS_PASSTHROUGH.CLOSE_CURSOR.

### Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR ()
     RETURN   BINARY_INTEGER;
```

### Returns

The cursor to be used on subsequent procedure and function calls.

### Exceptions

***Table C–26 OPEN_CURSOR Function Exceptions***

| Exception | Description |
|---|---|
| ORA-28554 | Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' OPEN_CURSORS initialization parameter. |

### Pragmas

Purity level defined: WNDS, RNDS

**See Also:** CLOSE_CURSOR function

## PARSE procedure

This procedure parses SQL statement at non-Oracle system.

### Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
     c     IN    BINARY_INTEGER NOT NULL,
     stmt  IN    VARCHAR2       NOT NULL);
```

### Parameters

***Table C–27 PARSE Procedure Parameters***

| Parameter | Description |
|---|---|
| c | Cursor associated with the pass-through SQL statement. Cursor must be opened using function OPEN_CURSOR. |
| stmt | Statement to be parsed. |

### Exceptions

***Table C–28 PARSE Procedure Exceptions***

| Exception | Description |
|---|---|
| ORA-28550 | The cursor passed is invalid. |
| ORA-28551 | SQL statement is illegal. |
| ORA-28555 | A NULL value was passed for a NOT NULL parameter. |

### Pragmas

Purity level defined: WNDS, RNDS

**See Also:**

- OPEN_CURSOR function
- PARSE procedure
- FETCH_ROW function
- GET_VALUE procedure
- BIND_OUT_VARIABLE procedure
- BIND_OUT_VARIABLE_RAW procedure
- BIND_INOUT_VARIABLE procedure
- BIND_INOUT_VARIABLE_RAW procedure

# D

# Data Dictionary Translation for Generic Connectivity

Generic connectivity agents translate a query that refers to an Oracle8*i* data dictionary table into a query that retrieves the data from a non-Oracle data dictionary. You perform queries on data dictionary tables over the database link in the same way you query data dictionary tables in Oracle8*i*. The generic connectivity data dictionary is similar to the Oracle8*i* data dictionary in appearance and use. Non-Oracle data dictionary information is supplied to the user in Oracle8*i* data dictionary format.

To better understand the data dictionary support provided by generic connectivity, read these sections:

- Data Dictionary Translation Support
- Data Dictionary Mapping
- Generic Connectivity Data Dictionary Descriptions

# Data Dictionary Translation Support

Data dictionary information is stored in the non-Oracle system as system tables and accessed through ODBC or OLE DB application programming interfaces (APIs). This section contains the following topics:


- Accessing the Non-Oracle Data Dictionary
- Supported Views and Tables


## Accessing the Non-Oracle Data Dictionary

Accessing a non-Oracle data dictionary table or view is identical to accessing a data dictionary in an Oracle database. You issue a SELECT statement specifying a database link. The Oracle8*i* data dictionary view and column names are used to access the non-Oracle data dictionary. Synonyms of supported views are also acceptable.

For example, the following statement queries the data dictionary table ALL_USERS to retrieve all users in the non-Oracle system:

```
SQL> SELECT * FROM all_users@sid1;
```

When you issue a data dictionary access query, the ODBC or OLE DB agent:

1. Maps the requested table, view, or synonym to one or more ODBC or OLE DB APIs (see "Data Dictionary Mapping"). The agent translates all data dictionary column names to their corresponding non-Oracle column names within the query.
2. Sends the sequence of APIs to the non-Oracle system.
3. Possibly converts the retrieved non-Oracle data to give it the appearance of the Oracle8*i* data dictionary table.
4. Passes the data dictionary information from the non-Oracle system table to the Oracle8*i*.

---

**Note:** The values returned when querying the generic connectivity data dictionary may not be the same as the ones returned by the Oracle Enterprise Manager DESCRIBE command.

---

## Supported Views and Tables

Generic connectivity supports only these views and tables:

- ALL_CATALOG
- ALL_COL_COMMENTS
- ALL_CONS_COLUMNS
- ALL_CONSTRAINTS
- ALL_IND_COLUMNS
- ALL_INDEXES
- ALL_OBJECTS
- ALL_TAB_COLUMNS
- ALL_TAB_COMMENTS
- ALL_TABLES
- ALL_USERS
- ALL_VIEWS
- DICTIONARY
- USER_CATALOG
- USER_COL_COMMENTS
- USER_CONS_COLUMNS
- USER_CONSTRAINTS
- USER_IND_COLUMNS
- USER_INDEXES
- USER_OBJECTS
- USER_TAB_COLUMNS
- USER_TAB_COMMENTS
- USER_TABLES
- USER_USERS
- USER_VIEWS

If you use an unsupported view, then you receive the Oracle8*i* message for no rows selected.

If you want to query data dictionary views using SELECT ... FROM DBA_*, first connect as Oracle user SYSTEM or SYS. Otherwise, you receive the following error message:

```
ORA-28506: Parse error in data dictionary translation for %s stored in %s
```

Using generic connectivity, queries of the supported data dictionary tables and views beginning with the characters "ALL_" may return rows from the non-Oracle system when you do not have access privileges for those non-Oracle objects. When querying an Oracle database with the Oracle data dictionary, rows are returned only for those objects you are permitted to access.

## Data Dictionary Mapping

The tables in this section list Oracle data dictionary view names and the equivalent ODBC or OLE DB APIs used.

***Table 4–2 Generic Connectivity Data Dictionary Mapping***

| View | ODBC API | OLE DB API |
|---|---|---|
| ALL_CATALOG | SQLTables | DBSCHEMA_CATALOGS |
| ALL_COL_COMMENTS | SQLColumns | DBSCHEMA_COLUMNS |
| ALL_CONS_COLUMNS | SQLPrimaryKeys, SQLForeignKeys | DBSCHEMA_PRIMARY_KEYS, DBSCHEMA_FOREIGN_KEYS |
| ALL_CONSTRAINTS | SQLPrimaryKeys, SQLForeignKeys | DBSCHEMA_PRIMARY_KEYS, DBSCHEMA_FOREIGN_KEYS |
| ALL_IND_COLUMNS | SQLStatistics | DBSCHEMA_STATISTICS |
| ALL_INDEXES | SQLStatistics | DBSCHEMA_STATISTICS |
| ALL_OBJECTS | SQLTables, SQLProcedures, SQLStatistics | DBSCHEMA_TABLES, DBSCHEMA_PROCEDURES, DBSCHEMA_STATISTICS |
| ALL_TAB_COLUMNS | SQLColumns | DBSCHEMA_COLUMNS |
| ALL_TAB_COMMENTS | SQLTables | DBSCHEMA_TABLES |
| ALL_TABLES | SQLStatistics | DBSCHEMA_STATISTICS |
| ALL_USERS | SQLTables | DBSCHEMA_TABLES |

*Table 4–2 Generic Connectivity Data Dictionary Mapping*

| View | ODBC API | OLE DB API |
|---|---|---|
| ALL_VIEWS | SQLTables | DBSCHEMA_TABLES |
| DICTIONARY | SQLTables | DBSCHEMA_TABLES |
| USER_CATALOG | SQLTables | DBSCHEMA_TABLES |
| USER_COL_COMMENTS | SQLColumns | DBSCHEMA_COLUMNS |
| USER_CONS_COLUMNS | SQLPrimaryKeys, SQLForeignKeys | DBSCHEMA_PRIMARY_KEYS, DBSCHEMA_FOREIGN_KEYS |
| USER_CONSTRAINTS | SQLPrimaryKeys, SQLForeignKeys | DBSCHEMA_PRIMARY_KEYS, DBSCHEMA_FOREIGN_KEYS |
| USER_IND_COLUMNS | SQLStatistics | DBSCHEMA_STATISTICS |
| USER_INDEXES | SQLStatistics | DBSCHEMA_STATISTICS |
| USER_OBJECTS | SQLTables, SQLProcedures, SQLStatistics | DBSCHEMA_TABLES, DBSCHEMA_PROCEDURES, DBSCHEMA_STATISTICS |
| USER_TAB_COLUMNS | SQLColumns | DBSCHEMA_COLUMNS |
| USER_TAB_COMMENTS | SQLTables | DBSCHEMA_TABLES |
| USER_TABLES | SQLStatistics | DBSCHEMA_STATISTICS |
| USER_USERS | SQLTables | DBSCHEMA_TABLES |
| USER_VIEWS | SQLTables | DBSCHEMA_TABLES |

## Default Column Values

The generic connectivity data dictionary differs from a typical Oracle database server data dictionary. The Oracle database server columns that are missing in a non-Oracle data dictionary table are filled with the following, depending on the column type:

- Zeros
- Spaces
- NULL values
- Default values

# Generic Connectivity Data Dictionary Descriptions

The generic connectivity data dictionary tables and views provide this information:

- Name, data type, and width of each column
- The contents of columns with fixed values

In the descriptions that follow, the values in the Null? column may differ from the Oracle8*i* data dictionary tables and views. Any default value is shown to the right of an item.

## ALL_CATALOG

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | VARCHAR2(11) | "TABLE" or "VIEW" or "SYNONYM" |

## ALL_COL_COMMENTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | NOT NULL | VARCHAR2(30) | |
| COMMENTS | | VARCHAR2(4000) | NULL |

## ALL_CONS_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |

| Name | Null? | Type | Value |
|---|---|---|---|
| COLUMN_NAME | | VARCHAR2(4000) | |
| POSITION | | NUMBER | |

## ALL_CONSTRAINTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_NAME | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_TYPE | | VARCHAR2(1) | "R" or "P" |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| SEARCH_CONDITION | | LONG | NULL |
| R_OWNER | | VARCHAR2(30) | |
| R_CONSTRAINT_NAME | | VARCHAR2(30) | |
| DELETE_RULE | | VARCHAR2(9) | "CASCADE" or "NO ACTION" or "SET NULL" |
| STATUS | | VARCHAR2(8) | NULL |
| DEFERRABLE | | VARCHAR2(14) | NULL |
| DEFERRED | | VARCHAR2(9) | NULL |
| VALIDATED | | VARCHAR2(13) | NULL |
| GENERATED | | VARCHAR2(14) | NULL |
| BAD | | VARCHAR2(3) | NULL |
| RELY | | VARCHAR2(4) | NULL |
| LAST_CHANGE | | DATE | NULL |

## ALL_IND_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| INDEX_OWNER | NOT NULL | VARCHAR2(30) | |

| Name | Null? | Type | Value |
|---|---|---|---|
| INDEX_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | | VARCHAR2(4000) | |
| COLUMN_POSITION | NOT NULL | NUMBER | |
| COLUMN_LENGTH | NOT NULL | NUMBER | |
| DESCEND | | VARCHAR2(4) | "DESC" or "ASC" |

# ALL_INDEXES

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| INDEX_NAME | NOT NULL | VARCHAR2(30) | |
| INDEX_TYPE | | VARCHAR2(27) | NULL |
| TABLE_OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | CHAR(5) | "TABLE" |
| UNIQUENESS | | VARCHAR2(9) | "UNIQUE" or "NONUNIQUE" |
| COMPRESSION | | VARCHAR2(8) | NULL |
| PREFIX_LENGTH | | NUMBER | 0 |
| TABLESPACE_NAME | | VARCHAR2(30) | NULL |
| INI_TRANS | | NUMBER | 0 |
| MAX_TRANS | | NUMBER | 0 |
| INITIAL_EXTENT | | NUMBER | 0 |
| NEXT_EXTENT | | NUMBER | 0 |
| MIN_EXTENTS | | NUMBER | 0 |
| MAX_EXTENTS | | NUMBER | 0 |

| Name | Null? | Type | Value |
|---|---|---|---|
| PCT_INCREASE | | NUMBER | 0 |
| PCT_THRESHOLD | | NUMBER | 0 |
| INCLUDE_COLUMNS | | NUMBER | 0 |
| FREELISTS | | NUMBER | 0 |
| FREELIST_GROUPS | | NUMBER | 0 |
| PCT_FREE | | NUMBER | 0 |
| LOGGING | | VARCHAR2(3) | NULL |
| BLEVEL | | NUMBER | 0 |
| LEAF_BLOCKS | | NUMBER | 0 |
| DISTINCT_KEYS | | NUMBER | |
| AVG_LEAF_BLOCKS_PER_KEY | | NUMBER | 0 |
| AVG_DATA_BLOCKS_PER_KEY | | NUMBER | 0 |
| CLUSTERING_FACTOR | | NUMBER | 0 |
| STATUS | | VARCHAR2(8) | NULL |
| NUM_ROWS | | NUMBER | 0 |
| SAMPLE_SIZE | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| DEGREE | | VARCHAR2(40) | NULL |
| INSTANCES | | VARCHAR2(40) | NULL |
| PARTITIONED | | VARCHAR2(3) | NULL |
| TEMPORARY | | VARCHAR2(1) | NULL |
| GENERATED | | VARCHAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |
| BUFFER_POOL | | VARCHAR2(7) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| DURATION | | VARCHAR2(15) | NULL |
| PCT_DIRECT_ACCESS | | NUMBER | 0 |
| ITYP_OWNER | | VARCHAR2(30) | NULL |

| Name | Null? | Type | Value |
|---|---|---|---|
| ITYP_NAME | | VARCHAR2(30) | NULL |
| PARAMETERS | | VARCHAR2(1000) | NULL |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| DOMIDX_STATUS | | VARCHAR2(12) | NULL |
| DOMIDX_OPSTATUS | | VARCHAR2(6) | NULL |
| FUNCIDX_STATUS | | VARCHAR2(8) | NULL |

# ALL_OBJECTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| OBJECT_NAME | NOT NULL | VARCHAR2(30) | |
| SUBOBJECT_NAME | | VARCHAR2(30) | NULL |
| OBJECT_ID | NOT NULL | NUMBER | 0 |
| DATA_OBJECT_ID | | NUMBER | 0 |
| OBJECT_TYPE | | VARCHAR2(18) | "TABLE" or "VIEW" or "SYNONYM" or "INDEX" or "PROCEDURE" |
| CREATED | NOT NULL | DATE | NULL |
| LAST_DDL_TIME | NOT NULL | DATE | NULL |
| TIMESTAMP | | VARCHAR2(19) | NULL |
| STATUS | | VARCHAR2(7) | NULL |
| TEMPORARY | | VARCHAR2(1) | NULL |
| GENERATED | | VARCHAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |

# ALL_TAB_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | NOT NULL | VARCHAR2(30) | |
| DATA_TYPE | | VARCHAR2(106) | |
| DATA_TYPE_MOD | | VARCHAR2(3) | NULL |
| DATA_TYPE_OWNER | | VARCHAR2(30) | NULL |
| DATA_LENGTH | NOT NULL | NUMBER | |
| DATA_PRECISION | | NUMBER | |
| DATA_SCALE | | NUMBER | |
| NULLABLE | | VARCHAR2(1) | "Y" or "N" |
| COLUMN_ID | NOT NULL | NUMBER | |
| DEFAULT_LENGTH | | NUMBER | 0 |
| DATA_DEFAULT | | LONG | NULL |
| NUM_DISTINCT | | NUMBER | 0 |
| LOW_VALUE | | RAW(32) | NULL |
| HIGH_VALUE | | RAW(32) | NULL |
| DENSITY | | NUMBER | 0 |
| NUM_NULLS | | NUMBER | 0 |
| NUM_BUCKETS | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| SAMPLE_SIZE | | NUMBER | 0 |
| CHARACTER_SET_NAME | | VARCHAR2(44) | NULL |
| CHAR_COL_DEC_LENGTH | | NUMBER | 0 |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| AVG_COL_LEN | | NUMBER | 0 |

# ALL_TAB_COMMENTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | VARCHAR2(11) | "TABLE" or "VIEW" |
| COMMENTS | | VARCHAR2(4000) | NULL |

# ALL_TABLES

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLESPACE_NAME | | VARCHAR2(30) | NULL |
| CLUSTER_NAME | | VARCHAR2(30) | NULL |
| IOT_NAME | | VARCHAR2(30) | NULL |
| PCT_FREE | | NUMBER | 0 |
| PCT_USED | | NUMBER | 0 |
| INI_TRANS | | NUMBER | 0 |
| MAX_TRANS | | NUMBER | 0 |
| INITIAL_EXTENT | | NUMBER | 0 |
| NEXT_EXTENT | | NUMBER | 0 |
| MIN_EXTENTS | | NUMBER | 0 |
| MAX_EXTENTS | | NUMBER | 0 |
| PCT_INCREASE | | NUMBER | 0 |
| FREELISTS | | NUMBER | 0 |
| FREELIST_GROUPS | | NUMBER | 0 |
| LOGGING | | VARCHAR2(3) | NULL |
| BACKED_UP | | VARCHAR2(1) | NULL |

| Name | Null? | Type | Value |
|---|---|---|---|
| NUM_ROWS | | NUMBER | |
| BLOCKS | | NUMBER | |
| EMPTY_BLOCKS | | NUMBER | 0 |
| AVG_SPACE | | NUMBER | 0 |
| CHAIN_CNT | | NUMBER | 0 |
| AVG_ROW_LEN | | NUMBER | 0 |
| AVG_SPACE_FREELIST_BLOCKS | | NUMBER | 0 |
| NUM_FREELIST_BLOCKS | | NUMBER | 0 |
| DEGREE | | VARCHAR2(10) | NULL |
| INSTANCES | | VARCHAR2(10) | NULL |
| CACHE | | VARCHAR2(5) | NULL |
| TABLE_LOCK | | VARCHAR2(8) | NULL |
| SAMPLE_SIZE | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| PARTITIONED | | VARCHAR2(3) | NULL |
| IOT_TYPE | | VARCHAR2(12) | NULL |
| TEMPORARY | | VARHCAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |
| NESTED | | VARCHAR2(3) | NULL |
| BUFFER_POOL | | VARCHAR2(7) | NULL |
| ROW_MOVEMENT | | VARCHAR2(8) | NULL |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| DURATION | | VARHCAR2(15) | NULL |
| SKIP_CORRUPT | | VARCHAR2(8) | NULL |
| MONITORING | | VARCHAR2(3) | NULL |

## ALL_USERS

| Name | Null? | Type | Value |
|---|---|---|---|
| USERNAME | NOT NULL | VARCHAR2(30) | |
| USER_ID | NOT NULL | NUMBER | 0 |
| CREATED | NOT NULL | DATE | NULL |

## ALL_VIEWS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| VIEW_NAME | NOT NULL | VARCHAR2(30) | |
| TEXT_LENGTH | | NUMBER | 0 |
| TEXT | NOT NULL | LONG | NULL |
| TYPE_TEXT_LENGTH | | NUMBER | 0 |
| TYPE_TEXT | | VARCHAR2(4000) | NULL |
| OID_TEXT_LENGTH | | NUMBER | 0 |
| OID_TEXT | | VARCHAR2(4000) | NULL |
| VIEW_TYPE_OWNER | | VARCHAR2(30) | NULL |
| VIEW_TYPE | | VARCHAR2(30) | NULL |

## DICTIONARY

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | | VARCHAR2(30) | |
| COMMENTS | | VARCHAR2(4000) | NULL |

## USER_CATALOG

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | VARCHAR2(11) | "TABLE" or "VIEW" or "SYNONYM" |

## USER_COL_COMMENTS

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | NOT NULL | VARCHAR2(30) | |
| COMMENTS | | VARCHAR2(4000) | NULL |

## USER_CONS_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | | VARCHAR2(4000) | |
| POSITION | | NUMBER | |

## USER_CONSTRAINTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OWNER | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_NAME | NOT NULL | VARCHAR2(30) | |
| CONSTRAINT_TYPE | | VARCHAR2(1) | "R" or "P" |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |

| Name | Null? | Type | Value |
|---|---|---|---|
| SEARCH_CONDITION | | LONG | NULL |
| R_OWNER | | VARCHAR2(30) | |
| R_CONSTRAINT_NAME | | VARCHAR2(30) | |
| DELETE_RULE | | VARCHAR2(9) | "CASCADE" or "NOACTION" or "SET NULL" |
| STATUS | | VARCHAR2(8) | NULL |
| DEFERRABLE | | VARCHAR2(14) | NULL |
| DEFERRED | | VARCHAR2(9) | NULL |
| VALIDATED | | VARCHAR2(13) | NULL |
| GENERATED | | VARCHAR2(14) | NULL |
| BAD | | VARCHAR2(3) | NULL |
| RELY | | VARCHAR2(4) | NULL |
| LAST_CHANGE | | DATE | NULL |

## USER_IND_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| INDEX_NAME | | VARCHAR2(30) | |
| TABLE_NAME | | VARCHAR2(30) | |
| COLUMN_NAME | | VARCHAR2(4000) | |
| COLUMN_POSITION | | NUMBER | |
| COLUMN_LENGTH | | NUMBER | |
| DESCEND | | VARCHAR2(4) | "DESC" or "ASC" |

# USER_INDEXES

| Name | Null? | Type | Value |
|---|---|---|---|
| INDEX_NAME | NOT NULL | VARCHAR2(30) | |
| INDEX_TYPE | | VARCHAR2(27) | NULL |
| TABLE_OWNER | NOT NULL | VARCHAR2(30) | |
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | VARCHAR2(11) | "TABLE" |
| UNIQUENESS | | VARCHAR2(9) | "UNIQUE" or "NONUNIQUE" |
| COMPRESSION | | VARCHAR2(8) | NULL |
| PREFIX_LENGTH | | NUMBER | 0 |
| TABLESPACE_NAME | | VARCHAR2(30) | NULL |
| INI_TRANS | | NUMBER | 0 |
| MAX_TRANS | | NUMBER | 0 |
| INITIAL_EXTENT | | NUMBER | 0 |
| NEXT_EXTENT | | NUMBER | 0 |
| MIN_EXTENTS | | NUMBER | 0 |
| MAX_EXTENTS | | NUMBER | 0 |
| PCT_INCREASE | | NUMBER | 0 |
| PCT_THRESHOLD | | NUMBER | 0 |
| INCLUDE_COLUMNS | | NUMBER | 0 |
| FREELISTS | | NUMBER | 0 |
| FREELIST_GROUPS | | NUMBER | 0 |
| PCT_FREE | | NUMBER | 0 |
| LOGGING | | VARCHAR2(3) | NULL |
| BLEVEL | | NUMBER | 0 |
| LEAF_BLOCKS | | NUMBER | 0 |
| DISTINCT_KEYS | | NUMBER | |

| Name | Null? | Type | Value |
|---|---|---|---|
| AVG_LEAF_BLOCKS_PER_KEY | | NUMBER | 0 |
| AVG_DATA_BLOCKS_PER_KEY | | NUMBER | 0 |
| CLUSTERING_FACTOR | | NUMBER | 0 |
| STATUS | | VARCHAR2(8) | NULL |
| NUM_ROWS | | NUMBER | 0 |
| SAMPLE_SIZE | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| DEGREE | | VARCHAR2(40) | NULL |
| INSTANCES | | VARCHAR2(40) | NULL |
| PARTITIONED | | VARCHAR2(3) | NULL |
| TEMPORARY | | VARCHAR2(1) | NULL |
| GENERATED | | VARCHAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |
| BUFFER_POOL | | VARCHAR2(7) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| DURATION | | VARHCAR2(15) | NULL |
| PCT_DIRECT_ACCESS | | NUMBER | 0 |
| ITYP_OWNER | | VARCHAR2(30) | NULL |
| ITYP_NAME | | VARCHAR2(30) | NULL |
| PARAMETERS | | VARCHAR2(1000) | NULL |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| DOMIDX_STATUS | | VARCHAR2(12) | NULL |
| DOMIDX_OPSTATUS | | VARCHAR2(6) | NULL |
| FUNCIDX_STATUS | | VARCHAR2(8) | NULL |

## USER_OBJECTS

| Name | Null? | Type | Value |
|---|---|---|---|
| OBJECT_NAME | | VARCHAR2(128) | |
| SUBOBJECT_NAME | | VARCHAR2(30) | NULL |
| OBJECT_ID | | NUMBER | 0 |
| DATA_OBJECT_ID | | NUMBER | 0 |
| OBJECT_TYPE | | VARCHAR2(18) | "TABLE" or "VIEW" or "SYNONYM" or "INDEX" or "PROCEDURE" |
| CREATED | | DATE | NULL |
| LAST_DDL_TIME | | DATE | NULL |
| TIMESTAMP | | VARCHAR2(19) | NULL |
| STATUS | | VARCHAR2(7) | NULL |
| TEMPORARY | | VARCHAR2(1) | NULL |
| GENERATED | | VARCHAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |

## USER_TAB_COLUMNS

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| COLUMN_NAME | NOT NULL | VARCHAR2(30) | |
| DATA_TYPE | | VARCHAR2(106) | |
| DATA_TYPE_MOD | | VARCHAR2(3) | NULL |
| DATA_TYPE_OWNER | | VARCHAR2(30) | NULL |
| DATA_LENGTH | NOT NULL | NUMBER | |
| DATA_PRECISION | | NUMBER | |
| DATA_SCALE | | NUMBER | |

| Name | Null? | Type | Value |
|---|---|---|---|
| NULLABLE | | VARCHAR2(1) | "Y" or "N" |
| COLUMN_ID | NOT NULL | NUMBER | |
| DEFAULT_LENGTH | | NUMBER | NULL |
| DATA_DEFAULT | | LONG | NULL |
| NUM_DISTINCT | | NUMBER | NULL |
| LOW_VALUE | | RAW(32) | NULL |
| HIGH_VALUE | | RAW(32) | NULL |
| DENSITY | | NUMBER | 0 |
| NUM_NULLS | | NUMBER | 0 |
| NUM_BUCKETS | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| SAMPLE_SIZE | | NUMBER | 0 |
| CHARACTER_SET_NAME | | VARCHAR2(44) | NULL |
| CHAR_COL_DECL_LENGTH | | NUMBER | 0 |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| AVG_COL_LEN | | NUMBER | 0 |

# USER_TAB_COMMENTS

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLE_TYPE | | VARCHAR2(11) | "TABLE" or "VIEW" |
| COMMENTS | | VARCHAR2(4000) | NULL |

# USER_TABLES

| Name | Null? | Type | Value |
|---|---|---|---|
| TABLE_NAME | NOT NULL | VARCHAR2(30) | |
| TABLESPACE_NAME | | VARCHAR2(30) | NULL |
| CLUSTER_NAME | | VARCHAR2(30) | NULL |
| IOT_NAME | | VARCHAR2(30) | NULL |
| PCT_FREE | | NUMBER | 0 |
| PCT_USED | | NUMBER | 0 |
| INI_TRANS | | NUMBER | 0 |
| MAX_TRANS | | NUMBER | 0 |
| INITIAL_EXTENT | | NUMBER | 0 |
| NEXT_EXTENT | | NUMBER | 0 |
| MIN_EXTENTS | | NUMBER | 0 |
| MAX_EXTENTS | | NUMBER | 0 |
| PCT_INCREASE | | NUMBER | 0 |
| FREELISTS | | NUMBER | 0 |
| FREELIST_GROUPS | | NUMBER | 0 |
| LOGGING | | VARCHAR2(3) | NULL |
| BACKED_UP | | VARCHAR2(1) | NULL |
| NUM_ROWS | | NUMBER | |
| BLOCKS | | NUMBER | |
| EMPTY_BLOCKS | | NUMBER | 0 |
| AVG_SPACE | | NUMBER | 0 |
| CHAIN_CNT | | NUMBER | 0 |
| AVG_ROW_LEN | | NUMBER | 0 |
| AVG_SPACE_FREELIST_BLOCKS | | NUMBER | 0 |
| NUM_FREELIST_BLOCKS | | NUMBER | 0 |
| DEGREE | | VARCHAR2(10) | NULL |

| Name | Null? | Type | Value |
|---|---|---|---|
| INSTANCES | | VARCHAR2(10) | NULL |
| CACHE | | VARCHAR2(5) | NULL |
| TABLE_LOCK | | VARCHAR2(8) | NULL |
| SAMPLE_SIZE | | NUMBER | 0 |
| LAST_ANALYZED | | DATE | NULL |
| PARTITIONED | | VARCHAR2(3) | NULL |
| IOT_TYPE | | VARCHAR2(12) | NULL |
| TEMPORARY | | VARHCAR2(1) | NULL |
| SECONDARY | | VARCHAR2(1) | NULL |
| NESTED | | VARCHAR2(3) | NULL |
| BUFFER_POOL | | VARCHAR2(7) | NULL |
| ROW_MOVEMENT | | VARCHAR2(8) | NULL |
| GLOBAL_STATS | | VARCHAR2(3) | NULL |
| USER_STATS | | VARCHAR2(3) | NULL |
| DURATION | | VARCHAR2(15) | NULL |
| SKIP_CORRUPT | | VARCHAR2(8) | NULL |
| MONITORING | | VARCHAR2(3) | NULL |

# USER_USERS

| Name | Null? | Type | Value |
|---|---|---|---|
| USERNAME | NOT NULL | VARCHAR2(30) | |
| USER_ID | NOT NULL | NUMBER | 0 |
| ACCOUNT_STATUS | NOT NULL | VARCHAR2(32) | "OPEN" |
| LOCK_DATE | | DATE | NULL |
| EXPIRY_DATE | | DATE | NULL |
| DEFAULT_TABLESPACE | NOT NULL | VARCHAR2(30) | NULL |
| TEMPORARY_TABLESPACE | NOT NULL | VARCHAR2(30) | NULL |

| Name | Null? | Type | Value |
|---|---|---|---|
| CREATED | NOT NULL | DATE | NULL |
| INITIAL_RSRC_CONSUMER_GROUP | | VARCHAR2(30) | NULL |
| EXTERNAL_NAME | | VARCHAR2(4000) | NULL |

## USER_VIEWS

| Name | Null? | Type | Value |
|---|---|---|---|
| VIEW_NAME | NOT NULL | VARCHAR2(30) | |
| TEXT_LENGTH | | NUMBER | 0 |
| TEXT | | LONG | NULL |
| TYPE_TEXT_LENGTH | | NUMBER | 0 |
| TYPE_TEXT | | VARCHAR2(4000) | NULL |
| OID_TEXT_LENGTH | | NUMBER | 0 |
| OID_TEXT | | VARCHAR2(4000) | NULL |
| VIEW_TYPE_OWNER | | VARCHAR2(30) | NULL |
| VIEW_TYPE | | VARCHAR2(30) | NULL |

# E

# Datatype Mapping

Oracle8*i* maps the datatypes used in ODBC and OLE DB compliant data sources to supported Oracle datatypes. When the results of a query are returned, Oracle8*i* converts the ODBC or OLE DB datatypes to Oracle datatypes. For information on how the datatypes are mapped for each data source, see the following:

- Mapping ODBC Datatypes to Oracle Datatypes
- Mapping OLE DB Datatypes to Oracle Datatypes

# Mapping ODBC Datatypes to Oracle Datatypes

This table shows the mapping from ODBC datatypes to Oracle datatypes:

| ODBC | Oracle |
|---|---|
| SQL_BIGINT | NUMBER(19,0) |
| SQL_BINARY | RAW |
| SQL_CHAR | CHAR |
| SQL_DATE | DATE |
| SQL_DECIMAL(p,s) | NUMBER(p,s) |
| SQL_DOUBLE | FLOAT(49) |
| SQL_FLOAT | FLOAT(49) |
| SQL_INTEGER | NUMBER(10) |
| SQL_LONGVARBINARY | LONG RAW |
| SQL_LONGVARCHAR | LONG |
| SQL_NUMERIC(p,s) | NUMBER(p,s) |
| SQL_REAL | FLOAT(23) |
| SQL_SMALLINT | NUMBER(5) |
| SQL_TIME | DATE |
| SQL_TIMESTAMP | DATE |
| SQL_TINYINT | NUMBER(3) |
| SQL_VARCHAR | VARCHAR |

# Mapping OLE DB Datatypes to Oracle Datatypes

This table shows the mapping from OLE DB datatypes to Oracle datatypes:

| OLE DB | Oracle |
|---|---|
| DBTYPE_UI1 | NUMBER(3) |
| DBTYPE_I1 | NUMBER(3) |
| DBTYPE_UI2 | NUMBER(5) |
| DBTYPE_I2 | NUMBER(5) |
| DBTYPE_BOOL | NUMBER(5) |
| DBTYPE_UI4 | NUMBER(10) |
| DBTYPE_I4 | NUMBER(10) |
| DBTYPE_UI8 | NUMBER(19,0) |
| DBTYPE_I8 | NUMBER(19,0) |
| DBTYPE_NUMERIC(p,s) | NUMBER(p,s) |
| DBTYPE_R4 | FLOAT(23) |
| DBTYPE_R8 | FLOAT(49) |
| DBTYPE_DECIMAL | FLOAT(49) |
| DBTYPE_STR | VARCHAR2 |
| DBTYPE_WSTR | VARCHAR2 |
| DBTYPE_CY | NUMBER(19,0) |
| DBTYPE_DBDATE | DATE |
| DBTYPE_DBTIME | DATE |
| DBTYPE_DBTIMESTAMP | DATE |
| DBTYPE_BYTES | RAW |
| DTYPE_BYTES (long attribute) | LONG RAW |
| DBTYPE_STRING (long attribute) | LONG |

# Index


## A

agents
  generic connectivity, 1-4
  Heterogeneous Services, 1-3
    disabling self-registration, 2-6
    registering, 2-5, 2-6, 2-7
  specifying initialization parameters for, 2-4
application development
  Heterogeneous Services, 4-1, 4-2
    controlling array fetches between non-Oracle server and agent, 4-11
    controlling array fetches between Oracle server and agent, 4-11
    controlling reblocking of array fetches, 4-11
    DBMS_HS_PASSTHROUGH package, 4-2
    pass-through SQL, 4-2
    using bulk fetches, 4-9
    using OCI for bulk fetches, 4-10
  using Heterogeneous Services, 4-1
Architecture of the Heterogenous Services Data Dictionary, 1-6
array fetches, 4-10
  agents, 4-11

## B

bind queries
  executing using pass-through SQL, 4-7
BIND_INOUT_VARIABLE procedure, 4-3, 4-7
BIND_OUT_VARIABLE procedure, 4-3, 4-6
BIND_VARIABLE procedure, 4-3
buffers
  multiple rows, 4-8
bulk fetches
  optimizing data transfers using, 4-9

## C

CATHO.SQL script
  installing data dictionary for Heterogeneous Services, 2-2
character sets
  Heterogeneous Services, A-7
CLOSE_CURSOR function, 4-3
commit point site
  commit point strength, A-3
configuring generic connectivity, 3-8
configuring transparent gateways, 2-2
CREATE_INST_INIT procedure, 2-19

## D

data dictionary
  contents with generic connectivity, D-3
  installing for Heterogeneous Services, 2-2
  mapping for generic connectivity, D-4
  Oracle server name / SQL Server name, D-4
  tables, 1-5
  translation support for generic connectivity, D-2
data dictionary views
  generic connectivity, D-3
  Heterogeneous Services, 2-9, B-1
database links
  heterogeneous systems, 1-5, 2-4
datatypes
  mapping, 1-5

ODBC, E-2
ODBC to Oracle, E-2
OLE DB, E-3
OLE DB to Oracle, E-3
date formats
Heterogeneous Services, A-8
DBMS_HS package
specifying HS parameters, 2-18
DBMS_HS_PASSTHROUGH package, 4-2
list of functions and procedures, 4-3
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE, C-15
describe cache high water mark
definition, A-4
drivers
ODBC, 3-13
OLEFS, 3-16
OLESQL, 3-15
dynamic performance views
Heterogeneous Services, 2-16
determining open sessions, 2-17
determining which agents are on host, 2-16

## E

EXECUTE_IMMEDIATE procedure, 4-3
restrictions, 4-4
EXECUTE_NON_QUERY procedure, 4-3

## F

FDS_CLASS, 2-8
FDS_CLASS_VERSION, 2-8
FDS_INST_NAME, 2-8
FETCH_ROW procedure, 4-3
executing queries using pass-through SQL, 4-7
fetches
bulk, 4-9
optimizing round-trips, 4-8

## G

generic connectivity
architecture, 3-3
Oracle and non-Oracle on same machine, 3-4
Oracle and non-Oracle on separate machines, 3-3
configuration, 3-8
creating initialization file, 3-8
data dictionary
translation support, D-2
definition, 3-2
DELETE statement, 3-7
editing initialization file, 3-9
error tracing, A-6
Heterogeneous Services, 1-4
INSERT statement, 3-7
non-Oracle data dictionary access, D-2
ODBC connectivity requirements, 3-13
OLE DB (FS) connectivity requirements, 3-16
OLE DB (SQL) connectivity requirements, 3-15
restrictions, 3-6
setting parameters for ODBC source, 3-10
UNIX, 3-11
Windows NT, 3-10
setting parameters for OLE DB source, 3-12
SQL execution, 3-6
supported functions, 3-7
supported SQL syntax, 3-7
types of agents, 3-2
UPDATE statement, 3-7
GET_VALUE procedure, 4-3, 4-6, 4-7

## H

heterogeneous distributed systems
accessing, 2-2
Heterogeneous Services
agent registration, 2-5
avoiding configuration mismatches, 2-6
disabling, 2-6
enabling, 2-5
agents
self-registration, 2-7
application development, 4-1, 4-2
controlling array fetches between non-Oracle server and agent, 4-11
controlling array fetches between Oracle server and agent, 4-11
controlling reblocking of array fetches, 4-11

DBMS_HS_PASSTHOUGH package, 4-2
locking behavior of non-Oracle systems, 4-12
pass-through SQL, 4-2
using bulk fetches, 4-9
using OCI for bulk fetches, 4-10
creating database links, 2-4
data dictionary, 1-6
classes and instances, 1-7
data dictionary views, 2-9, B-1
types, 2-10
understanding sources, 2-12
using general views, 2-12
using SQL service views, 2-14
using transaction service views, 2-13
database links to non-Oracle systems, 1-5
DBMS_HS package
using to specify initialization parameters, 2-19
using to unspecify initialization parameters, 2-20
defining maximum number of open cursors, A-9
dynamic performance views, 2-16
V$HS_AGENT view, 2-16
V$HS_SESSION view, 2-17
generic connectivity
architecture, 3-3
creating initialization file, 3-8
definition, 3-2
editing initialization file, 3-9
non-Oracle data dictionary access, D-2
ODBC connectivity requirements, 3-13
OLE DB (FS) connectivity requirements, 3-16
OLE DB (SQL) connectivity requirements, 3-15
restrictions, 3-6
setting parameters for ODBC source, 3-10
setting parameters for OLE DB source, 3-12
SQL execution, 3-6
supported functions, 3-7
supported SQL syntax, 3-7
supported tables, D-3
types of agents, 3-2
initialization parameters
specifying, 2-19
unspecifying, 2-20
installing data dictionary, 2-2
locking behavior of non-Oracle systems, 4-12
optimizing data transfer, A-10
overview, 1-2
setting global name, A-4
setting up access using transparent gateway, 2-2
setting up environment, 2-2
specifying cache high water mark, A-4
specifying cache size, A-10
specifying commit point strength, A-3
specifying domain, A-3
specifying instance identifier, A-4
SQL service, 1-5
testing connections, 2-4
transaction service, 1-4
tuning internal data buffering, A-11
types, 1-4
HS_AUTOREGISTER initialization parameter, 2-18
using to enable agent self-registration, 2-9
HS_BASE_CAPS view, 2-11
HS_BASE_DD view, 2-11
HS_CLASS_CAPS view, 2-11
HS_CLASS_DD view, 2-11
HS_CLASS_INIT view, 2-11
HS_COMMIT_POINT_STRENGTH initialization parameter, A-3
HS_DB_DOMAIN initialization parameter, 2-20, A-3
HS_DB_INTERNAL_NAME initialization parameter, A-4
HS_DB_NAME initialization parameter, A-4
HS_DESCRIBE_CACHE_HWM initialization parameter, A-4
HS_FDS_CLASS view, 2-11
HS_FDS_CONNECT_INFO initialization parameter, A-5
specifying connection information, 3-9
HS_FDS_FETCH_ROWS initialization parameter, 4-11
HS_FDS_INST view, 2-11
HS_FDS_SHAREABLE_NAME initialization parameter, A-6

HS_FDS_TRACE initialization parameter, A-6
HS_FDS_TRACE_FILE_NAME initialization parameter, A-6
HS_FDS_TRACE_LEVEL initialization parameter
enabling agent tracing, 3-10
HS_INST_CAPS view, 2-11
HS_INST_DD view, 2-11
HS_INST_INIT view, 2-11
HS_LANGUAGE initialization parameter, A-7
HS_NLS_DATE_FORMAT initialization parameter, A-8
HS_NLS_DATE_LANGUAGE initialization parameter, A-8
HS_NLS_NCHAR initialization parameter, A-9
HS_OPEN_CURSORS initialization parameter, A-9
HS_ROWID_CACHE_SIZE initialization parameter, A-10
HS_RPC_FETCH_REBLOCKING initialization parameter, 4-11, A-10
HS_RPC_FETCH_SIZE initialization parameter, 4-11, A-11

## L

Limitations to Heterogeneous Services, 4-12
listeners, 2-2
locks
in non-Oracle systems, 4-12

## M

multiple rows
buffering, 4-8

## N

National Language Support (NLS)
Heterogeneous Services, A-7
character set of non-Oracle source, A-9
date format, A-8
languages in character date values, A-8
Net8 listener, 1-3, 2-2

## O

OCI
optimizing data transfers using, 4-10
ODBC agents
connectivity requirements, 3-13
functions, 3-14
ODBC connectivity
data dictionary mapping, D-4
mapping OBDC datatypes, E-2
mapping Oracle datatypes, E-2
ODBC driver, 3-13
requirements, 3-13
specifying connection information
UNIX, A-5
Windows NT, A-5
specifying path to library, A-6
OLE DB agents
connectivity requirements, 3-15, 3-16
OLE DB connectivity
data dictionary mapping, D-4
mapping to Oracle datatypes, E-3
setting connection information, A-5
OLEFS drivers, 3-16
data provider requirements, 3-16
initialization properties, 3-18
rowset properties, 3-18
OLESQL drivers, 3-15
security, 3-15
OPEN_CURSOR procedure, 4-3
operating system dependencies, C-1
Oracle precompiler
optimizing data transfers using, 4-10
OUT bind variables, 4-6

## P

PARSE procedure, 4-3
pass-through SQL, 4-2
avoiding SQL interpretation, 4-2
executing statements, 4-3
non-queries, 4-4
queries, 4-7
with bind variables, 4-4
with IN bind variables, 4-5
with IN OUT bind variables, 4-6

with OUT bind variables, 4-6
implications of using, 4-3
overview, 4-2
restrictions, 4-3
PL/SQL
development environment, 4-2

## Q

queries
pass-through SQL, 4-7

## R

reblocking, 4-11
Researching, 4-12
rows
buffering multiple, 4-8

## S

security
OLESQL driver, 3-15
SELECT statement
accessing non-Oracle system, D-2
service names
specifying in database links, 2-4
SQL capabilities
data dictionary tables, 2-15
SQL dialect
understood by non-Oracle system, 1-5
SQL service
capabilities, 1-5
data dictionary views, 1-8, 2-10
Heterogeneous Services, 1-5
views
Heterogeneous Services, 2-14
SQL statements
mapping to non-Oracle datastores, 4-2

## T

transaction service
Heterogeneous Services, 1-4
views
Heterogeneous Services, 2-13
transparent gateways
accessing Heterogeneous Services agents, 2-2
creating database links, 2-4
Heterogeneous Services, 1-3
installing Heterogeneous Services data dictionary, 2-2
testing connections, 2-4

## U

unsupported functions
generic connectivity, 3-7

## V

V$HS_AGENT view
determining which agents are on host, 2-16
V$HS_PARAMETER view
listing HS parameters, 2-18
V$HS_SESSION view
determining open sessions, 2-17
variables
BIND, 4-4