Progress®



Use the Oracle Data Server

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....13
Purpose....13
Audience....14
Organization....14
Using ABL documentation....15
References to ABL compiler and run-time features....15
References to ABL data types....15
Typographical conventions....16
Examples of syntax descriptions....17
Long syntax descriptions split across lines....18
Complex syntax descriptions with both required and optional elements....19
OpenEdge messages....19
Obtaining more information about OpenEdge messages....20

Chapter 1: Introduction....21
Oracle DataServer overview....21
DataServer components....22
OpenEdge DataServer for Oracle logic....23
The schema holder....24
Security....25
DataServer utilities....26
DataServer demonstration database....26
DataServer configurations....27
SSL enabled DataServer....29
Distributed DataServer applications....29
Software requirements....31
Guidelines for using the DataServer....32
Using the OpenEdge DataServer for Oracle for the first time....32
Using the DataServer to migrate a database from OpenEdge to Oracle....33
Upgrading to the OpenEdge Release 10 DataServer for Oracle....33
Documentation resources....33
Guide to related topics....34

Chapter 2: Initial Programming Considerations....35
Database design issues 36
Oracle and OpenEdge objects and terminology....36
Naming conventions....37
Code pages....37

Support for Unicode....39
Indexes....41
Case-sensitive....44
Index repositioning....44
Initial values....45
Oracle views....45
Database triggers....47
Sequence generator....47
Oracle synonyms....49
Data types....50
Oracle NUMBER data type....52
Oracle CHAR and VARCHAR2 data types....52
Oracle Number data....52
Logical data type and Oracle equivalents....52
ABL supports RAW data type....53
Oracle BLOB and BFILE data types....53
Oracle CLOB and CFILE data types....53
Oracle DATE data type....53
Working with Datetime data types....54
Unsupported data types....62
Arrays....62
Unknown Value (?)....62
Zero-length character strings....63
Record creation....64
Comparing record creation behavior....64
Ensuring record are created in a predictable order....66
Record locking....66
Monitoring locks....68
Transactions....68
Error handling....69
Overflow checking....70
Cursors....70
Static FIND cursor repositioning....71
Stale cursors....71
Allocating cursors....71
ABL issues....72
RECID function....72
ROWID function....72
Progress_Recid change....73
DEFINE BROWSE statement....74
Field lists....74
FIND PREV/LAST statements....75
Compiling ABL procedures....75
Unsupported ABL statements....76
National Language Support....78

Handling lock timeouts....79
RDBMS Stored Procedures....80

Chapter 3: RDBMS Stored Procedure Details....81

Overview....82
Defining native stored procedures to ABL....82
RDBMS stored procedure basics....82
RUN STORED-PROCEDURE details....86
Syntax reference for RUN STORED-PROCEDURE....87
RUN STORED-PROCEDURE statement....87
CLOSE STORED-PROCEDURE statement....89
RUN STORED-PROC statement execution without the LOAD-RESULT-INTO phrase....90
RUN STORED-PROC statement execution with the LOAD-RESULT-INTO phrase....91
RUN STORED-PROC statement with send-sql-statement option....92
RUN STORED-PROC statement with special internal option....93
Data output and retrieval options....94
Return codes....94
Values of output parameters you define when you create a procedure....94
Results retrieved from a database....94
Interfacing with RDBMS stored procedures....96
Retrieving return codes....96
Retrieving output parameter values....97
Retrieving results with cursor arguments....97
Retrieving results sets using proc-text-buffer....98
Defining a view to use as a buffer....99
Loading a result set into a temp-table....102
Handling errors....110
Error messages related to using the LOAD-RESULT-INTO phrase....112
ROWID support....113
Understanding the ROWID Implementation....113
ROWID characteristics....113
Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO....114
Additional ProDataSet support....117

Chapter 4: Additional Features to Enhance DataServer Performance..119

Enhancements overview....120
SQL and ABL queries....120
Bind variables....120
Caching records....121
Oracle hints....121
Using field lists when updating records....122
Setting the data-source result set retrieval option....123
Join by SQL DB....123

Improving join performance for queries with REVERSE-FROM option....125
Improving join performance for queries with FIRST or LAST criteria....125
Skipping schema verification....126
Writing queries for performance....126
Bulk-insert with ABL....127
Closing inactive cursors....128
Query tuning....128

Chapter 5: Configuring the DataServer....135

DataServer modules....135
Configuring Windows modules....137
Configuring the local DataServer in Windows....137
Configuring the DataServer in the Windows host....137
Configuring UNIX modules....138
Configuring the local DataServer on UNIX....139
Configuring the DataServer on the UNIX host....139
Configuring an international environment....140
Configuring the DataServer in the Unified Broker framework....142
Configuring the Unified Broker framework in Windows....142
Configuring the Unified Broker framework on UNIX....143
Establishing the SSL protocol in a DataServer broker instance of the Unified Broker....147
Creating, Maintaining, and Deploying a schema holder....149
Schema holder security....149
Running Oracle....151
Starting the DataServer processes....152
Starting ProBroker distributed DataServer broker with SSL-enabled Progress Networking....153
Creating an empty OpenEdge database....154
Using the DataServer utility to create a schema image....155
Maintaining a schema holder....158
Deploying a schema holder....159
Setting up a schema holder as a Windows service....160
Configuring a self-service DataServer agent in the OpenEdge Appserver and Progress Application Server....160
Configuring the OpenEdge AppServer agent as a self-service DataServer session....161
Configuring the Progress Application Server agent as a self-service DataServer session....161

Chapter 6: Connecting the DataServer....165

Starting the local DataServer....165
Starting the remote DataServer....166
Starting the DataServer in the Unified Broker framework....167
Starting and stopping the ProBroker process from the command line....169
Starting the UNIX client process....170
Starting the Windows client process....171

Connecting a schema holder....171
Connecting a schema holder at startup....172
Connecting through Oracle Networking....175
Optional connection and startup parameters....177
Query tuning with connection and startup parameters....178
Setting a session's timestamp....182
Analyzing application execution with Enhanced Logger....182
Index cursors....197
Local schema caching....199
Connection failures and OpenEdge responses....199
Connection troubleshooting....200
Accessing the DataServer log using qt_debug logging options....201

Chapter 7: The DataServer Tutorial....205

Oracle Utilities menu....206
Managing server attributes....207
Overview of foreign database constraint definitions....209
Constraint definition names....209
Defining constraint definitions in Data Administration....210
Modifying an existing constraint definition....216
Activating or deactivating a class of existing constraint definitions....216
Deleting an existing constraint definition....217
Creating a schema image....217
Updating a schema image....221
Verifying a schema image....223
Changing logical name and connection information....228
Modifying a schema image....228
Modifying table-level information....229
Modifying field-level information....230
Defining the ROWID....231
Changing a code page....233
Deleting a schema image....234
Migrating an OpenEdge Database to Oracle with PRO/SQL....234
OpenEdge DB-to-Oracle utility....235
Preparing a database for the utility....236
Oracle size limitations....236
Naming conventions....238
Oracle tablespaces for tables and indexes....238
Running the OpenEdge DB-to-Oracle utility....239
Handling character length during database migration....247
OpenEdge DB-to-Oracle Incremental Schema Migration utility....249
Adjusting field widths during migration....253
Updating the Oracle database....253
Adjust an Oracle schema image to an OpenEdge database....254

Appendix A: Upgrading DataServer Applications.....255

Upgrading schema holders from Progress Version 9.....255
Manually adding PROGRESS_RECID to Oracle.....256

Appendix B: Environment Variables.....259

Appendix C: Sample Queries.....261

Query 1.....262
Query 2.....262
Query 3.....263
Query 4.....264
Query 5.....265
Query 6.....266
Query 7.....267
Query 8.....268
Query 9.....269
Query 10.....270
Query 11.....271
Query 12.....272
Query 13.....273

Appendix D: Building DataServer Executables.....275

Overview of the building process.....275
Requirements for Building the DataServer Executables.....277
Building executables for the UNIX Client.....278
Replacing your default client executable:.....279
Identifing your new executable with an environment variable and without relocating.....279
Building executables for the UNIX Host.....279
Building a UNIX host executable.....280
Replacing your default server executable.....281
Identifying your new executable with an environment variable and without relocating.....281

Appendix E: DataServer Command Line Utilities and Startup Parameters.....283

Command line utilities for the DataServer.....283
NSCONFIG utility.....284
NSMAN utility.....285
ORACONFIG utility.....287
ORAMAN utility.....289
PROBRKR command line syntax.....290

PROSHUT command.....291

Index.....293

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

Purpose

This manual explains how to use the OpenEdge® DataServer for Oracle. It provides startup instructions and a brief tutorial that introduces the utilities that support the DataServer. Additionally, it discusses database design and programming issues to consider when creating applications that access OpenEdge and Oracle database management systems.

Introduction on page 21 introduces you to the Oracle DataServer and discusses how OpenEdge and WebSpeed applications work with it to access data sources through the ODBC standard. Subsequent chapters provide additional information about using the DataServer. If you are using the DataServer with WebSpeed and with applications written in Progress® SpeedScript®, all information regarding ABL applies to your application.

For the latest documentation updates see the OpenEdge Product Documentation Overview page on Progress Communities:
https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx.

Before you begin

Before you attempt to use the Oracle DataServer, read OpenEdge Release Notes, *OpenEdge Getting Started: Installation and Configuration*, and Initial Programming Considerations on page 35. Also, be sure to follow the step-by-step instructions in Configuring the DataServer on page 135 and Connecting the DataServer on page 165 for installing, configuring, and connecting to Oracle. See the Documentation resources on page 33 for additional information.

Audience

This book is intended for programmers who want to develop OpenEdge applications with ABL that run with Oracle databases. It assumes a fundamental knowledge of both OpenEdge and Oracle.

Organization

Introduction on page 21

Describes the basic architecture of the DataServer for Oracle and presents guidelines for using the DataServer.

Initial Programming Considerations on page 35

Discusses the differences between Oracle and OpenEdge and how the DataServer resolves them.

RDBMS Stored Procedure Details on page 81

Defines relational database management system (RDBMS) stored procedures and describes how to create and implement them in an OpenEdge environment. It discusses various techniques to execute RDBMS stored procedures and Send SQL statements on the MS SQL server and load results sets directly into temp-tables. ProDataSet functionality, available through the use of temp-tables, is also briefly discussed.

Additional Features to Enhance DataServer Performance on page 119

Presents various DataServer performance enhancement techniques you can use, including connection pooling.

Configuring the DataServer on page 135

Presents instructions for configuring the DataServer and creating a schema holder.

Connecting the DataServer on page 165

Presents instructions for connecting the DataServer and a schema holder.

The DataServer Tutorial on page 205

Provides the opportunity to work with the DataServer utilities for Oracle that you use to maintain the schema holder. In addition, it describes the OpenEdge DB-to-Oracle migration utility.

Upgrading DataServer Applications on page 255

Describes the steps required to upgrade from earlier versions of the DataServer to OpenEdge Release 10.

Environment Variables on page 259

Describes the environment variables that affect building and running the DataServer.

Sample Queries on page 261

Contains examples of queries and the SQL statements that the DataServer generates for the Oracle DBMS.

Building DataServer Executables on page 275

Provides instructions for building DataServer executables using OEBuild-based scripts.

DataServer Command Line Utilities and Startup Parameters on page 283

Describes the utilities you use to configure, manage, start, and stop the DataServer host and client.

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all `UPPERCASE`, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever *character* appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever *decimal* appears, this is a reference to the `DECIMAL` data type.
- Wherever *numeric* appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.

Convention	Description
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, ACCUM is a keyword, and *aggregate* and *expression* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } . . .
       | { record [ EXCEPT field . . . ] }
```

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

```
OpenEdge-install-dir/bin/pro
```

2. Press **F3** to access the menu bar, then choose **Help** > **Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose **File** > **Exit**.

1

Introduction

This chapter describes the OpenEdge DataServer for Oracle. It also discusses how OpenEdge and WebSpeed® applications work with the DataServer to access data in an Oracle database through the Oracle Call Interface.

For details, see the following topics:

- Oracle DataServer overview
- DataServer components
- Software requirements
- Guidelines for using the DataServer
- Documentation resources

Oracle DataServer overview

The OpenEdge DataServer for Oracle allows you to access your Oracle database with the Advanced Business Language (ABL), and develop applications within the OpenEdge Studio. ABL is specifically designed to translate knowledge about business operations into software.

The OpenEdge Studio is a set of tools that helps you to maintain data sources and develop applications with graphical user interfaces. When you develop applications with the OpenEdge Studio, you can design a separation of your user interface, business logic, and data sources. OpenEdge Studio provides user interface independence, whether you need a character UI, a Graphical Windows UI, or a Web interface. When you incorporate the DataServer for Oracle with OpenEdge Studio, you are enabling the use of your ABL business logic to access data in your Oracle database.

OpenEdge provides you the tools for the development, deployment, management, and integration of your application. The DataServer for Oracle allows you to implement the OpenEdge features and ABL expansions in applications that run with an Oracle database.

DataServer components

The DataServer is a set of software modules that allows OpenEdge applications to access information in an Oracle database.

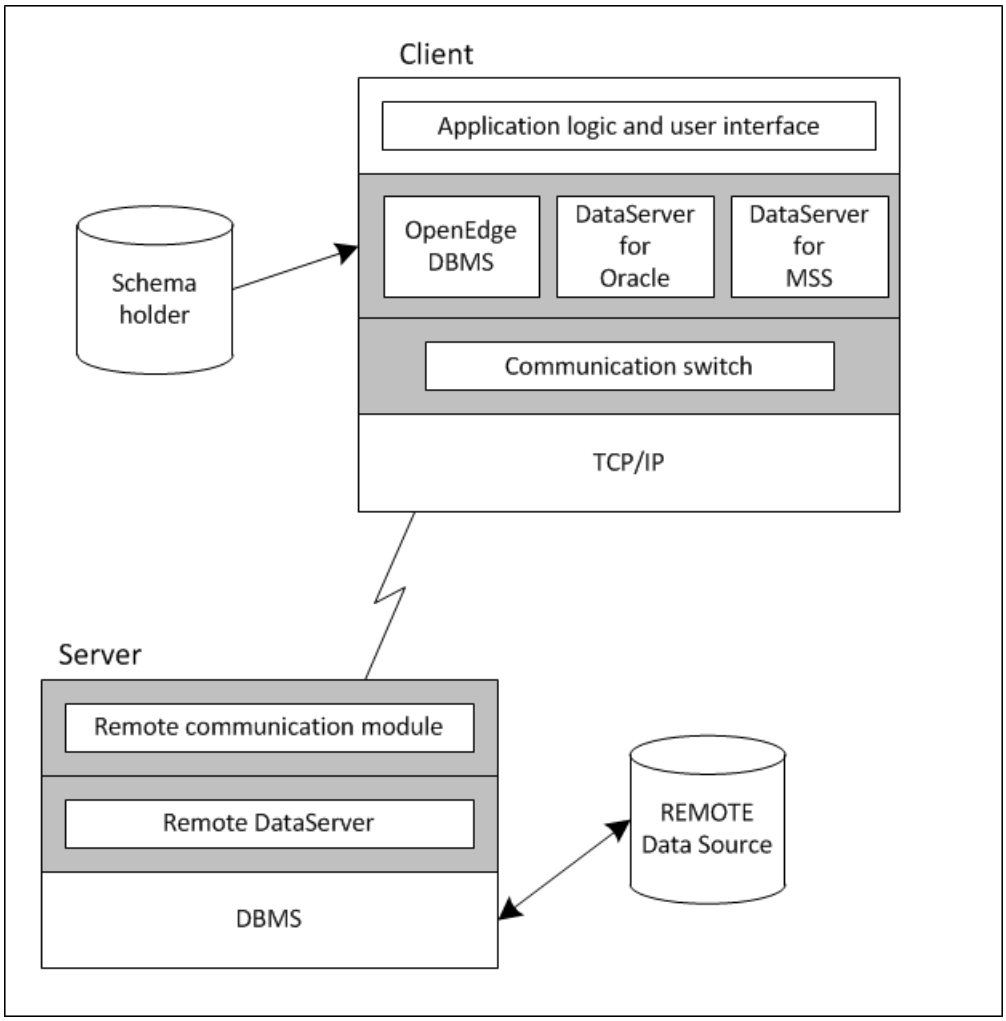
Adding the DataServer to the standard OpenEdge architecture allows the OpenEdge client to access data managed by non-OpenEdge data managers. ABL can then manipulate database records in the same fashion, regardless of the database from which they come. When accessing an Oracle database, the DataServer translates standard ABL code into the appropriate calls to the Oracle database.

The DataServer product consists of several components, some of which are linked with elements of the standard OpenEdge architecture—the OpenEdge client, the DataServer, the schema holder, and the DataServer utilities. These components work together to create and support a software module that allows OpenEdge applications to access an Oracle database.

The DataServer runs in a variety of configurations that can be local, or include OpenEdge or Oracle networking. Depending on its configuration, a DataServer is a single OpenEdge executable or a set of OpenEdge executables that you can distribute across operating systems. See DataServer configurations on page 27 for descriptions of your configuration options.

The following figure illustrates the DataServer modules as they appear in a networked, or remote, configuration.

Figure 1: DataServer architecture

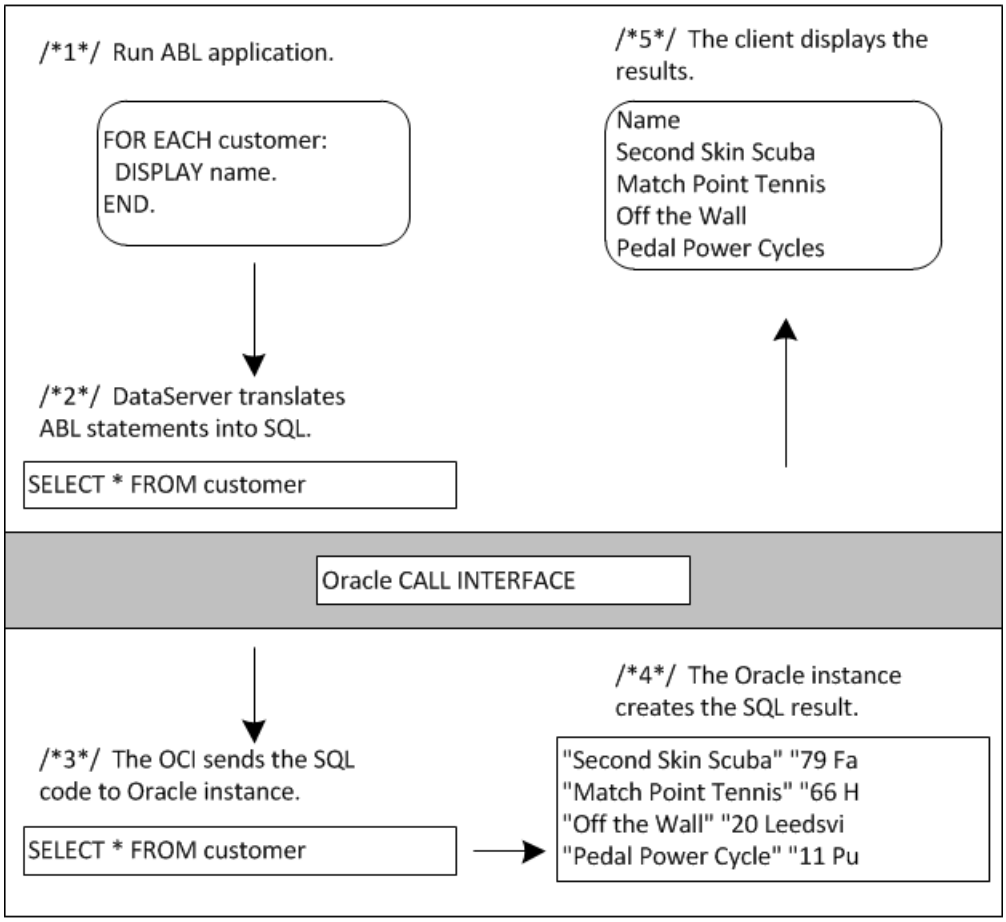


OpenEdge DataServer for Oracle logic

The DataServer places OpenEdge equivalents for the data definitions from an Oracle database into a schema holder. A *schema holder* contains only data definitions for one or more non-OpenEdge databases. When you issue ABL statements, they are compiled into calls to the Oracle database. When the OpenEdge client executes ABL statements and retrieves information from the Oracle database, it relies on data definitions maintained by the Data Dictionary in the OpenEdge schema holder.

The following figure shows the internal logic of the OpenEdge DataServer for Oracle.

Figure 2: The OpenEdge DataServer for Oracle logic



The DataServer uses the Oracle Call Interface (OCI) to access an Oracle instance. An Oracle *instance* provides the software mechanisms for accessing and controlling a database. When you start up the Oracle Database Manager (DBMS) against a database, you create an Oracle instance. See your Oracle documentation for more information on instances.

When you execute an OpenEdge application that accesses an Oracle database, the ABL compiler translates ABL statements in the OpenEdge application into their SQL equivalents. The DataServer then issues the SQL statements to the Oracle instance through the OCI. The Oracle instance processes the SQL statements and returns the results to the application through the OCI.

You can also send SQL statements directly to Oracle. The DataServer passes `SELECT` SQL statements that you reference in an ABL application directly to Oracle without translating them. The DataServer also allows you to issue PL/SQL statements from an OpenEdge application.

The schema holder

The *schema holder* contains information about the data definitions for one or more Oracle databases, or from any other data manager supported by the OpenEdge DataServer for Oracle architecture. The *schema* of a database is a description of its structure, the tables, the fields within the tables, the indexes, stored procedures, and other objects. That information in the schema holder is called the *schema image*. When you use an OpenEdge client with an Oracle database, the DataServer uses the schema image to translate OpenEdge database requests into a format that can be used to access the data in the Oracle database.

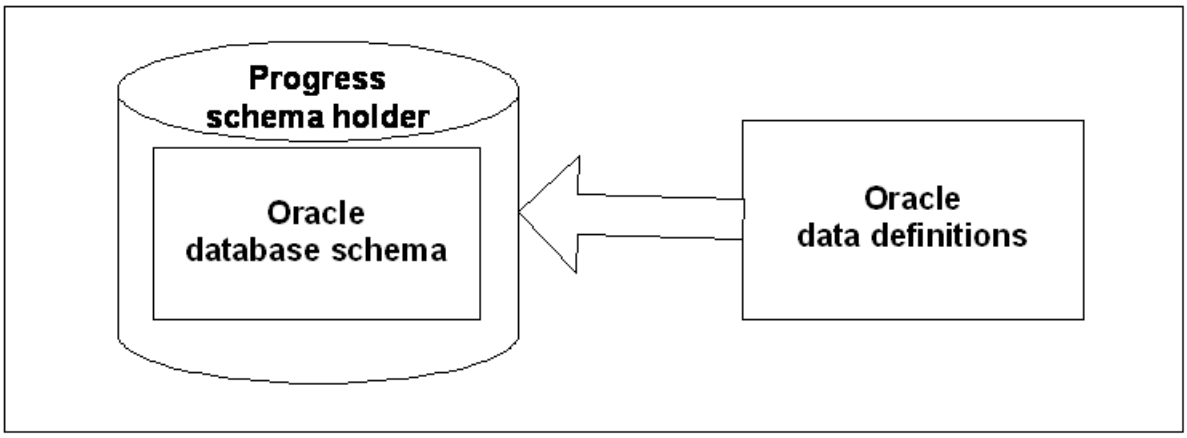
The schema image contains all the database information for developing an OpenEdge application for the DataServer, which lets you develop and compile applications for an Oracle database without being connected to it.

OpenEdge accesses the schema holder only when it compiles procedures and at the beginning of a run-time session for schema caching. *Schema caching* occurs when data definitions are loaded into memory. Typically, the schema holder is idle during a run-time session after the initial schema caching.

Before an OpenEdge client can access Oracle data, you must create a schema holder and load the Oracle database definitions—the schema image—into the schema holder. Then you can use the Data Dictionary to add OpenEdge database features, such as validation expressions or messages.

The following figure illustrates the schema-loading process.

Figure 3: The schema-loading process



Security

Using the DataServer involves following the security guidelines required by both OpenEdge and Oracle. By default, OpenEdge does not impose security on databases, so at a minimum, you follow the guidelines Oracle requires for your applications.

OpenEdge security

The OpenEdge database management system has no minimum security requirements. You can, however, impose security features on any OpenEdge database or schema holder. There are four levels of application security you can impose:

- Database-connection security
- Schema security
- Compile-time security
- Run-time security

For more information about compile-time and run-time security, see *Manage ABL Applications*. For general information about OpenEdge security, see *Introduction to Security and Auditing*.

Oracle security

All users must supply a valid user ID and password to access an Oracle database. In addition, Oracle provides security in the form of distinct privileges that control access to databases and objects within databases. Oracle regulates access to database objects at the table level based on types of tasks. For example, `INDEX` privileges allow the user to create or drop indexes.

The DataServer does not support Trusted Oracle.

To use the OpenEdge DataServer for Oracle, there are three security requirements:

- To create the schema holder for the Oracle database, you must have sufficient privileges to connect and create a session and select privileges for system objects in the Oracle database.
- To run DataServer applications that access an Oracle database, you must have `SELECT` privileges for the sys.dual system table.
- To connect to the schema holder for the Oracle database, you must supply a valid user ID and password for the target database. OpenEdge provides connection parameters that the DataServer uses to pass this information to the Oracle Database Manager.

See Schema holder security on page 149 for a detailed description of the required privileges.

DataServer utilities

OpenEdge provides a set of utilities that allows you to perform certain tasks related to the DataServer. There are utilities for:

- Creating a schema image
- Updating a schema image
- Verifying that the schema image matches the corresponding Oracle definitions
- Editing connection information for a schema image
- Changing the code page for a schema image
- Deleting the schema image
- Running Oracle SQL*Plus
- Migrating an existing OpenEdge database to Oracle
- Creating an incremental schema description of an Oracle database

See The DataServer Tutorial on page 205 for information on how to use these utilities.

DataServer demonstration database

The OpenEdge Sports and Sports2000 demonstration databases allow you to experiment with the DataServer. You can create an Oracle database based on the OpenEdge Sports or Sports2000 databases.

See The DataServer Tutorial on page 205 for information on how to create the Sports demonstration database, which you can use to run sample code when learning to work with the DataServer utilities.

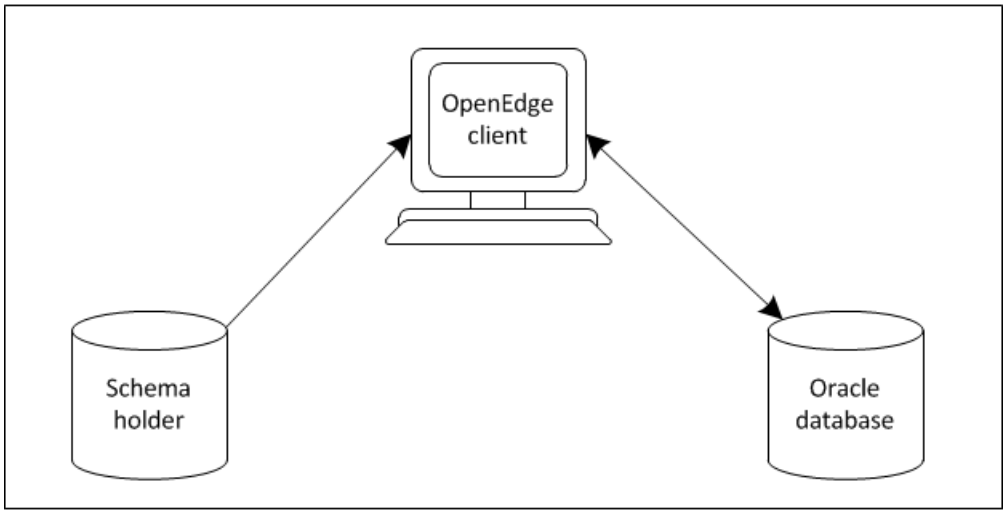
DataServer configurations

The OpenEdge DataServer for Oracle supports many possible configurations. There are two general types of configurations—local and remote—with several variations on each type:

- **Local DataServer** — All the DataServer software modules run on one machine. Your Oracle database can also run on this same machine, or it can run on a separate machine that you access through Oracle Networking.
- **Remote DataServer** — The DataServer software modules run on different machines. Typically, the client module runs on one machine and the DataServer module runs on another. The machine on which the DataServer module runs is called the host machine. The host machine and the machine where the client is running can communicate through OpenEdge networking (TCP/IP).

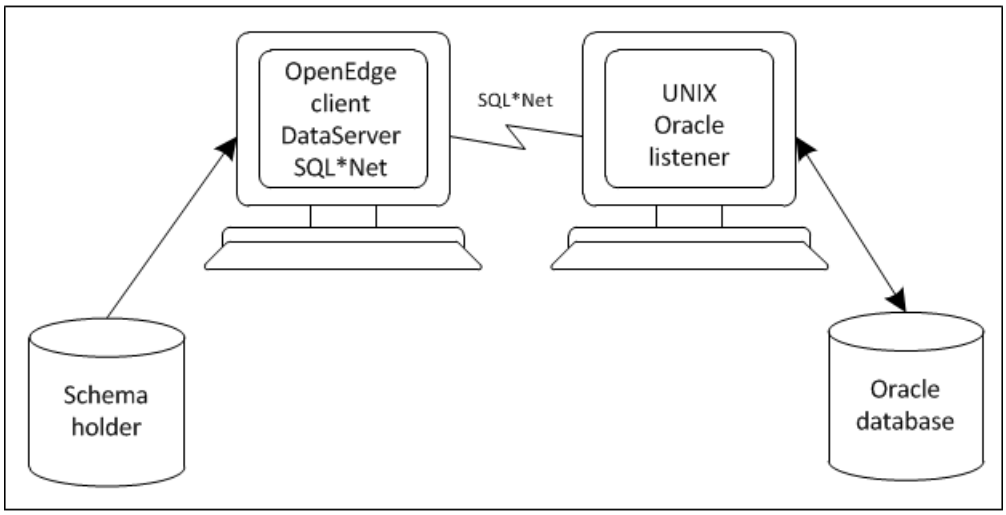
The following figure shows a local DataServer configuration where all the modules run on one machine. In this case, the Oracle database is also local.

Figure 4: The local DataServer



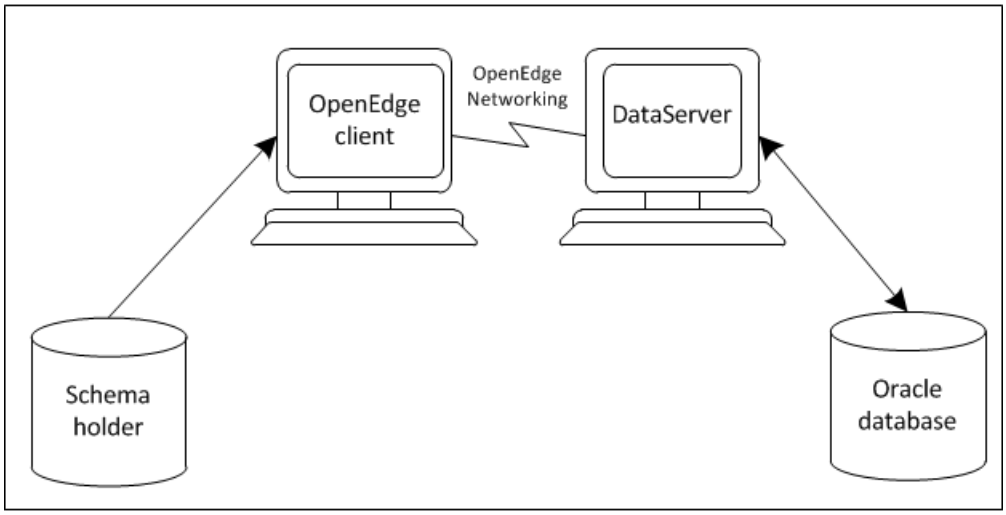
The following figure shows a local DataServer accessing an Oracle database on another machine through Oracle Networking. You must install Oracle Networking on the client machine. In configurations that include Oracle Networking, there are no OpenEdge processes running on the machine where the Oracle database is running.

Figure 5: A local DataServer accessing remote Oracle database through Oracle networking



The following figure shows one possible configuration for the remote DataServer where a client accesses a remote DataServer for Oracle. Here, the Oracle database and the OpenEdge DataServer for Oracle are running on the same machine.

Figure 6: The remote DataServer for Oracle



OpenEdge handles the communication between the client and the DataServer. The client and server processes that make up the DataServer adapt to a variety of network configurations. The previous illustrations showed a few possible configurations. The following table lists all supported configurations. It considers possible client/server combinations and networking options.

Table 1: Supported configurations

Client	Networking	DataServer
OpenEdge on UNIX	None	DataServer, client, and Oracle on the same machine

Client	Networking	DataServer
OpenEdge on UNIX	Oracle	DataServer, client, and Oracle Networking on the client machine, and Oracle on a server
OpenEdge on UNIX	OpenEdge	Client on the client machine, and the DataServer and Oracle on a server
OpenEdge in Windows	Oracle	DataServer, client, and Oracle Networking on the client machine, and Oracle on a server
OpenEdge in Windows	OpenEdge	Client on the client machine, and the DataServer and Oracle on a server

SSL enabled DataServer

Configuring the clent and server components of a remote DataServer connection with Secure Sockets Layer (SSL) communications is optional. Users can maintain both SSL-enabled and non-SSL instances of a DataServer broker environment. However a given instance supports only one type of connection, either secure or non-secure.

DataServer Security is based on the client authenticating the server's identity using a Public Key Infrastructure (PKI) and a symmetric data encryption system. To configure a Broker instance for SSL operation, you must:

- Install a server private key and a public key certificate. OpenEdge provides built-in keys and certificates that are suitable for use on development or demonstration servers; for production machines, you should obtain server certificates from an internal or public Certificate Authority (CA).
- Specify the `keyAlias` and `keyAliasPasswd` parameters in the `UBroker.OR.orabroker1` section of the ubroker.properties file for access to the private key/digital certificate
- Disable session caching of the orabroker using the `noSessionCache` parameter, or enable it with a specified timeout using the sessionTimeout parameter.

For more information see Establishing the SSL protocol in a DataServer broker instance of the Unified Broker on page 147 and *Introduction to Security and Auditing* guide.

To connect to an SSL-enabled OE DataServer component, Client and Servers must have access to a digital (public key) certificate that can authenticate with the digital certificate used by the server, and the client must be configured to send SSL requests. All OpenEdge-managed SSL servers rely on a common OpenEdge key store to manage the private keys and server digital certificates required to support SSL connections from clients. Similarly, most OpenEdge-managed SSL clients and servers rely on a common OpenEdge certificate store to manage the root CA digital certificates that enable them to establish connections to appropriate SSL servers. With the OpenEdge installation, a third party Public/Private key pair is provided for testing. The root CA certificate is located in `$DLC/keys/default_server.pem` and the public key is located in `$DLC/certs/pscca.cer`.

Distributed DataServer applications

OpenEdge has integrated much of its server technology into a single framework, which allows you to administer distributed components as a single system. OpenEdge products such as the WebSpeed® Transaction Server™ are dependent on this single administrative system, known as the Unified Broker framework. It can support DataServers that allow you to include a non-OpenEdge data source into this distributed system.

AppServer applications can use the DataServer to access data from a non-OpenEdge data source. WebSpeed Agents can connect to a non-OpenEdge data source and execute their SpeedScript applications against it. Note that the AppServer and WebSpeed can access DataServers that are not administered by the Unified Broker framework.

In the Unified Broker framework, a single administrative service (the AdminService) controls the various processes required by the AppServer, WebSpeed, and DataServers. This allows you to centralize your resources. The framework also supports a single access point for configuring, running, managing, and analyzing your distributed architecture. In Windows, this utility is the OpenEdge Management or OpenEdge Explorer, which provides a graphical interface to configuration and administrative tasks. On UNIX, you work with the Unified Broker framework through a properties file and command-line utilities. See the The Distributed DataServer using the Unified Broker Framework on page 30and for an overview of the Unified Broker framework, *Configure OpenEdge* for an overview of the Unified Broker framework. For more information on configuring ProBroker, see The Distributed DataServer using ProBroker on page 31.

Distributed DataServer configuration with SSL

The DataServer broker (_probrkr.exe) or an Oracle DataServer instance of the Unified Broker) on a host machine determines the protocol requirements of the clients that connect to server components (_orasrv.exe) started by the broker on behalf of a connecting client. A distributed ProBroker or Unified Broker environment can accepts a `-ssl`startup or command line argument on the host machine to indicate that the types of client requests that the spawned DataServer (_msssrv.exe/_orasrv.exe) can handle should be using ssl protocol over the network. OpenEdge DataServer client should only send SSL enabled requests over the network. A `-ssl` start-up parameter is also accepted by the client to instantiate an ssl connection with the server component. If an OpenEdge DataServer client is started without a `-ssl`l startup parameter and it sends a connection request to a running broker in ssl mode, the broker fails to authenticate the client connection request. However, a message is then returned internally to the client requesting it to send an SSL enabled handshake. If the client is capable of sending an SSL request, the SSL connection is automatically established without specifying the client startup parameter.

See the The Distributed DataServer using the Unified Broker Framework on page 30and *Configure OpenEdge* for an overview of the Unified Broker framework.

The Distributed DataServer using the Unified Broker Framework

The Unified Broker framework is a system administration framework that provides a consistent interface in which specific OpenEdge products such as the Oracle DataServer can be managed. It supports elements that allow you to perform such common administrative tasks as

- Starting and stopping processes
- Managing, configuring, and validating property files

These elements include the OpenEdge Management/OpenEdge Explorer configuration tool, mergeprop utility, and command-line utilities.

The framework also facilities activities that are fundamental to the Oracle DataServer's broker-based technology. For example, the AdminService, the framework's central element, enables supported products like the Oracle DataServer in managing an application's resources.

For details about the Unified Broker Framework, its elements, and the OpenEdge products that employ its features, see *Configure OpenEdge*.

The Distributed DataServer using ProBroker

ProBroker is the original broker framework for spawning server component of the DataServer products. ProBroker is started from a command line or from an operating system script. It uses a traditional “fork/exec” broker model as it receives client connection requests. It is a standalone process rather than part of a system framework. As such, it has none of the management, configuration or validation capabilities of the Unified Broker Framework. There is one additional command, “proshut”, that allows the user to terminate a started ProBroker process.

For a tutorial on configuring ProBroker for DataServer connectivity, see Starting the remote DataServer on page 166

A closer look at OpenEdge Management or OpenEdge Explorer tool

OpenEdge Management or OpenEdge Explorer tool is a graphical user interface that provides an easy way for you to manage OpenEdge servers. OpenEdge Management or OpenEdge Explorer tool runs as a Windows client and works with another administration component, the AdminService, to provide a client/server framework for managing the following OpenEdge servers:

Using OpenEdge Management or OpenEdge Explorer tool, you can:

- Create new instances of OpenEdge servers and configure their property settings.
- Modify property settings of existing OpenEdge server instances.
- Start and stop OpenEdge servers.
- Monitor the status of OpenEdge servers.

For more information about working with OpenEdge Management or OpenEdge Explorer tool, see the OpenEdge Management or OpenEdge Explorer online help.

Introducing the mergeprop utility

The `mergeprop` utility is a command-line utility that supports functionality similar to that supported by OpenEdge Management or OpenEdge Explorer configuration tool. It is an alternative approach to configuring and managing the content of the OpenEdge property files. Property files, such as the `ubroker.properties` file used by the DataServer, store configuration information that specifies and controls the behavior of various components.

For more information, about configuring and connecting the DataServer, see:

- Configuring the DataServer in the Unified Broker framework on page 142.
- Starting the DataServer in the Unified Broker framework on page 167 for more information.

Software requirements

These are the software requirements for running DataServer applications with an Oracle database:

- OpenEdge installation that includes the OpenEdge DataServer for Oracle
- Oracle version supported by the DataServer
- Oracle Networking on the client machine if you want to connect to an Oracle database using Oracle networking

Building and linking customized DataServer executables additionally requires:

- Oracle client libraries

- A C compiler
- A linker for building executables

Guidelines for using the DataServer

OpenEdge supports many capabilities not found in other database management systems, such as backward scrolling and the ability to find the previous or last record in a table. The DataServer supports these and other programming and database features to ensure that your applications work with both OpenEdge and Oracle databases.

The DataServer allows you to use OpenEdge features as extensions to Oracle. Some of the OpenEdge programming and database design techniques that you can implement on your Oracle database using the DataServer are:

- `ROWID` function
- Arrays
- Cursor repositioning
- Case-insensitive indexes

For access to some of these features, you might have to make minor modifications to your Oracle tables. For a discussion of these issues and instructions for modifying Oracle tables, see Initial Programming Considerations on page 35

When you create an Oracle database from an existing OpenEdge database with the OpenEdge DB-to-Oracle migration utility and you select the **Create RECID** option, you can use the `FIND PREV/LAST` statements, which takes advantage of ABL-like cursor behavior.

How you use the DataServer depends on whether you plan to access information in an Oracle database through an OpenEdge or WebSpeed application, or whether you plan to migrate an OpenEdge database to Oracle. The following sections outline these possibilities and point you to the information you need in this manual.

Note: If you are developing a WebSpeed application, all the programming information in this guide applies to your SpeedScript code. For information on connecting WebSpeed Agents, see *OpenEdge Getting Started: WebSpeed Essentials*.

Using the OpenEdge DataServer for Oracle for the first time

Prior to using the DataServer for the first time, software needs to be appropriately installed and configured.

To prepare to use the DataServer:

1. Install the DataServer modules on the machines your configuration requires.
2. Create a local schema holder on the client machine.

Note: See Configuring the DataServer on page 135 for information about where to install DataServer modules and creating a schema holder.

Using the DataServer to migrate a database from OpenEdge to Oracle

A conversion utility is supplied to migrate an OpenEdge database to an Oracle data source.

To set up and use the DataServer:

1. Install the DataServer modules on the machines your configuration requires.
2. Run the OpenEdge DB-to-Oracle migration utility.

Note: See OpenEdge DB-to-Oracle utility on page 235 for specific instructions.

Upgrading to the OpenEdge Release 10 DataServer for Oracle

In order to take advantage of OpenEdge Release 10 features, you must upgrade your schema holder.

To prepare to use the DataServer if you want to use OpenEdge Release 10 features:

1. Install the Release 10 DataServer modules on the machines your configuration requires.
2. Create a schema holder.
3. Upgrade Version 8, 9, or 10g schema holders to Release 10 by dumping data definitions and loading them into the new schema holder.

Note:

See Configuring the DataServer on page 135 for information about where to install DataServer modules and creating a schema holder.

Documentation resources

The following table suggests paths through this manual that accommodate different approaches to using the OpenEdge DataServer for Oracle.

Table 2: How to use this manual

If you are . . .	Read the contents of . . .
New to the OpenEdge DataServer for Oracle	Initial Programming Considerations on page 35, Configuring the DataServer on page 135, Connecting the DataServer on page 165, The DataServer Tutorial on page 205

If you are . . .	Read the contents of . . .
Migrating an OpenEdge database to Oracle	OpenEdge DB-to-Oracle utility on page 235, Initial Programming Considerations on page 35, Configuring the DataServer on page 135, The DataServer Tutorial on page 205,
Upgrading the DataServer to OpenEdge Release 10	Initial Programming Considerations on page 35, Configuring the DataServer on page 135, The DataServer Tutorial on page 205

Guide to related topics

The following manuals in the OpenEdge documentation set contain useful information on different aspects of using a DataServer:

- *Introduction to the Database* explains how to design an OpenEdge database and provides good relational database design guidelines.
- *Progress OpenEdge Studio* and *Develop ABL Applications* provide information on how to develop applications in ABL.

The following table lists topics related to building and using the DataServer and indicates where to find the information.

Table 3: DataServer-related topics

Topic	OpenEdge manuals
Starting up OpenEdge	*Configure OpenEdge* *Manage the OpenEdge Database*
Using the Data Dictionary	*Introduction to the Database* *Manage the OpenEdge Database*
Defining security for an OpenEdge database	*Manage the OpenEdge Database*
Writing applications in ABL	*Progress OpenEdge Studio* *Develop ABL Applications*

2

Initial Programming Considerations

Progress Developer Studio for OpenEdge used in conjunction with the DataServer allows you to develop applications that transparently access data from multiple sources. The OpenEdge DataServer for Oracle achieves database transparency for applications running against OpenEdge and Oracle, but some of the ways it accomplishes this might affect how you develop applications. This chapter discusses the differences between OpenEdge and Oracle that you must consider when planning your applications and designing your databases. In addition, your applications might have to accommodate the strategies that the DataServer uses to resolve these differences.

This chapter discusses equivalencies, differences, and strategies between Oracle and OpenEdge.

For details, see the following topics:

- Database design issues
- Data types
- Arrays
- Unknown Value (?)
- Record creation
- Record locking
- Transactions
- Error handling
- Cursors
- ABL issues
- Handling lock timeouts

- RDBMS Stored Procedures

Database design issues

When you create or modify the OpenEdge or Oracle databases that your applications access, you might have to consider certain issues, such as OpenEdge and Oracle database objects, naming conventions, indexes, and Oracle views. The following sections discuss the differences between OpenEdge and Oracle in these areas, and how the DataServer deals with them.

Oracle and OpenEdge objects and terminology

Oracle and OpenEdge databases share the structural elements common to relational databases, but each system has its own elements. These structural elements are called schema objects in Oracle and database objects in OpenEdge. *Schema objects* or *database objects* are components of the database's logical structure. The two database management systems have different terminology for referring to these objects, as shown in the following table.

Table 4: Oracle and OpenEdge objects

Oracle schema object	OpenEdge equivalent
Table	Table
Column	Field
Row	Record
Index	Index
Sequences	Sequences
Integrity constraint[1]	Validation expression
No equivalent	Validation message
Trigger	Trigger
Stored procedures	Stored procedures (SQL Engine)
View	View (SQL Engine)
Clusters	No equivalent
Oracle Object methods	No equivalent
Synonyms	Synonyms (SQL Engine)

[1] The Oracle Integrity constraint enforces data integrity and business rules. However, unlike ABL validation expression, you cannot include code in the integrity constraint. You choose one of the following checks: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, or `CHECK`.

Naming conventions

One factor to consider when planning for maximum compatibility across OpenEdge and Oracle is the kind of restrictions each has for naming tables and fields.

Field and column names

These are the restrictions for naming OpenEdge fields and Oracle columns:

- **OpenEdge** — Field names can be up to 32 characters long and can consist of alphabetic characters (A-Z and a-z), digits (0-9), and the special characters $, &, #, %, -, and _. A field name must begin with an alphabetic character.
- **Oracle** — Column names can be up to 30 characters long. They can consist of the same characters as OpenEdge names, except % and -.

The set of characters that OpenEdge allows includes all of the characters Oracle allows. The OpenEdge DB-to-Oracle utility replaces characters that Oracle does not accept with the underscore (_). For example, the OpenEdge `custnum` field maps to the `CUST_NUM` column in Oracle.

Table names

There can be more than one Oracle table (or view) with the same name within a database, because Oracle qualifies tables by owner as well as by name. OpenEdge requires unique table names.

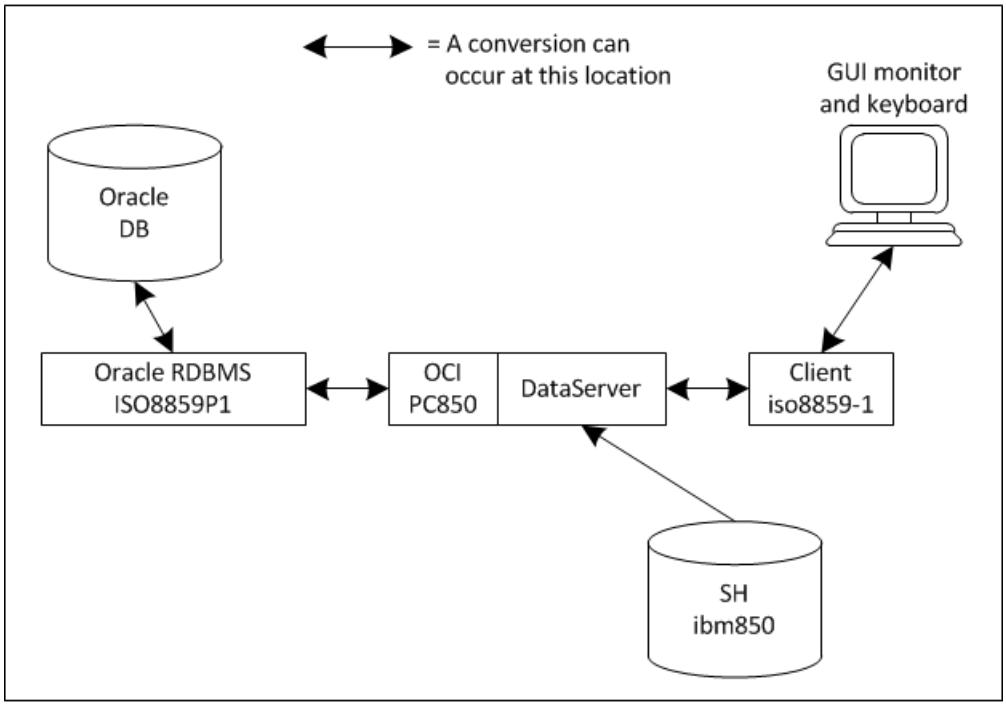
OpenEdge resolves nonunique table names for you. When OpenEdge encounters matching table names while creating or updating a schema image, for the second and subsequent occurrence of a table name, it names the corresponding Oracle table **table**-1, **table**-2, etc. For example, if OpenEdge encounters a second instance of a table named employee in the Oracle database, it names the corresponding schema-holder table employee-1. The Table Properties Sheet in the Data Dictionary shows you which Oracle table maps to the schema holder table.

Code pages

When you access Oracle through the DataServer, the DataServer usually retrieves character data as determined by the code page (character set) that Oracle is using. The DataServer is linked with the Oracle Call Interface (OCI) and receives data from the OCI, thus the code page that the OCI uses determines the data's code page. See the Oracle documentation for more information.

The following figure shows a possible configuration of code pages for the DataServer components and processes.

Figure 7: DataServer processes and code pages



In some configurations, the instance might not be using the same code page as the OCI. In this case, conversion between the two code pages occurs when the Oracle RDBMS sends character data to the OCI.

For OpenEdge applications accessing the DataServer, the schema image identifies the code page of the character data. You must set the code page in the schema image to match the code page used by the OCI. Although the DataServer is linked to the OCI, it cannot automatically determine which code page it uses.

The code page name that you specify in the schema image must be the name by which OpenEdge recognizes the code page. For example, in this configuration, the OCI uses a code page named PC850. This is the same code page that OpenEdge recognizes as ibm850. You then specify ibm850 as the code page for the schema image. You must also specify a collation table that has been defined for the specified code page in the convmap file.

The default code page setting for the schema image is ISO8859-1. You can specify a different code page for the schema image:

- When you create the DataServer schema image for the Oracle database.
- At any time. However, because changing the code page does not affect the data already in the database, writing data to your database using a different code page can corrupt your database.
- When you load a new schema image with a specified code page into an existing schema holder. In this case, the newly loaded schema's code page overrides the schema holder's original code page.

You cannot use the PROUTIL utility to change the database code page. If you are using one schema holder to hold the schema images of several databases, you can specify a different code page for each schema image.

Note: Do not load data definition (`.df`) files that contain character translation tables into the schema holder. Using a translation table that is different from the internal table can corrupt data in a database.

Client code page

The OpenEdge client might be using a different code page than the code page defined for the schema image in the schema holder.

Note: If you are using the OpenEdge DataServer for Oracle to access double-byte character data, the client code page must match the code page of the schema-holder database. The code page of the schema image must match the code page of the Oracle database. See Configuring an international environment on page 140 for more information.

The Internal Code Page (`-cpinternal`) startup parameter determines the code page that the OpenEdge client uses when manipulating data in memory. If the client uses the `-cpinternal` startup parameter, then the DataServer translates between the two code pages. You must verify that the `convmap.cp` file contains a conversion table for the client and the code page setting in the schema image. For example, you might have set the schema image to code page xxx and the client might use code page zzz. The `convmap.cp` file must include a table that converts from xxx to zzz and from zzz to xxx. If `convmap.cp` does not include the appropriate table, OpenEdge allows you to define your own conversion table.

OpenEdge also allows you to define your own collation tables; however, customized collation tables have no effect when you use the DataServer to access Oracle. The Oracle collation tables are in effect when you perform comparisons and sorts.

In some cases, a superset of an ABL query is passed to Oracle and then a process called "client selection" is responsible for doing record sorting and/or comparisons that match server results against the original ABL query. Client selection is dependent on the collation table selected for your Oracle schema image. If the sort criteria and comparison rules of the schema holder's collation table do not match the server's, it is possible that client selection will exclude records in the client's results that the server would have included.

For a complete discussion of how OpenEdge handles code-page issues, see *Internationalize ABL Applications*.

Support for Unicode

Beginning with OpenEdge Release 10.1C, the DataServer for Oracle supports Unicode, a globally recognized standard for software character encoding. The Unicode standard is a collection of characters, code charts, and rules that allows software to be localized or internationalized.

Unicode support provides the DataServer for Oracle with the following functionality:

- Support for Unicode data and data types to and from foreign data sources.
- Migration of OpenEdge databases to foreign data sources that support Unicode. Any OpenEdge database, whether it uses ANSI or UTF-8 character sets, is eligible for migration to a Unicode-enabled foreign data source.
- The ability to retain Unicode data consistency and Unicode type definitions when pulling foreign data sources into OpenEdge UTF-8 schema holders.

The DataServer for Oracle enables this functionality by utilizing Unicode character sets in the Oracle database and the OCI driver implementation. Therefore, you can migrate an OpenEdge database to either an Oracle database defined with a Unicode code page or a Unicode/non-Unicode database that uses Unicode character data types (with encoding AL16UTF16 or UTF8).

Support for Unicode character sets

The OpenEdge database supports the Unicode UTF-8 character set, enabling you to store UTF-8 data in OpenEdge database character columns. The corresponding Oracle Unicode database can support a number of Unicode character set encodings. Progress Software recommends setting the database character set to AL32UTF8 to provide the greatest range of character support and the least amount of translation from your OpenEdge applications. This encoding is consistent with the UTF-8 encoding provided by OpenEdge and removes any potential for data loss that is associated with transforming other Unicode character sets.

Oracle can support Unicode data in `SQL CHAR` data type columns (`CHAR`, `VARCHAR`, `LONG`,and `CLOB`) when the database code page is set to a Unicode character set. Refer to Oracle's Unicode database documentation for more information on supported database encodings.

Support for the UTF-8 character set enables the DataServer to perform a migration from an OpenEdge database to an Oracle Unicode-compliant data source. After performing this migration, you can correctly insert, update, delete, and retrieve UTF-8 data between the OpenEdge database and the Oracle data source.

Support for Unicode data types

In Oracle 9i, the database introduced Unicode data types—`NCHAR`, `NVARCHAR2`, and `NCLOB`. Data stored in columns of Unicode date types are exclusively stored in their own Unicode encodings regardless of database character set.

Beginning in In OpenEdge 10.1C, the DataServer for Oracle recognizes Oracle Unicode data types and processes them during schema pulling and beyond. By recognizing Unicode data types `NCHAR`, `NVARCHAR2`, and `NCLOB`, the DataServer for Oracle performs data conversion through the OCI.

Configuration requirements

In order to use Unicode in the OpenEdge DataServer for Oracle, you must use OpenEdge 10.1C and Oracle Database 9i or later, and OCI Client Libraries for Oracle 9i or later. Additionally, you must perform the following prerequisite tasks:

- **Set appropriate environment variables** — Set the national language support variable to `NLS_LANG=.AL32UTF8` on the system where the Oracle DataServer and Oracle OCI client libraries reside. When setting this variable, you must use the dot (.) notation. `NLS_LANG` can also be set to `UTF-8` in a Unicode environment. However, this is not a recommended setting as it may not be fully compatible with all the character data formatted in your Unicode database. If you set the code page in your schema image to `UTF-8` and `NLS_LANG` is not set to a compatible Unicode value, the DataServer for Oracle will fail to connect to Oracle using that schema holder.
- **Prepare your Unicode Schema Holder** — If you create a new OpenEdge application to be Unicode-enabled for migration to an ORACLE DataServer, ensure that you create the database from a Unicode source. A copy of the empty OpenEdge `UTF-8` database can be obtained from `$DLC/prolang/utf/empty`. If you use the Create DataServer Schema ORACLE DataServer utility to create a DataServer schema and you plan to set the schema holder's code page to `UTF-8`, the schema holder must also be derived from a copy of the empty `UTF-8` database located at `$DLC/prolang/utf/empty`.

Note: The migration process derives a schema holder this way automatically if you set the code page of your migration to `UTF-8`.

The following command would create an empty OpenEdge database named "empty" that could be used to describe a Unicode-enabled foreign data source:

```
prodb <OpenEdge utf-8 schema holder> $DLC/prolang/utf/empty
```

Note: The best way to ensure that the database path is correct for utilities that create a Unicode-enabled database is to set the $DLCDB environment variable to $DLC/prolang/utf. This ensures that databases will be constructed with Unicode enablement. If you do not set $DLCDB, you will need to reference the $DLC/prolang/utf directory explicitly whenever you want to create a Unicode-enabled database.

- **Use appropriate startup parameters** — To process Unicode data in your client application, set your `cpinternal` client code page to `utf-8`. If you are executing file I/O against your Unicode data, set your client's `cpstream` code page to `utf-8`. You also might consider several other available client code page settings for Unicode. For more information, see *Internationalize ABL Applications*.
- **Prepare OpenEdge databases for Unicode** — Your OpenEdge database does not need to be Unicode-enabled in order to be migrated to an ORACLE DataServer. The migration process will convert data from any ANSI code page to Unicode as part of the migration if you chose to move data during a migration and have specified a Unicode destination. If you have existing data in an OpenEdge database that you wish to convert to Unicode prior to migration, you can convert your non-Unicode data using one of two conversion utilities:
  - Data Administration dump and load utility
  - `PROUTIL CONVCHAR` character conversion utility

  For more information on these utilities, see *Internationalize ABL Applications*

Implications of data widening

If you Unicode-enable an OpenEdge database represented by an ANSI code page, you must consider the impact of potential data widening when that data is converted to Unicode during migration. You should also consider the impact of any future data widening if data from certain locales is expected to increase the byte count of your data. These implications are further discussed in OpenEdge DB-to-Oracle utility on page 235. Specific considerations for handling character length are discussed in Handling character length during database migration on page 247.

Indexes

You create and maintain all indexes from within Oracle. When you create a schema image from a target Oracle database, OpenEdge automatically copies the Oracle index definitions. However, note that function-based indexes will not be copied within the schema image.

Database index key widths

Updates in the 10.1B OpenEdge database enable existing index key limits, currently set at approximately 200 bytes, to be widened. For example, this enhancement enables databases with a block size of 4K or 8K to support a maximum index key of approximately 2000 bytes. Also, the index key width expansion extends the maximum key size supported in the OpenEdge client that can be used with databases exported to foreign data sources.

When you are migrating OpenEdge keys to a foreign data source, the key sizes cannot be larger than those supported by the foreign data source. Because the maximum key size is data-source dependent, you should consider your target data source's capacity with respect to the maximum key-size capacity before you perform a migration.

Indexes allow you to use the OF keyword in ABL with FOR EACH and FIND statements. Using the OF keyword improves the readability of your code. The OF keyword is a shorter version of a WHERE clause. You can use OF only when you have a field of the same name in two tables and this field is a unique index in at least one of the tables. You can then write the following statement:

```
FOR EACH customer OF order:
```

Index definitions support ABL USE-INDEX modifier. ABL translates USE-INDEX to ORDER BY for DataServer operations. For example, if you define city-dept as an index on the city and department fields without using the single shot query mechanism, the following ABL statements are equivalent when accessing an Oracle database:

```
FOR EACH employee USE-INDEX city-dept:
```

```
FOR EACH employee BY employee.city BY employee.department:
```

The following examples illustrate the behavior of the index definition if the single-shot query mechanism is used.

If single-shot query mechanism is used, the following query results in a server-side join:

```
FOR EACH customer NO-LOCK USE-INDEX custnum WHERE custnum = 1,
FIRST order OF customer USE-INDEX custorder:
```

If the single-shot query mechanism is used, the following query results in a client-side join

```
FOR EACH customer NO-LOCK WHERE custnum = 1,
FIRST order OF customer
BY customer.custnum BY order.custnum BY order.ordernum:
```

Note:

If you do not specify USE-INDEX, your query will return records in an unpredictable order. Your application might not require predictable ordering, but if it does, be sure to include USE-INDEX in your query definition.

Progress recommends that you use a unique index as a USE-INDEX modifier when using it with a FOR EACH query that joins FIRST or LAST on a child table. If the index is not unique, then DataServer determines the FIRST or LAST of the joined table by appending the ROWID key to the USE-INDEX key components. However, ROWID key is not appended if NO-QUERY-UNIQUE-ADDED is specified in the QUERY-TUNING parameter of the FOR EACH query making the FIRST or LAST criteria unpredictable.

Oracle chooses which index, if any, to use when the OpenEdge application accesses information in the Oracle database. However, the DataServer passes an index hint to Oracle that specifies the index to use for a query and in which order to read the index. The hints take the form of comments in the SQL code generated by the DataServer.

The DataServer issues index hints to Oracle according to two guidelines:

1. If you use ABL `USE-INDEX` modifier in your code, the DataServer generates a hint telling Oracle which to use. The DataServer considers the direction of your query and whether you declared the first component of your index to be ascending or descending. The DataServer then issues an SQL statement to Oracle that it should read the index either forward or backward to ensure that it retrieves the records in the order you specified.

   By including the `USE-INDEX` modifier in your ABL code, you can enhance Oracle performance, especially in cases where your application returns records in a descending order.

2. If you do not use the `USE-INDEX` modifier, the DataServer might generate an index hint based upon the `WHERE` or `BY` option. If the `WHERE` clause has one of the following elements, the DataServer generates an index hint based on the `BY` option:

   - The not equal operator (< >)
   - A function
   - An expression,

   For example, the DataServer passes an index hint to Oracle to use `custnum` for the following query:

```
FOR EACH customer WHERE customer.address = "55 Cambridge"
  BY customer.custnum:
```

   If you issue a query that includes `BY` options, the DataServer considers whether the fields for the `BY` option participate in a compound index and generates an index hint to Oracle to use that index if the `WHERE` clause does not imply a different index.

You can prevent the DataServer from passing hints to Oracle by using the `NO-INDEX-HINT` option for the `QUERY-TUNING` phrase or by using the `-noindexhint` startup parameter. See Query tuning on page 128 and Oracle hints on page 121 for more information.

Note: An ABL `INDEX-INFORMATION` cannot be used against the Oracle DataServer. The DataServer does not inform ABL which index or indexes Oracle uses to perform a query, therefore this attribute does not contain any valid information.

Leading and trailing spaces

ABL and Oracle handle leading or trailing blanks in an indexed column differently. For example, when you attempt to create a record in Oracle and include leading or trailing blanks in a column that participates in a unique index, Oracle returns a message that there is a duplicate unique key. To address this, the DataServer, by default, trims leading or trailing blanks when you use them in a `WHERE` clause. If you want to specify leading or trailing blanks, specify the `-znotrim` startup parameter when you start the OpenEdge client session.

Case-sensitive

OpenEdge allows you to set the attributes of a field so that it is either case sensitive or case-insensitive (case insensitive being the default). The DataServer makes this feature compatible across OpenEdge and Oracle databases. When a field is not case sensitive, OpenEdge does not distinguish between uppercase and lowercase letters for that index when sorting or matching data. In general, this flexibility in an application makes data entry easier for end users since they can enter lowercase or uppercase versions of an index. However, when you want to enforce an uppercase/lowercase distinction in your applications, set the attribute to case-sensitive. Oracle supports case-insensitive indexes by using the `UPPER(column name)` function as an index component.

After the OpenEdge to Oracle migration process, if you perform a schema-pull operation, the columns migrated as case-insensitive, which were used as index components, get appropriately pulled as case-insensitive columns into the schema holder, but, the other columns that did not participate in indexes do not get any indication from the Oracle server of their case-insensitivity. Because Oracle databases are assumed to be case-sensitive by default, when the **Update/Add Table Definitions** option is selected during the Schema pull operation to pull table definitions directly from an Oracle server, the unindexed table columns are assumed to be case-sensitive, even if they were migrated as case-insensitive and you had not made any changes to the table definitions you pulled. If the generated r-code was compiled before the schema pull operation, it would have a schema mismatch with the pulled server definitions. This is because the r-code takes into account case-sensitivity during the schema check.

To avoid schema mismatch during r-code execution, run the **Adjust Schema utility** against the original OpenEdge database after the **Update/Add Table Definitions** operation on tables that were originally migrated with unindexed case-insensitive columns. This causes the schema holder to be adjusted with the case-insensitive attribute of the column originating from the migrated OpenEdge database.

Index repositioning

The DataServer supports index repositioning, which allows you to scroll through a query result set. You must define the query as `SCROLLING` and open the query with the `INDEXED-REPOSITION` option as the following syntax examples show:

Syntax

```
DEFINE QUERY query FOR buf-name SCROLLING.
```

Syntax

```
OPEN QUERY query FOR EACH record-phrase INDEXED REPOSITION.
```

The DataServer enforces the ordering of the query according to the index that the ABL compiler chooses when processing the query. If the index is not unique, the table that you query must have a unique record identifier so that the DataServer can ensure that duplicates order predictably. The record identifier that the DataServer uses is the same one that supports the OpenEdge `ROWID` function, that is, a `PROGRESS_RECID` column, the native `ROWID`, or a unique integer index. Performing an index reposition might cause the DataServer to issue a new query to Oracle. This new result set might contain rows that were added or changed since the original query was ordered. Note that by default, you can only scroll through the result set returned by the query. That is, you cannot scroll beyond the boundaries of the result set returned by the query. You cannot scroll through the rest of the database. If you wish to scroll outside the boundaries of the result set, you must add the `-Dsrv srv-get-prev` switch. This indicates to the DataServer, that it should generate new SQL to satisfy a `GET PREV` following a reposition which would have otherwise been outside the result set.

Initial values

The DataServer preserves settings for initial values of columns in the Oracle database when you create or modify a schema holder. You can then use the Data Dictionary to modify the initial value for that column. Unlike most other changes that you make to the schema holder, the modified initial value is retained during subsequent updates of the schema. If you are inserting records into the Oracle database outside of an OpenEdge application, you might also need to change the default value in Oracle.

Oracle views

Oracle schema objects include views. A *view* is a presentation of data in one or more tables. The following table lists the level of support available for various types of views.

Table 5: Supported Oracle views

View	Supported ABL

All columns from a single table:
`SELECT custnum, name, postalcode`
`. . .`
`FROM customer`
	Full support for this view. Use the `USE-INDEX` option with `FIND NEXT` statements to get the same results you would expect from ABL.

Some columns from a single table: `SELECT custnum, name FROM customer`	Full support for this view. Use the `USE-INDEX` option with `FIND NEXT` statements to get the same results you would expect from ABL.

All columns from a single table with an expression:
`SELECT custnum, UPPER(name),`
`postalcode . . .`
`WHERE postalcode < 20000`
`FROM customer`
	Full support for this view, except that you cannot update the column with the expression (`name`, in this example).

View	Supported ABL

All columns from a single table with an aggregate or function:
`SELECT custnum, name, postalcode`
`. . .`
`WHERE postalcode < 20000`
`FROM customer`
`GROUP BY postalcode`
	The only supported statements are:
	`FOR EACH . . .`
	`NO-LOCK | SHARE-LOCK:`
	`FIND . . . NO-LOCK | SHARE-LOCK:`

Columns from multiple tables:
`SELECT custnum, name,`
`postalcode, ordernum`
`FROM customer, order`
`WHERE salesrep = 'HXM'`
	The only supported statements are:
	`FOR EACH . . .`
	`NO-LOCK | SHARE-LOCK:`
	`FIND . . . NO-LOCK | SHARE-LOCK:`

Oracle views appear as tables in the Data Dictionary's table list for the schema image, not as views. The Data Dictionary's SQL View Report does not list Oracle or other non-OpenEdge views. Nor can you access them through the PRO/SQL menu functions.

In addition, ABL does not allow you to undo the deletion of a record with a view name inside a subtransaction block, so you must perform the deletion inside a transaction block. If you delete a view in a subtransaction block and then try to undo the deletion later, ABL returns a run-time error. See *Develop ABL Applications* for information on subtransactions.

Multi-table views

The DataServer supports direct access to multi-table views. Use the following ABL syntax to read rows from multi-table views:

Syntax

```
FOR EACH view-name { NO-LOCK | SHARE-LOCK } :
```

You cannot use other ABL queries, such as the `DEFINE QUERY`, `OPEN QUERY`, `GET`, and `DEFINE BROWSE` statements, to access multi-table views.

ABL cannot recognize whether a view in an Oracle database is a multi-table view. Although the DataServer copies multi-table views into the schema image, ABL returns run-time errors if you try to update them with an OpenEdge application.

Views containing aggregates

The DataServer supports access to columns in views that contain aggregates or functions only when the affected column has a name associated with it. Assign specific names to the columns when you define an Oracle view. For example, the following SQL statement names a computed column in a view definition:

```
CREATE VIEW view-name AS SELECT COUNT(*) cust_count
  FROM customer
```

Use the following ABL syntax to read rows from views that contain aggregates or functions:

```
FOR EACH view-name { NO-LOCK | SHARE-LOCK } :
```

You can also access the view by using the `RUN STORED-PROC send-sql-statement` option to send an SQL statement to select the data from the view. You can access a view by using the `send-sql-statement` option without adding index definitions for the view in the schema holder. See the Initial Programming Considerations on page 35 for more information.

Database triggers

Database triggers are code that an application associates with a database object and an action. For example, writing a record can cause code associated with that object or action to execute. The DataServer allows your applications to access both OpenEdge and Oracle triggers. In an application that combines both OpenEdge and Oracle triggers, ABL trigger (`CREATE`, `UPDATE`, `DELETE`) executes first. As each OpenEdge trigger executes successfully, the DataServer passes the request to Oracle, and the Oracle trigger (`INSERT`, `UPDATE`, `DELETE`) executes. In the case of an Oracle `BEFORE` trigger, the Oracle trigger executes after the DataServer passes the request but before the insert, update, or delete occurs. When you use the OpenEdge DB-to-Oracle utility to convert an OpenEdge database to Oracle, the utility does not convert ABL triggers into Oracle triggers. The triggers remain in ABL, and are part of the schema holder.

Sequence generator

A *sequence generator* is a database object that provides incremental values within any integer range. You can specify any positive or negative increment. The DataServer supports the Oracle sequence generator. To define Oracle sequences, you use the SQL `CREATE SEQUENCE` statement and indicate the name, minimum and maximum values, increment, and whether numbers are reused. For example, this is the code for creating a sequence named *`seq-table-name`* that starts with 1 and increments by 1:

```
CREATE SEQUENCE seq-table-name START WITH 1 INCREMENT BY 1
```

Note: Do not define Oracle sequences with names ending in `_SEQ` unless this manual instructs you to do so. The DataServer uses Oracle sequences whose names end in `_SEQ` for internal purposes.

Comparing OpenEdge and Oracle sequence generator features

The following table compares the features of the OpenEdge and Oracle sequence generators. An application that relies on sequence generators can access an OpenEdge database and, through the DataServer, an Oracle database. See the chapter on database access in *Develop ABL Applications* for information on defining and using OpenEdge sequences in your database applications.

Table 6: OpenEdge and Oracle sequence generators

Characteristic	OpenEdge	Oracle
Maximum/minimum range	Yes	Yes
Ascending/descending	Yes	Yes
Adjustable incrementation	Yes	Yes
Database object	Yes	Yes
Loses a number when a record creation fails	Yes	Yes
Sequence is disrupted during dumping or loading	No	Yes

If you add a sequence to a table in your supporting Oracle database, you must update the schema image to reflect this addition. See The DataServer Tutorial on page 205 for instructions on updating a schema image.

There are two ABL functions that provide information about database sequences, `NEXT-VALUE` and `CURRENT-VALUE`. When you use the DataServer, you can use these functions to get information about sequences in your Oracle database, but you must use the `NEXT-VALUE` function first in the same session you use the `CURRENT-VALUE` function.

The INT64 data type effects on Oracle database sequences

An OpenEdge database by default defines a sequence field in the schema as an `INT64` data type. As a result, sequences migrated from a new OpenEdge database to a target Oracle DataServer automatically expand to support 64-bit data types.

The `INT64` data type enables an Oracle DataServer to maintain OpenEdge compatibility with and provide 64-bit numeric capability for sequences when they are used to generate `ROWIDS`.In the Oracle DataServer, `ROWIDS` are supported by the 64-bitnative `NUMBER` data type. For details about `NUMBER` data type support, see Data types on page 50.

When generating sequences, remember that:

- Sequences migrated from a new OpenEdge database expand to `INT64`.
- Sequence functions return an `INT64` data type, using the `NUMBER` data type.
- An overflow condition can occur and subsequently an error message can appear on a client machine when a sequence generator is defined as a 64-bit value, but the value into which it is to be loaded is defined as a 32-bit data type (OpenEdge Release 10.1A or earlier).
- When an Oracle database is the target database, and the sequence's upper limit is defined as the `Unknown value (?)` in the source database, the migration utility will specify the sequence's upper limit value as 922337203685775807. However, Oracle continues to generate sequence definitions with the `NOMAXVALUE` clause when the value for the upper limit is defined as the `Unknown value (?)` in the `delta.df` file.

The value generated through this clause is the real upper limit of the sequence value in the OpenEdge client as distinguished from the upper limit defined through the INT64 data type.

Oracle synonyms

Oracle allows you to define a synonym for a database object. For example, an Oracle database might have a table called NY_CUSTOMER with a synonym CUSTOMER. Applications can refer to the table as CUSTOMER. Using synonyms in applications results in generic, more maintainable code. The DataServer supports Oracle synonyms for these objects:

- Tables
- Views
- Sequences
- Buffers
- Functions
- Stored procedures
- Packages

The DataServer does not support synonyms that point to other synonyms.

Your OpenEdge DataServer applications can reference Oracle objects by their defined synonyms. The DataServer supports synonyms by allowing you to include them when you build the schema image for an Oracle database.

Note: If you use a synonym for a table that has a PROGRESS_RECID column, you must also define a synonym for the corresponding sequence generator and include the synonym in the schema image. For example, if the customer table includes customer_SEQ and you define a synonym, CUST, you must also define and include in the schema image a synonym for the sequence generator, such as CUST_SEQ.

Synonyms and distributed databases

Support for synonyms extends to distributed Oracle databases. You can include in the schema image synonyms that point to objects in remote databases as long as you select the links to those databases when you create the schema image. However, you can only use synonyms defined in linked databases that point to objects in that same database.

Synonyms for stored procedures and functions in linked databases are not supported. The following table describes the support for synonyms in local and distributed (remote) databases.

Table 7: Supported synonyms

Synonym	Points to . . .	Supported
CUSTOMER in local DB	NY_CUSTOMER in local DB	Yes
CUSTOMER in local DB	NY_CUSTOMER in remote DB I	Yes
CUSTOMER in remote DB I	NY_CUSTOMER in remote DB I	Yes
CUSTOMER in remote DB I	NY_CUSTOMER in remote DB II	No

Data types

Oracle data types differ from OpenEdge data types. However, each Oracle internal data type supported by the DataServer has at least one OpenEdge equivalent.

The DataServer translates the Oracle data types into OpenEdge equivalents. When an Oracle data type has more than one OpenEdge equivalent, the DataServer supplies a default data type. The schema image contains the OpenEdge data definitions for the Oracle columns, which you can modify by using the Data Dictionary. For example, the DataServer assigns the NUMBER data type the OpenEdge equivalent, DECIMAL. You can then change the data type from DECIMAL to either INTEGER, INT64, or LOGICAL.

Note: In 10.1B, the OpenEdge INTEGER and INT64 data types, respectively, can each be recognized by the Oracle NUMBER data type.

See Modifying a schema image on page 228 for an explanation of how to change OpenEdge data types in the schema image.

Note: You cannot change the data type of a stored procedure parameter. Although you can use the Data Dictionary to view the stored procedure properties in the schema holder, you cannot modify them.

Oracle allows users to define their own data types, known as *external* data types. Oracle converts these external types to an equivalent internal type. For example, a FLOAT data type maps to NUMBER. The DataServer also considers it to be a NUMBER and maps it to DECIMAL in the schema image by default. However, beginning in OpenEdge Release 10.1B, you can choose to update the NUMBER definition to an alternative INT64 data type. For more information, see the following table.

The following table lists the Oracle internal data types supported by the DataServer and their OpenEdge equivalents. The table also shows the default equivalent supplied by the DataServer for those Oracle data types with more than one OpenEdge equivalent. The sections directly following the table provide additional details about several of the Oracle data types and each data type's OpenEdge equivalent.

Table 8: Oracle and OpenEdge data types

Oracle data type	OpenEdge equivalent data type
CHAR	CHARACTER
CFILE	CLOB
CLOB	CLOB
VARCHAR2	CHARACTER
NCHAR[2] NVARCHAR2[2]	CHARACTER
NCLOB[2]	CLOB

[2] Schema image code page must be UTF-8 to support these data types.

Oracle data type	OpenEdge equivalent data type
NUMBER	DECIMAL INTEGER INT64 [3] LOGICAL
NUMBER (10) [4]	INT64
NUMBER (11) [4]	INT64
NUMBER (7,2) [5] NUMBER(13,2) [6] NUMBER (20*) [7]	DECIMAL
CURSOR	INTEGER
DATE	DATE INTEGER
LONG	CHARACTER
RAW (limited support)	RAW
LONG RAW	RAW
BLOB	BLOB
BFILE	BLOB
FLOAT	DECIMAL
DATETIME (CHAR) [8]	CHARACTER
TIME	INTEGER
LONG CHARACTER	CHARACTER
UNDEFINED	CHARACTER
ROWID	CHARACTER
LOGICAL	LOGICAL

[3] Starting with OpenEdge Release 10.1B, Oracle Data Servers users have the option to employ the INT64 data type, as needed, to better accommodate foreign column definitions. However, it is strongly recommended that you plan carefully and only make INT64 data type-related changes to those table columns and program variables that you expect will require large number values.
[4] The number within parentheses indicates the number of digits.
[5] The numbers within parentheses indicate the NUMBER data type has a precision of 7 and a scale of 2
[6] The numbers within the parentheses indicate that the NUMBER data type has a precision of 13 and a scale of 2.
[7] The information in parentheses indicates 20 or more digits.
[8] For more information, see Working with Datetime data types on page 54.

Note: The only Oracle internal data type that the DataServer does not support is ROWID. The Oracle ROWID, however, has its programming equivalent in the OpenEdge ROWID. See the for more information.

Oracle NUMBER data type

OpenEdge supports the Oracle NUMBER data type through the use of the INT64 data type in OpenEdge. This enables a 64-bit value defined in Oracle to be successfully pulled into a schema holder as a native NUMBER for Oracle DataServers. On a schema migration or a schema pull, fields defined as INT64 are converted to the NUMBER for Oracle databases.

Oracle and OpenEdge data types Table identifies support for not only the NUMBER data type (with no precision), but also the scale and precision of the NUMBER column. OpenEdge handles the OpenEdge INTEGER and INT64 data types more efficiently than it does a DECIMAL data type. You can use the Data Dictionary to change the NUMBER data type to any supported compatible the data type from DECIMAL to INTEGER or INT64 in the schema image. See Modifying a schema image on page 228 for instructions.

Oracle CHAR and VARCHAR2 data types

The VARCHAR2 data type does not pad data with trailing spaces. However, Oracle CHAR does pad with trailing spaces. For example, in a CHAR column 20 characters wide, the entry MA includes the two characters and 18 spaces. Your application will find the entry only if a WHERE clause searches for the string that includes MA and the 18 spaces. If the column is a VARCHAR2 column, your application will find the entry if it searches for the two characters. The VARCHAR2 data type is more consistent with the OpenEdge CHARACTER data type.

Oracle Number data

Consider the local version of the Oracle database when accessing NUMBER data. The internal radix (decimal point symbol) varies among versions. For example, some European versions expect the radix to be a comma (,) rather than a period (.). The DataServer issues an ALTER SESSION SET SEPARATOR statement, which might result in stored procedures that you call from ABL seeing a different radix separator.

Logical data type and Oracle equivalents

Oracle does not have a LOGICAL data type. You can change the data type for a field from DECIMAL to LOGICAL in the schema holder. OpenEdge then reads the numeric values stored in the Oracle column, as the following table shows.

Table 9: LOGICAL data type and Oracle equivalents

OpenEdge	Oracle
True	Any nonzero value
False	0

The DataServer stores the contents of an OpenEdge LOGICAL data type in an Oracle NUMBER column as:

- True = 1

- False = 0

If you retain values other than 1 or 0 in the Oracle column, do not write a value to that column as a LOGICAL data type.

ABL supports RAW data type

ABL supports RAW data types for non-OpenEdge databases. For information about programming using the RAW data type, see *Develop ABL Applications* . For information about the specific statements and functions, see *ABL Reference*.

Oracle BLOB and BFILE data types

The Oracle BLOB and BFILE data types are used to store blocks of unstructured data. Database columns of these types store locators. For the BLOB data type, the data is stored within the database. For the BFILE, the data is stored outside of Oracle as an operating system file. For this reason, the BFILE data type is read-only, and cannot be modified or participate in a transaction. For more information on how Oracle defines these data types, see your Oracle documentation. For information about the specific ABL statements and functions that allow access to and manipulation of LOB data and related objects, see *ABL Reference*.

Note: Beginning with OpenEdge Release 11.1, the DataServer for Oracle supports partial read/write capability to BLOB data types using STARTING AT, FOR, and OVERLAY AT n [TRIM] operations.

Oracle CLOB and CFILE data types

The Oracle CLOB and CFILE data types are used to store blocks of unstructured data. Database columns of these types store locators. CLOB and CFILE data that is copied to the OpenEdge client are translated from the codepage defined to the schema image to the -cpinternal code page just like other character data. This includes translation for CLOBs defined in the UTF-8 Unicode character set. If data from an Oracle CLOB is copied to a OpenEdge data type that has a code page attached to its type definition, then the data is converted to that target's code page.

Note: Beginning with OpenEdge Release 11.1, the DataServer for Oracle supports partial read/write capability to CLOB data types using STARTING AT, FOR, and OVERLAY AT n [TRIM] operations.

Oracle DATE data type

The Oracle DATE data type contains both date and time information. By default, in the schema image, an Oracle date column is represented by two OpenEdge fields: a DATE field and an INTEGER field. The DataServer follows this convention when naming the fields: column **column**-1. For example, an Oracle date column named Date_Due converts to two fields named Date_Due and Date_Due-1 in the schema image. Date_Due is a DATE field and Date_Due-1 is an INTEGER field.

OpenEdge converts the time component of the Oracle date to an INTEGER value. To convert the INTEGER value into time, use ABL STRING and TIME functions, as described in *ABL Reference*.

You can change the data type mapping for DATE to CHARACTER in the schema image using the Data Dictionary. If you change the mapping, you must remove the column representing the time as an INTEGER. Use this feature only if you must use the time portion of an Oracle DATE in WHERE clauses.

Note: Do not include the time portion of a date field in an index.

The range of Oracle DATE are the years 4712 B.C. to 4712 A.D. The range of DATE that the DataServer supports are the years 999 B.C. to 9999 A.D. The DataServer converts all years greater than 4712 to 4712. For example, if your OpenEdge application updates a DATE column with the year 4750, the DataServer converts it to 4712 before writing it to the Oracle database.

Working with Datetime data types

DataServer support for datetime data types enhances your ABL application's ability to precisely manage date and time data and to move similar data to and from foreign data sources. This section covers the following topics:

- ABL datetime data types
- Oracle datetime data types

ABL datetime data types

The datetime data types introduce a data type into the OpenEdge language that includes a time component. The DATETIME-TZ data type extends the DATETIME data type with a time zone component. For the DataServer, the ABL datetime data types enable you to manage date and time data from your OpenEdge application and from the foreign data source as one element instead of two separate entities. The timezone offset provides you with additional ability to manage data that spans time zones.

The following table provides definitions of the ABL datetime data types:

Table 10: ABL datetime data types

Data Type	Definition
DATETIME	The DATETIME data type consists of two parts, one an ABL date and one an ABL time. The unit of time is milliseconds from midnight. DATETIME is equivalent to Oracle TIMESTAMP.
DATETIME-TZ	The ABL DATETIME-TZ is a variation of the DATETIME data type that includes a time zone offset. Its Oracle equivalent is TIMESTAMP WITH TIMEZONE.

Oracle datetime data types

Oracle datetime data types provide functionality similar to the ABL datetime data types. The following table provides their definitions:

Table 11: Oracle datetime data types

Data Type	Description
TIMESTAMP	Includes year, month, day values of date and hour, minute, second values of time with an optional fractional seconds portion. This is the Oracle equivalent of the ABL DATETIME field.

Data Type	Description
TIMESSTAMP WITH TIME ZONE	Contains all the values and default precisions of TIMESTAMP including the time zone displacement value. This is the Oracle equivalent of an ABL DATETIME-TZ field.
TIMESTAMP WITH LOCAL TIME ZONE	Contains all the same values and default precisions as TIMESTAMP and are stored without a time zone displacement defined by the database. All stored values are normalized to the database time zone and stored as timestamps without time zone displacement.

Performing data type conversions

Making changes to the data types, either within the ABL or between ABL data types and Oracle data types, will affect how the data is stored. The following tables describe the effects of changing one data type to another.

Converting data types within ABL

The following table describes the effects of converting from one ABL DATETIME data type to another:

Table 12: ABL DATETIME data type conversions

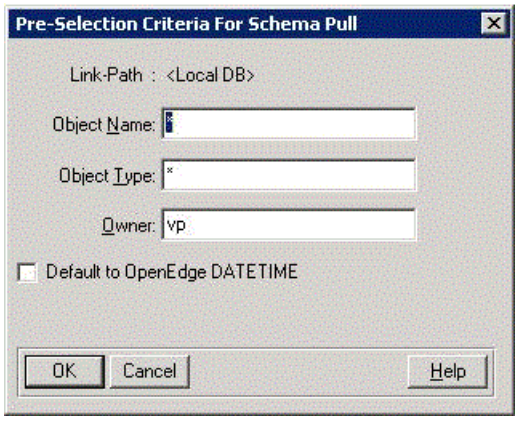
Source data type	Destination data type	Conversion
DATE	DATE	Straight copy.
DATE	DATETIME	Time portion of data type set by default to midnight (00:00:00:000).
DATE	DATETIME-TZ	Time portion of data type set by default to midnight (00:00:00:000) and timezone is set to time zone of the OpenEdge client session.
DATETIME	DATE	Date portion of the DATETIME value is retained. Time portion of DATETIME value is dropped.
DATETIME	DATETIME	Straight copy.
DATETIME	DATETIME-TZ	Destination's time zone becomes time zone of the OpenEdge client session.
DATETIME-TZ	DATE	DATETIME-TZ value is converted to destination's time zone before extracting date portion. Time and time zone portions are dropped. DATE value is converted to the local date of the OpenEdge client session.

Source data type	Destination data type	Conversion
DATETIME-TZ	DATETIME	DATETIME-TZ value converted to destination's time zone before extracting date and time. Time zone portion is dropped and DATETIME value is converted to the local time for the OpenEdge client session.
DATETIME-TZ	DATETIME-TZ	Straight copy.

OpenEdge initializes datetime values with NULL and then treats the whole DATETIME as UNKNOWN if any portion for the datetime value is NULL. By comparison, when Oracle performs data conversion, date values without a time component get a default time of 12:00:00 a.m. (midnight) or a time field set to 0.

Pre-selection criteria for the schema pull

In the Oracle DataServer Dictionary, the default behavior is for Oracle DATE data types in the foreign database to be pulled as a DATE field with a surrogate INT field for the time component. You can override this default treatment of the Oracle DATE data type during schema pull by instead mapping to the DATETIME data type in your Oracle schema image. To override, check the box marked **Default to OE Datetime** in the Pre-selection Criteria dialog box of the schema pull Data Administration tool. For this option, only new tables, not existing tables, in the schema holder are affected by the **Default to OE Datetime** option.



To change tables that contain the previous default mapping from an Oracle DATE to an OpenEdge DATE with surrogate INT field for the time component:

1. Manually change the dictionary type from DATE to DATETIME. Delete the table or definition in your schema holder and re-pull the table with the **Default to OE Datetime** option checked.
2. Go back to the places in your application where the DATE and INT fields were referenced and replace this code with algorithms that utilize the DATETIME data type in place of the old references.

Note: You must not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)

Converting datetime data types between OpenEdge and Oracle

Maintaining proper session time settings is critical to ensuring accurate datetime data type conversion between an OpenEdge application and an Oracle data source. The datetime data types are converted in either a default or non-default mode.

In the default setting, the client time zone context is set by the operating system on which the client operates, providing local context for timestamp values in an OpenEdge application. Oracle DataServer controls the OIC session time, setting it to Universal Coordinated Time (UTC).

In the non-default setting, you can set the application time zone context via the OIC session time. You employ the non-default setting through the `-Dsrv` startup parameter. Utilize the following syntax:

Syntax

```
-Dsrv native-tz-context time-zone-value
```

The time zone value must be a string value in a supported format such as `"local"`, `"-5.0"`, `"-8.00"`, `"dbtimezone"`, or `"Europe/London"`.

You may also choose to invoke the `-Dsrv native-tz-context` parameter without specifying a time zone value. If so, the parameter will not set the default timezone session to UTC. This will allow the OpenEdge client application to set the session time zone, such as through the `RUN STORED-PROC` interface. The Oracle command used to set the OCI session time zone is:

```
ALTER SESSION SET time_zone = 'time-zone-value';
```

Note: OpenEdge DataServer client applications referencing `DATETIME` or `DATETIME-TZ` datatypes mapped to Oracle `TIMESTAMP WITH TIME ZONE` datatype are required to set the session time zone using the `-Dsrv native-tz-context` parameter only by specifying the time zone value in any supported format accepted by Oracle `TIME_ZONE` setting or it can also be set with the `RUN STORED-PROC` interface using `ALTER SESSION SET time_zone = 'time-zone-value'` with the time-zone-value in `"TZH:TZM"` format. Setting the session time zone by other means may result in inaccurate data.

Converting ABL data types to Oracle data types

Converting data from an ABL data type to an Oracle data type might affect the precision of the data that is changed in the conversion.

The following table provides details on converting ABL `DATE` and `DATETIME`, and `DATETIME-TZ` data types to similar Oracle data types:

Table 13: Converting ABL DATETIME types to ORACLE TIMESTAMP types

Source Data Type (OpenEdge)	Destination Data Type (Oracle)	Default Conversion	Non-default Conversion
DATE	DATE	Straight copy of date portion. Time set to midnight up to second precision (00:00:00).	Straight copy of date portion. Time set to midnight up to second precision (00:00:00).

Source Data Type (OpenEdge)	Destination Data Type (Oracle)	Default Conversion	Non-default Conversion
DATE	TIMESTAMP	Straight copy of date portion. Time set to midnight (00:00:00.000) up to millisecond precision.	Straight copy of date portion. Time set to midnight (00:00:00.000) up to millisecond precision.
DATE	TIMESTAMP-LTZ	OpenEdge client session time zone context of the DATE data type is normalized to the Oracle database time zone. The date portion is then stored. Time set to midnight up to millisecond precision (00:00:00.000) and stored.	Oracle Session time zone context (defined by the application to the Oracle client OCI driver) of the DATE datatype is normalized to the Oracle database time zone.Date portion (in session time) is then stored. Time set to midnight up to millisecond precision (00:00:00.000) and stored.
DATE	TIMESTAMP-TZ	Straight copy of date portion. Time set to midnight up to millisecond precision (00:00:00.000). Time zone set to the OpenEdge client session time zone context.	Straight copy of date portion. Time set to midnight up to millisecond precision (00:00:00.000). Time zone set to the Oracle client session time zone context.
DATETIME	DATE	Straight copy of datetime portion up to second precision. Millisecond portion is dropped.	Straight copy of datetime portion up to second precision. Millisecond portion is dropped.
DATETIME	TIMESTAMP	Straight copy of the datetime up to millisecond precision.	Straight copy of the datetime up to millisecond precision.
DATETIME	TIMESTAMP-LTZ	OpenEdge client session time zone context of the DATETIME data type is normalized to the Oracle database time zone. Datetime value is then stored to millisecond precision.	Oracle client session time zone context of the DATETIME data type is normalized to the Oracle database time zone. Datetime value is then stored to millisecond precision.
DATETIME	TIMESTAMP-TZ	Straight copy of the datetime portion. Time zone set to the OpenEdge client session time zone context.	Straight copy of the datetime portion. Time zone set to the Oracle session time zone context (defined by the application or through -Dsrv switch).
DATETIME-TZ	DATE	Straight copy of the datetime portion of OpenEdge client session time zone context of the DATETIME-TZ data type up to second precision. Millisecond portion is dropped. Time zone portion is dropped.	Straight copy of the datetime portion of Oracle session time zone context of the DATETIME-TZ data type up to second precision. Millisecond portion is dropped. Time zone portion is dropped.

Source Data Type (OpenEdge)	Destination Data Type (Oracle)	Default Conversion	Non-default Conversion
DATETIME-TZ	TIMESTAMP	Straight copy of the datetime portion of OpenEdge client session time zone context of the DATETIME-TZ data type up to millisecond precision. Time zone portion is dropped.	Straight copy of the datetime portion of Oracle session time zone context of the DATETIME-TZ data type up to millisecond precision. Time zone portion is dropped.
DATETIME-TZ	TIMESTAMP-LTZ	OpenEdge client session time zone context of the DATETIME-TZ data type is normalized to Oracle database time zone. The datetime value is then stored to millisecond precision.	Oracle session time zone context of the DATETIME-TZ data type is normalized to Oracle database time zone. The Datetime value is then stored to millisecond precision.
DATETIME-TZ	TIMESTAMP-TZ	Straight copy with millisecond precision.	Straight copy with millisecond precision.

Converting Oracle TIMESTAMP types to ABL DATETIME types

The following table provides details on converting Oracle TIMESTAMP data types to similar OpenEdge data types.

Table 14: Converting Oracle TIMESTAMP types to ABL DATETIME types

Source Data Type (Oracle)	Destination Data Type (OpenEdge)	Default Conversion	Non-default Conversion
DATE	DATE	Straight copy of date portion. Time portion is dropped.	Straight copy of date portion. Time portion is dropped.
TIMESTAMP	DATE	Straight copy of date portion. Time portion is dropped.	Straight copy of date portion. Time portion is dropped.
TIMESTAMP-LTZ	DATE	Oracle converts TIMESTAMP-LTZ to Oracle session time zone (UTC). DataServer converts UTC to OpenEdge client time zone. Date portion of converted timestamp is copied. Time and time zone portions are dropped.	Oracle converts TIMESTAMP-LTZ to Oracle session time defined by the application to the Oracle client OCI driver. Date portion in session time is copied. Time and time zone portions are dropped.

Source Data Type (Oracle)	Destination Data Type (OpenEdge)	Default Conversion	Non-default Conversion
TIMESTAMP-TZ	DATE	Receives Oracle TIMESTAMP-TZ with time zone context from the database. DataServer converts to OpenEdge client time zone context. Date portion of converted timestamp is copied. Time and time zone portions are dropped.	Receives Oracle TIMESTAMP-TZ with timezone from the database. DataServer converts Oracle TIMESTAMP-TZ to Oracle session time (defined by the application to the Oracle client OCI driver). Date portion of converted timestamp is copied. Time and time zone portions are dropped.
DATE	DATETIME	Straight copy of datetime portion up to second level accuracy. Millisecond portion will be set to ‘000'.	Straight copy of datetime portion up to second level accuracy. Millisecond portion will be set to ‘000'.
TIMESTAMP	DATETIME	Straight copy with millisecond precision.	Straight copy with millisecond precision.
TIMESTAMP-LTZ	DATETIME	Oracle converts TIMESTAMP-LTZ to Oracle session time zone (UTC). DataServer converts UTC to OpenEdge local time zone context accuracy. Timestamp in client time is copied with millisecond accuracy. Time zone portion is dropped.	Oracle converts TIMESTAMP-LTZ to Oracle session time context defined by the application to the Oracle client OCI driver. Timestamp in session time is copied with millisecond accuracy. Timezone will be dropped.
TIMESTAMP-TZ	DATETIME	Receives Oracle TIMESTAMP-TZ with stored time zone context. DataServer converts to OpenEdge client time zone context. Timestamp portion in client time is copied with millisecond accuracy. Timezone portion is dropped.	Receives Oracle TIMESTAMP-TZ with stored time zone context. DataServer converts to Oracle session time defined by the application to the Oracle client OCI driver. Timestamp portion in session time is copied with millisecond accuracy. Time zone portion is dropped.
DATE	DATETIME-TZ	Straight copy of the date and time portions of the Oracle DATE type into the DATETIME-TZ. Millisecond portion will be set to '000'. Time zone will be set to OpenEdge client time zone context.	Straight copy of the date and time portions of the Oracle DATE type into the DATETIME-TZ. Millisecond portion will be set to '000'. Time zone will be set to Oracle session time zone context defined by the application to the Oracle client OCI driver.

Source Data Type (Oracle)	Destination Data Type (OpenEdge)	Default Conversion	Non-default Conversion
TIMESTAMP	DATETIME-TZ	Straight copy of the Oracle TIMESTAMP into the DATETIME portion of the DATETIME-TZ with millisecond accuracy. Time zone is set to OpenEdge client time zone context.	Straight copy of the Oracle TIMESTAMP into the DATETIME portion of the DATETIME-TZ with millisecond accuracy. Time zone is set to Oracle session time zone context defined by the application to the Oracle client OCI driver.
TIMESTAMP-LTZ	DATETIME-TZ	Oracle converts TIMESTAMP-LTZ to Oracle session timezone (UTC). DataServers then converts UTC to OpenEdge client timezone context with millisecond accuracy and stores the client time zone context.	Oracle converts TIMESSTAMP-LTZ to Oracle session timezone defined by application to the Oracle client OCI driver. Timestamp with its timezone context in session time is passed directly into the DATETIME-TZ value with millisecond accuracy and stores the session time zone context.
TIMESTAMP-TZ	DATETIME-TZ	Straight copy with millisecond accuracy.	Straight copy with millisecond accuracy.

Working with Oracle 8 and earlier releases

Oracle 8 and earlier releases offer a DATE data type that provides date and time components but does not offer subsecond precision. To ensure compatibility with Oracle 8 and earlier releases, DataServer for Oracle by default converts the DATE data type in a result set to an OpenEdge DATE column that stores the DATE portion and an OpenEdge INTEGER column that stores the time portion of the specific Oracle DATE. However, if your application does not require Oracle DATE data type mapping to OpenEdge DATE and INTEGER columns and you have since upgraded to Oracle 9i or later, you should consider using the Data Dictionary to explicitly map the Oracle DATE data type to the new OpenEdge DATETIME data type. If your Oracle data source has limited support for time zones, you may want to consider parsing OpenEdge DATETIME columns and storing those attributes, such as precision and time zone, that may not exist in the foreign datetime description, into associated companion columns to the foreign data type. See Pre-selection criteria for the schema pull on page 56 for more detail about changing the OpenEdge type mapping for Oracle DATE types.

Working with Oracle 9i and later releases

Oracle Release 9i and later provide support to three specific date time data types - TIMESTAMP, TIMESTAMP with TIMEZONE and TIMESTAMP with LOCAL TIMEZONE. All three data types are supported by DataServer for Oracle through the OCI driver. The values of all three data types are normalized to the NLS timezone by the OCI driver before it is received into or sent from the DataServer. Therefore, support of all Oracle datetime data types and their interaction with Daylight Savings is seamless at the OCI level.

Data type value retention

If your ABL application requires datetime accuracy more exact than millisecond precision and the Oracle data source provides it, you may need to retain `CHARACTER` data type mapping with the data parsing necessary to extract that precision from you application. For data sources that can handle OpenEdge datetimeattributes, conversion storage and retrieval is handled internally by the ABL.

Unsupported data types

The DataServer does not support the following data types:

- The Oracle `CFILE` data types
- Any OpenEdge SQL data type

Arrays

The DataServer supports arrays, which are also called field extents. When you create a schema image for an Oracle database, OpenEdge interprets a number of specially named Oracle columns of the same data type as a single OpenEdge field with the same number of extents. You name the Oracle columns *`column-name##1`*, *`column-name ##2`*, etc. A single field definition is created in the schema image for the field extent.

When you use the OpenEdge DB-to-Oracle migration utility, OpenEdge automatically modifies Oracle data definitions to support arrays. For example, if your OpenEdge database has an array named `MONTH` with 12 elements, the utility creates 12 columns named `MONTH##1`, `MONTH##2`, etc. in the Oracle table.

Unknown Value (?)

The DataServer supports the `Unknown value (?)`, which is represented by a question mark (?) in OpenEdge. In Oracle the `Unknown value (?)` is stored as a `NULL` value. This section describes how the DataServer handles the `Unknown value (?)`.

The following table summarizes how the OpenEdge `Unknown value (?)`, the Oracle unknown, and `NULL` values map to each other.

Table 15: `Unknown value (?)` and NULL values

ABL value	Oracle equivalent
`Unknown value (?)`	`NULL`
"" (Zero-length string)	" " (One space)

You can assign the Unknown value (?) to a field in an Oracle database by using the question mark (?) operator, which represents the Unknown value (?). For example, the following procedure assigns the OpenEdge Unknown value (?) to the address2 field of the customer table:

```
FIND FIRST customer.
customer.address2 = ?.
```

The Unknown value (?) sorts high in Oracle DataServer applications, as it does in OpenEdge applications. For an ascending index, the Unknown value (?), or values (?), always appear at the end of a sort, for a descending index, they appear at the beginning.

Note: Columns that have the NOT NULL attribute cannot contain the Unknown value (?).

Another difference in behavior between the Unknown value (?) and NULL is that Oracle does not return rows that contain NULL unless an operation involving NULL is specified. The following query will not return rows with NULL for an Oracle database. However, when you run it against an OpenEdge database, the query returns all the customers whose names are not Smith, including those whose names are unknown. For example:

```
FOR EACH customer WHERE customer.name <> "smith":
  DISPLAY customer.name.
END.
```

Although the example statement will not generate an illegal operator message, it will not necessarily generate the same results with an Oracle database that it will with an OpenEdge database.

Zero-length character strings

In addition to accepting the unknown operator, Oracle assumes that all zero-length character strings are unknown and stores them as NULL. In addition, a zero-length character string is represented as a single space in the Oracle database. This allows OpenEdge applications to distinguish between the Unknown value (?) and zero-length character strings.

When you use the Unknown value (?) in a WHERE clause with the DataServer, the Unknown value (?) satisfies only the equals (=) operator. Both of the following statements find the first customer record with the Unknown value (?) in the address2 field. Notice the space between the quotation marks in the first statement:

```
FIND FIRST customer WHERE customer.address2 = " ".
FIND FIRST customer WHERE customer.address2 = "".
```

Although "" and " " evaluate the same way in a WHERE clause, they have different results when you use them with the BEGINS function. For example, the following statement retrieves all customer names except those that have the Unknown value (?):

```
FOR EACH customer WHERE customer.name BEGINS "":
```

The following statement uses " " to retrieve only those names that begin with a space:

```
FOR EACH customer WHERE customer.name BEGINS " ":
```

Because `Unknown value (?)`, or values, satisfy only the equals condition, the following code does not retrieve customers with an `Unknown value (?)` in the `address2` field:

```
FOR EACH customer WHERE customer.address2 <> "foo":
  DISPLAY customer.name.
END.
```

The following statement is not meaningful to Oracle. It generates the error, "Illegal operator for `Unknown value (?)` or zero-length character string":

```
FIND FIRST customer WHERE customer.address2 < ?.
```

This restriction has been relaxed for columns of the `DATE` data type, as shown in the following statement:

```
FIND FIRST order WHERE order.orderdate < ?.
```

Record creation

This section discusses record creation topics.

- Comparing record creation behavior on page 64
- Ensuring record are created in a predictable order on page 66

Comparing record creation behavior

Record creation behavior differs between an OpenEdge database and an Oracle database accessed through the DataServer. The following code fragments provide examples of the different behavior.

If you have a table called `customer` with a field called `custnum` defined as an indexed field, and you write the following procedure:

```
CREATE customer.
ASSIGN
  customer.name    = "SMITH"
  customer.custnum = 10
  customer.address = "1 Main St".
```

ABL — Does not create the record right away (at the CREATE statement). It writes it to the database at the end of the record scope or when the index information is supplied. In this example, ABL writes the record after the statement:

```
customer.custnum = 10
```

OpenEdge DataServer for Oracle — Writes the record to Oracle later than it is written to an OpenEdge database. The DataServer writes the record at the end of the record scope.

Another example of the differences between ABL and the DataServer occurs if you write a procedure similar to the following:

```
DEFINE BUFFER bfCustomer FOR customer.

CREATE customer.
customer.custnum = 111.

FIND bfCustomer WHERE bfCustomer.custnum = 111.
DISPLAY bfCustomer.
```

ABL — Displays customer 111.

OpenEdge DataServer for Oracle — Fails to find customer 111 because it has not yet written the record that contains customer 111 to the database.

To get the correct response from the DataServer, use the following program:

```
DEFINE BUFFER bfCustomer FOR customer.

CREATE customer.
customer.custnum = 111.
VALIDATE customer. /* or RELEASE customer. */

FIND bfCustomer WHERE bfCustomer.custnum = 111.
DISPLAY bfCustomer.
```

In this example, however, using a VALIDATE statement causes the DataServer to write the record to the database. The VALIDATE statement causes the DataServer to write the record that contains customer 111 to the database before the FIND statement occurs. The RELEASE statement also causes the DataServer to write the record to the database; however the RELEASE statement clears the contents of buffers.

Note: Using the ROWID function might cause a newly created record to be written earlier to an Oracle database than to an OpenEdge database. If you do not assign values to all mandatory fields for a record, the ROWID function will fail.

This difference in behavior might also be apparent when you open a query. Newly created records might not appear in the results set of queries that you opened before you created the records. Reopen the query to access the new records.

Ensuring record are created in a predictable order

Records created within a transaction are not necessarily created in a predictable order. For example, when two records from two tables are created within the same transaction, you might expect that there is a sequential order to the record creation process; the initial record would be created first followed by the second and so forth. However, it is possible that the second record is created before the first record. The unpredictable nature of the record creation process can therefore affect the integrity of your data should the data require a predictable order to the write process on the Oracle data source.

Progress Software Corporation recommends that you use one of the following ABL language elements to force the order in which ABL records are written to the Oracle data source:

- Release statement
- Validate statement
- RECID function

For example, the following code example shows two transactions: master and detail. In the initial sample code pair, it is possible that the detail code could be created before the master code:

```
CREATE master.
ASSIGN master.name = "transaction".
```

```
CREATE detail.
ASSIGN
  detail.name = "transaction"
  detail.id   = 10.
```

The following sample code pair shows the identical transactions as those previously presented. However, note the addition of the VALIDATE phrase to the master record to ensure that the master record is created before the detail record:

```
CREATE master.
ASSIGN master.name = "transaction".
VALIDATE master.
```

```
CREATE detail.
ASSIGN
  detail.name = "transaction"
  detail.id   = 10.
```

Record locking

OpenEdge applications rely on the Oracle RDBMS to handle all record locking for the target Oracle database. OpenEdge locks do not apply to your Oracle database. The following table compares OpenEdge locks to their Oracle equivalents.

Table 16: ABL and Oracle locks

OpenEdge lock	Oracle lock	Transaction processing option lock[9]
NO-LOCK	None	None
SHARE-LOCK	None	None
EXCLUSIVE-LOCK	SHARE UPDATE(row-level)	SHARE UPDATE(row-level)
EXCLUSIVE-LOCK . . . UPDATE	SHARE UPDATE (row-level) . . EXCLUSIVE (table-level)	SHARE UPDATE(row-level) ...ROW EXCLUSIVE (row-level)

In applications that use the DataServer, locks occur as a result of ABL statements that the DataServer translates into SQL statements and sends to the Oracle RDBMS.

The following table shows examples of ABL statements, the SQL statements they generate, and the resulting Oracle locks in an Oracle database. The examples assume the default is SHARE-LOCK. The notes that follow the table help explain the locking behavior.

[9] The Oracle Transaction Processing option provides a low-level locking manager.

Table 17: Oracle locking

ABL statement	SQL statements generated	Oracle locks
`FIND customer.`	`SELECT . . . FROM customer;`	`None`
`FIND customer EXCLUSIVE-LOCK.`	`SELECT . . . FROM customer FOR UPDATE;`	`Share Update`

`FIND customer. . . . UPDATE customer.`[10]	`SELECT . . . FROM customer`
	`SELECT . . . FROM customer FOR UPDATE;`
	`Compares records`
	`UPDATE customer . . . ;`[10]
		`None`
		`Share Update`
		`{Row} Exclusive`[11]

ABL and Oracle release locks at different points in a transaction. When an application issues an `UPDATE`, ABL releases the lock once the new data is input. Oracle does not release the lock until the application issues a `COMMIT` or `ROLLBACK`. ABL allows you to hold a lock outside of a transaction or beyond a transaction's scope, but Oracle always releases all locks at the end of a transaction.

The Oracle database supports "deferred constraints" which are enforced at a transaction boundary. If a deferred constraint violation occurs at a transaction boundary, the transaction is rolled back and the application is terminated.

See the Oracle documentation for details on Oracle locking. See *Develop ABL Applications* for details on how ABL transactions and locks work.

Monitoring locks

Since the DataServer relies only on Oracle locks, the OpenEdge PROMON utility cannot provide you with information on locks in your database applications. Instead, you must use the Oracle monitoring tools.

Transactions

The Oracle RDBMS handles transaction roll back and recovery, but ABL transaction scoping rules apply.

[11] If you use Oracle with the Transaction Processing option, the result is a Row Exclusive Lock. Without Transaction Processing, the result is a table-level Exclusive Lock.

[10] When ABL encounters an UPDATE statement that involves an Oracle database, it uses a FIND . . . EXCLUSIVE-LOCK statement to check whether the record referenced by the UPDATE statement is already locked.If the record in the buffer is locked, ABL starts the UPDATE. If not, it immediately issues an SQL SELECT . . . FOR UPDATE statement to determine whether the value in the buffer is the same as the value in the database. This statement also locks the record. If the values are different, ABL returns a run-time error. When the SELECT . . . FOR UPDATE statement completes successfully, the UPDATE starts.When ABL UPDATE completes, ABL generates an SQL UPDATE statement that performs the actual change to the Oracle database. For example, if you have to retrieve a record for a subsequent update, use the EXCLUSIVE-LOCK modifier with the FIND statement to avoid the second SELECT . . . FOR UPDATE operation.**NOTE:** The last ABL statement in the table is an example of a lock upgrade.

In ABL, transactions end at the end of the outermost block where an update takes place. When a transaction that updates an Oracle database ends successfully, ABL sends a `COMMIT` to the Oracle instance. If you interrupt the transaction, ABL sends a `ROLLBACK` to the Oracle instance.

The Oracle database supports "deferred constraints" which are enforced at a transaction boundary. If a deferred constraint violation occurs at a transaction boundary, the transaction is rolled back and the application is terminated.

See *Develop ABL Applications* for more information on how ABL handles transactions and error conditions.

Error handling

Attempting to add a duplicate record is a common data-entry error. For example, the user might try to create a record, such as a `customer` record, using a unique key that already exists in the database. If a `customer` record already exists where the `custnum` equals 1, and the user tries to add another `customer` with the same `custnum`, ABL generates an error. When this type of error occurs, ABL tries to resolve the error by working its way back through the procedure, looking at each block header until it finds the closest block that has the error property. It then undoes and retries the block. See *Develop ABL Applications* for more information about error handling.

Because the DataServer is accessing a non-OpenEdge database, it cannot detect duplicate key errors until the end of a transaction block. As a result, if the error occurs in a subtransaction, ABL cannot detect the error until the end of the entire transaction block, so it performs default error handling for the transaction block.

This code example illustrates ABL and DataServer error handling:

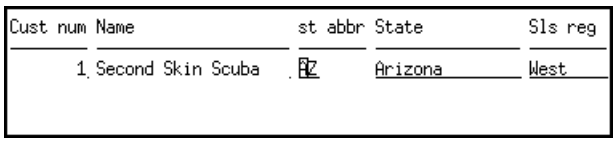
```
/* DataServer returns control here when a duplicate state key is
   encountered. */
rep-blk:
REPEAT:
  /* DataServer prompts here on duplicate state key. */
  PROMPT-FOR customer.custnum.
  FIND customer USING customer.custnum NO-ERROR.
  IF AVAILABLE customer THEN
    UPDATE customer.custnum customer.name.

  /* OpenEdge DB returns control here when a duplicate state key is
      encountered.*/
  do-blk:
  DO ON ERROR UNDO do-blk, RETRY do-blk:
    FIND state WHERE state.state = customer.state.
    DISPLAY state.

    /* OpenEdge DB prompts here on duplicate state key. */
    SET state.
  END.
END.
```

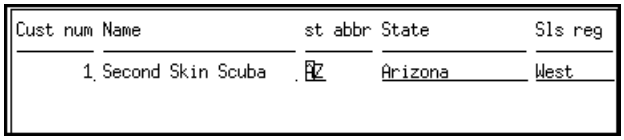
This procedure displays the screen shown below. The procedure prompts the user to enter data into the `custnum` field, and then into the `state` field. Suppose a user enters an existing `state` (for example, NH) while ABL is processing the `DO` block. With an OpenEdge database if a duplicate key entry occurs in the `DO` block, control returns to the `DO` block. After ABL displays a message that the record exists, it reprompts the user for the state abbreviation. For example:

Figure 8: OpenEdge database reprompts for the state abbreviation.



With the OpenEdge DataServer for Oracle, if a duplicate key entry occurs in the `DO` block, ABL returns control to the `REPEAT` block. The procedure reprompts the user to enter the customer number after the inner transaction completes. For example:

Figure 9: DataServer for Oracle reprompts for the customer



When two users simultaneously attempt to create records with duplicate keys, another difference in behavior occurs. ABL raises an error immediately, but Oracle raises an error after the first transaction commits, and only if the second transaction does not roll back. It is important to note that the second attempt to create a duplicate key will wait until the first user sends the transaction. Oracle does not notify the DataServer that it is waiting for the other user's transaction to end, so the DataServer cannot produce a message on the client indicating the lock wait situation.

To avoid this difference, change the scope of the transaction so that it completes more quickly or make the key nonunique and enforce uniqueness at the application level. Another technique is to use a `RELEASE` or `VALIDATE` statement when you check for the key's uniqueness.

Overflow checking

Overflow checking is a process by which the DataServer evaluates whether the value defined for a data type exceeds the data type's capacity. If the value is greater than the bit size defined for a data type, an error message occurs.

Note: Because in-memory operations on `INTEGER` and `INT64` data types are 64-bit, potential conflicts arise when data is exchanged between Oracle DataServers and OpenEdge clients of Release 10.1A and earlier. If a networked 10.1B DataServer produces an `INT64` data type that is sent to a pre-10.1B client, and the `INT64` data type is mapped to a 32-bit data type that exceeds 32-bits of data storage, the Oracle DataServer will detect an overflow condition and an error message appears on the client machine.

Cursors

ABL uses cursors to keep track of where it is in a file. A *cursor* is like a pointer that points to consecutive records in a file. For example, ABL uses cursors when it processes `FOR EACH` types of statements. ABL maintains cursor positioning across queries.

The DataServer supports this behavior for Oracle tables that have a mandatory unique index on an INTEGER column or that contain the PROGRESS_RECID column. (The OpenEdge DB-to-Oracle migration utility creates an indexed NUMBER column named PROGRESS_RECID with unique values for the rows in each Oracle table.)

Suppose that you are reading records from the customer file using the custnum index, and your "current" record is customer number 50. This means that ABL has a cursor positioned at custnum 50.

Static FIND cursor repositioning

With the DataServer, cursor repositioning will work with an Oracle database the same as it does with ABL, except when the Oracle Database Manager fails to find a record. In ABL, the cursor is located after the last record that was read. In Oracle, a failed search does not affect the cursor. For example, if a cursor is positioned at custnum 50 and a request to find customer "smith" in an Oracle table fails, the cursor remains at custnum 50. The same failed request from an ABL table places the cursor after the last customer read trying to find customer "smith."

The DataServer repositions only cursors used by FIND NEXT/PREV/CURRENT statements. Cursors are never repositioned for FOR EACH statements, new queries, or dynamic FINDs. When the DataServer instructs the Oracle RDBMS to perform a join (JOIN-BY-SQLDB), the cursor that the join uses does not reposition other cursors in use by the query. The cursor for DEFINE QUERY or dynamic FIND statements does not reposition cursors in other queries.

Stale cursors

A *stale cursor* is a cursor that points to information that is out of date in an Oracle database. When you execute SQL on a cursor, Oracle hides any changes that occur to the database after that point. It reconstructs the state of the database at the time the cursor was created by using before-image information it writes to rollback segments. There is a limited number of rollback segments so Oracle reuses them. A cursor whose rollback-segment information has been overwritten is stale, and if you attempt to fetch data on a stale cursor, the fetch fails.

An application is most likely to encounter a stale cursor if it has a long loop or a loop that might take a long time because it involves user interaction and simultaneously allows updates to the same table the loop accesses. FIND NEXT and PREV statements are also likely candidates for stale cursors.

To avoid stale cursors, you can tune your database environment or adjust your application. For example, you can increase the number and size of rollback segments for the Oracle database. To adjust your application, consider putting the record identifiers for the table that must be updated into a temp table. Standard cursors are somewhat less likely to become stale than lookahead cursors. Although you might lose some performance, using the QUERY-TUNING NO-LOOKAHEAD option might help you avoid stale cursors. See Query tuning on page 128 for instructions.

Allocating cursors

Oracle allocates a maximum number of cursors that your application can use based on system configuration, the version of Oracle, and the database-configuration parameters. An application can run out of available cursors. If this occurs, you can specify that Oracle should allow more cursors, or you can control how many cursors the DataServer uses with the Index Cursor (-c) connection parameter. See Index cursors on page 197 for more information.

Issuing the CLOSE QUERY statement after opening a query is another technique that helps you minimize the number of cursors your application requires.

Issuing a close inactive cursor statement, RUN STORED-PROC ds-close-cursor, helps you close the inactive cursors that are either no longer (or minimally) required for future application use. For more information on closing inactive cursors, see RUN STORED-PROC statement with special internal option on page 93 and Closing inactive cursors on page 128.

ABL issues

The following sections present information about using various ABL statements in DataServer applications. They discuss how the DataServer supports the RECID function, the DEFINE BROWSE statement, the COMPILE statement, and the FIND PREV/LAST statements, among others.

Note: The ROWID and RECID functions might cause a newly created record to be written earlier to your Oracle database than to an OpenEdge database. If you do not assign values to all fields that are defined as mandatory in Oracle for a record, the ROWID and RECID functions will fail.

RECID function

For backward compatibility, the DataServer supports the RECID function for Oracle tables that have a PROGRESS_RECID column. The OpenEdge DB-to-Oracle conversion utility creates an indexed NUMBER column, called PROGRESS_RECID, with unique values for each row. You can also add this column to Oracle tables manually. See Manually adding PROGRESS_RECID to Oracle on page 256 for instructions.

You can make an existing application that includes RECID behave more consistently across data sources by replacing RECID with ROWID. See the for more information.

ROWID function

The ROWID data type provides a record identifier that is compatible across OpenEdge and Oracle databases. Applications that use the ROWID function behave the same way whether they access ABL or Oracle records.

Note: The default external representation of ROWID enables the Oracle Dataserver to use a 64-bit integer value for ROWID. Using an integer value to define ROWID is not required, but rather an option to efficiently represent ROWID as a numeric binary value. The Oracle DataServer supports the NUBER data type to emulate ROWID expansion.

By default, the DataServer designates a column to support the ROWID function. It evaluates the indexes available for a table and selects one in the following order:

1. PROGRESS_RECID column
2. Unique index on a single, mandatory, NUMBER column with precision < 10 or undefined and scale 0 or undefined
3. Native ROWID

Oracle might not provide a native ROWID for views that contain aggregates or perform joins. For one of these views to support the ROWID function, use the Data Dictionary to select a NUMBER column, create a unique index based on it, and designate it as the ROWID.

The Data Dictionary allows you to select a different column to support the ROWID function or to select the native ROWID. See Defining the ROWID on page 231 for instructions about changing how a table supports ROWID.

The native ROWID typically provides the fastest access to a record. However, the native ROWID does not support ABL's FIND PREV/LAST statement or cursor repositioning. FIND FIRST/NEXT statements for tables that use the native ROWID as a row identifier have unpredictable results. Qualify the FIND FIRST statement with the USE-INDEX option, as in the following example, to get consistent results across various data sources, as shown:

```
FIND FIRST customer USE-INDEX custnum.
```

ROWID provides the same functionality as the RECID function, but ROWID is more consistent across data sources. Replace the RECID function with ROWID in existing applications.

Follow these guidelines when using ROWID in applications that you want to deploy across several databases:

- Do not try to get the ROWID value before the user assigns values to the unique keys of that record. Some DataServers use the unique key to generate a ROWID value.
- Refresh the ROWID value if a value of a unique key might have changed.
- Refresh the ROWID value after you undo a DELETE. The ROWID value might be different after the record is re-created.
- ROWID values are stable for a transaction. Do not rely on them being the same across transactions or sessions.

Note: Reposition functions such as REPOSITION-BACKWARDS and REPOSITION-TO-ROW typically use ROWID to identify records. Functions of this type require integer expressions, which can be either INTEGER or INT64.

See the ROWID function entry in *ABL Reference* for more information and examples.

Progress_Recid change

In OpenEdge Release 10.1B, the default PROGRESS_RECID recognizes an INT64 value for an Oracle DataServer. The unique key value in the PROGRESS_RECID is determined by default from a 64-bit value. (For prior releases, the value was derived, by default, from the 32-bit INTEGER.)

DEFINE BROWSE statement

The DEFINE BROWSE statement relies on a unique record identifier for forward and backward scrolling. If your Oracle table does not support the ABL ROWID function (either through a PROGRESS_RECID column or an indexed NUMBER column with unique values), you can write the following code that explicitly requests the scrolling behavior that the ABL browse has by default:

```
/* Define a query named qCustomer for customer scrolling. */
DEFINE VARIABLE iRow as INTEGER NO-UNDO.

DEFINE QUERY qCustomer FOR customer FIELDS(address custnum name) SCROLLING.
DEFINE BROWSE b QUERY qCustomer DISPLAY custnum name address WITH 10 DOWN.
DEFINE BUTTON upd.

OPEN QUERY qCustomer FOR EACH customer.
ENABLE upd b WITH FRAME x.
ON CHOOSE OF upd DO:
  iRow = CURRENT-RESULT-ROW("qCustomer").
  GET PREV qCustomer.
  GET NEXT qCustomer EXCLUSIVE-LOCK.
  IF CURRENT-RESULT-ROW("qCustomer") = iRow THEN
    UPDATE customer.address WITH FRAME z VIEW-AS DIALOG-BOX.
    /* else, indicate that an error occurred:  the record was deleted in the
       meantime. */
  DISPLAY customer.address WITH BROWSE b.
END.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Field lists

The DataServer fully supports field lists in queries (DEFINE QUERY, FOR EACH, PRESELECT, and SQL SELECT statements). For example, the following statement returns the same results for OpenEdge and Oracle databases:

```
DEFINE QUERY myquery FOR customer FIELDS (custnum name) SCROLLING.
```

Include the SCROLLING option to enable record prefetch. You must include the NO-LOCK option when you open queries with field lists, as in the following example:

```
OPEN QUERY myquery NO-LOCK.
```

Similarly, you must include the NO-LOCK option in FOR EACH statements that include field lists, as in the following example:

```
FOR EACH customer FIELDS (custnum name) NO-LOCK:
```

The NO-LOCK option ensures that the DataServer does not have to refetch rows, which might slow performance. In addition, combining lookahead cursors and field lists especially improves a query's performance. See Sample Queries on page 261 for a comparison of lookahead and standard cursors with field lists.

Use field lists to retrieve only those fields that your application requires. (For performance reasons, the DataServer retrieves the first index field even when you do not include it in the field list. In cases when the DataServer can predict that a query requires a refetch, it retrieves the entire record.) The DataServer allocates memory based on the maximum size specified for a field in a record. Omitting larger fields or unnecessary fields from a query enhances performance.

When you specify a field that has an extent, the query returns the entire array. You can specify an Oracle `LONG` column in a field list. However, when the query selects a `LONG` column that has more than 255 bytes of data, the DataServer refetches the column using a row identifier (`PROGRESS_RECID` column, unique `NUMBER` index, or native `ROWID`). In the case of views with aggregates or joins where there is no row identifier, the query stops and you receive an error that the record was truncated.

When the DataServer processes a query with a field list, it caches the fields that are part of the field list and any other fields that the query specified on the client, which you can then access without making another call to the Oracle RDBMS. For example, the DataServer fetches the name and the zip field to process the following query:

```
FOR EACH customer FIELDS(name) WHERE customer.postalcode = 01730 NO-LOCK:
```

If you specify a field list in a join, you might have to adjust the cache size for lookahead cursors, either with the `CACHE-SIZE` option in a `QUERY-TUNING` phrase, or at the session level with the `-Dsrv qt_cache_size` startup parameter.

Any performance gained through field lists is lost if you use nonlookahead cursors or `SHARE-LOCK`.

See the Record Phrase entry in *ABL Reference* for more information on the `FIELDS` option.

FIND PREV/LAST statements

OpenEdge applications that use the `FIND PREV` or `FIND LAST` statements work on Oracle tables in a manner consistent with ABL.

To support these statements, an Oracle table must include support for the `PROGRESS_RECID` function (either through a `PROGRESS_RECID` column or an indexed `NUMBER` column with unique values). See the RECID function on page 72for more information.

Compiling ABL procedures

The `COMPILE` statement lets you compile ABL procedures and save the compilations to disk. This speeds up your application since ABL does not have to recompile it every time you want to run a procedure.

To compile procedures that access an Oracle database, start up the OpenEdge procedure editor and connect to the schema holder for your target Oracle database using the schema holder's logical database name. Then use the `COMPILE` statement. If you change the name of the schema holder after compiling a procedure, you must connect to the renamed schema holder and recompile the procedure. You do not have to connect to the target Oracle database to compile a procedure.

For more information, see the "COMPILE Statement" reference entry in *ABL Reference*.

r-code

r-code is generated when an ABL procedure is compiled. The compiled r-code is portable among machines. For example, r-code that you compile on a Sun machine can run on any other UNIX machine.

r-code is **not** portable among windowing systems; that is, r-code compiled for a character application will not run under Windows and r-code compiled for Windows will not run under a character application.

The r-code is also not portable among database management systems. For example, code that you have compiled for an Oracle database will not run with an OpenEdge database.

r-code is also **not** portable among different databases. ABL generates calls that are specific to a database. For example, code that you compile for a database named Sports will not run with a database named **mysports**.

The size of r-code can grow when you compile procedures against an Oracle database as compared with compiling against an OpenEdge database. The r-code for a DataServer application contains as text the portions of SQL statements that the DataServer passes to Oracle. The r-code descriptors for column references contain more attribute information than corresponding OpenEdge descriptors.

Prior to OpenEdge Version 11, r-code stored the physical position of a database column. r-code was not compatible with the server schema if the position of a column changed on the server. In Version 11, r-code stores a logical identifier for the physical position of a database column. Therefore, in OpenEdge 11, column positions in the server table can be rearranged without the need to recompile. However, introducing a new column or deleting an existing column from an existing record structure always requires the r-code to be recompiled against the new schema layout.

The r-code is dependent only on a logical identification of the column, and not the physical position of the column. The logical identification, in the OpenEdge database, takes place before migrating to a foreign data source, and the logical identification does not change once the column is defined in the OpenEdge table. When the columns are rearranged on the server, the code need not be recompiled, but the foreign schema must be pulled back into the schema image. This is to ensure that the new physical position of a column is matched in the schema to the logical identifier that is also known to the r-code.

You can only run r-code or load data (.d file) while connected to a schema image that is derived from the same database that was migrated and connected to the server when the code was first compiled and/or the data was first dumped. This ensures that the same logical column identifier is preserved after migration and/or after any subsequent schema pulls.

Note: When the server table is re-pulled into the same schema image (after the columns are rearranged on the server), the logical column identifier is re-matched to the physical column position so that the r-code can find again the column in a run-time application.

See *Manage ABL Applications* for more details on r-code portability.

Unsupported ABL statements

The DataServer supplies you with the complete functionality of ABL when accessing Oracle databases. Nearly all ABL language elements (statements, functions, etc.) and Data Dictionary features work the same whether your application accesses an Oracle database through the DataServer or an OpenEdge database. You can use the `DBRESTRICTIONS` function to find out which ABL features that your non-OpenEdge databases, or specific tables in those databases, do not support.

For the DataServer for Oracle, `DBRESTRICTIONS` can return `RECID`, `PREV`, `LAST`, `SETUSERID`, `SET-CURRENT-VALUE`. If you connected to the logical database in read-only mode, the function returns `READ-ONLY` for the schema holder connection, not for the connection to the Oracle database. You can specify `-RO` in the connection to Oracle; in that case, the function returns `READ-ONLY` for the Oracle database as well. See the "DBRESTRICTIONS function" reference entry in *ABL Reference* for information on syntax.

The following table lists some ABL statements and functions that behave differently between the OpenEdge DataServer for Oracle and a native OpenEdge database.

Table 18: ABL and Oracle differences

Feature	Description
BEGINS functionAbbreviated IndexUSING option	When you use these ABL elements to access data in an Oracle database, you will have different results than you would expect from ABL in the following case. If you have a customer named SI and one named SIM and you issue this FIND statement:FIND customer WHERE name BEGINS "SI"ABL returns the customer named SI; with the DataServer, the find fails because it is considered ambiguous. You receive the same results when you use an abbreviated index or the USING option in your query.
BEGINS operatorMATCHES function	To resolve BEGINS or MATCHES comparisons, Oracle does not use an index as ABL does. Instead, it might do a complete table scan. The table scan typically occurs when the Oracle DBMS does not know the value of the pattern when the SQL is compiled. See Query tuning on page 128 for information on using the NO-BIND option to handle this situation.Do not use the MATCHES or BEGINS function with a pattern that is not an expression, but is stored in the Oracle database. Although theoretically possible with ABL, using this kind of criteria results in poor performance with an OpenEdge database.
CONTAINS operator	This option relates to word-indexing, which the DataServer does not support. It returns a compilation error.
COUNT-OF function	The DataServer does not support this function.
CREATE statement	Records you create after a cursor was opened might be invisible to that cursor. Oracle maintains a view of a database's state at the time when the user opens a cursor. Changes you make to a database after opening a cursor might not be visible.
CURRENT-VALUE function	You can use CURRENT-VALUE only after you have successfully called the NEXT-VALUE function.
CURRENT-VALUE statement	The OpenEdge DataServer for Oracle does not support setting a sequence generator's current value.
DBTASKID function	The OpenEdge DataServer for Oracle does not support this function.
FIND statementsQUERY option	To reduce the number of records Oracle includes in the results set, you should qualify your FIND statements and queries with a WHERE clause. This achieves a performance rate that is closer to OpenEdge performance. For better performance, use the DEFINE QUERY statement instead of FIND.To control the order of the results, include the USE-INDEX or BY options in your queries.

Feature	Description
MATCHES function	If you want to use the MATCHES function for a string containing double-byte characters, you must fill out the character expression with periods (.). For example, where J is a double-byte character, the following statement does not find a match:FIND customer WHERE name BEGINS "SI"Add periods to search the entire field. For a field that is defined as 20 bytes long, to adjust the example, include seventeen periods after the single double-byte character, J.
OPEN QUERY statement	Newly created records might not appear in the results set of queries that you opened before you created the records. Reopen the query to access the new records.
INDEXED REPOSITION clause	Index reposition might cause new SQL to be executed thereby causing newly created records to appear.
SESSION:TIME-SOURCE handle	This system handle returns the Oracle DataServer's time information.
SETUSERID function	You cannot use this function to change the user ID and password of an Oracle login.
SHARE-LOCK option	You cannot use this option for a query with the FIELDS option.SHARE-LOCK is NO-LOCK for Oracle.
Time in the WHERE clause	ABL supports this option only if you have mapped Oracle DATE columns to the OpenEdge CHARACTER fields in the schema image.
USERID function	For a connection to an Oracle database, the USERID function returns the value that you specified for the -U parameter. For example, if you specify -U bob/password, USERID returns bob/password.
LONG/CLOB in WHERE clause	Oracle does not support any field defined as LONG or CLOB in a WHERE clause. Usually, a table that has a column >4000 characters is defined as LONG or CLOB in an oracle database.

National Language Support

After the DataServer connects to Oracle, it issues the following statements to set National Language Support (NLS) parameters for the Oracle environment:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDD'
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,'
```

These settings might affect how stored procedures (including those using the `send-sql-statement` option) behave. Specifically, the default format that the `TO_DATE` function and the format of a date returned might be different. Always specify the format for the date as the second argument to the `TO_DATE` and `TO_CHAR` functions when dealing with date values.

Handling lock timeouts

If you are upgrading your Oracle DataServer Oracle DataServer to 10.2B08 or 11.x and if you face Lock Error, you can overcome the same by using PRGRS_NATIVE_LOCKWAIT switch.

The default behavior for handling a lock timeout condition in the OpenEdge DataServers, if the `PRGRS_NATIVE_LOCKWAIT` parameter is not set, is to return control immediately to the OpenEdge client. Therefore, the lock wait timeout at the data source is set to zero at the beginning of a client session when using the OpenEdge DataServer for Oracle. This is desirable behavior for clients that want immediate control over lock handling.

The client application can choose to handle the lock timeout directly using the NO-WAIT and keywords. Then, when a record cannot be accessed because it is locked by another user, the application can test for the server timeout condition by testing for TRUE returned from the LOCKED function. The application consumes the timeout condition in this case and is free to perform whatever action is deemed necessary.

If the client application does not specify NO-WAIT, then the application automatically loops back to the server in an internal wait mode and retries record access. It continues to do so until the Lock Timeout period set on the client (–lkwtmo parameter specified in seconds) is exceeded. If this wait period is exceeded without being able to successfully access the server resource, the process times out, the wait is canceled and the client raises a stop condition.

When NO-WAIT is unspecified, the client consistently returns to the server to retry access to the locked resource. If is also unspecified, then during the Lock Timeout period (-lkwtmo) a resource wait dialog box continues to be displayed to the user. It allows the user to select **Cancel** from the dialog to end the wait period. If the user does not cancel and the –lkwtmo period has not been exceeded, the client performs constant retries and multiple round-trips to the server. This constant re-cycling, especially during a period of high resource contention, can be normalized by setting a small timeout period on the server in which to handle lock conditions before returning timeouts to the client application.

The server wait period is set through the PRGRS_NATIVE_LOCKWAIT –Dsrv connection parameter. The disadvantage to setting this parameter to a non-zero value is that the client application is blocked for the timeout period set on the server. This may produce some amount of server-bound latency that should be considered when setting the number of seconds for this option. However, if the server is able to complete the resource request in the server timeout period, the resource is returned to the client immediately and the application is un-blocked. Therefore, the advantage of setting a non-zero server timeout is that the server is given the opportunity to resolve record access without further round trips from the client repeatedly requesting the same resource.

A non-zero value may be especially useful during periods of high contention and may increase the overall efficiency of the DataServer application. Progress recommends a nominal but non-zero setting for the number of seconds in most cases. Evaluate your average contention for resources in setting this value for your own application.

The PRGRS_NATIVE_LOCKWAIT –Dsrv option permits an application to set a maximum time threshold that the server will block the application waiting to fulfill a resource request that is locked. When the server waits for the resource longer than the PRGRS_NATIVE_LOCKWAIT number of seconds, control is returned to the client application which then handles the lock condition as described earlier. As the PRGRS_NATIVE_LOCKWAIT time is increased, the number of retries from the client within the –lkwtmo period is decreased (assuming NO-WAIT is unspecified).

The PRGRS_NATIVE_LOCKWAIT setting will affect all transactions for all connections to the foreign data source for a given application session. This includes read-only connections, stored-procedure connections, and transactions on the sequences connection.

`PRGRS_NATIVE_LOCKWAIT`

> "n" is the number of second(s), implemented to resolve Oracle locking issues. When the value of the parameter is set to ‘0’, SQL generated from ABL to Oracle uses ‘WAIT’ instead of ‘NOWAIT’.

See *Develop ABL Applications* for more information on how ABL handles transactions and error conditions.

RDBMS Stored Procedures

Relational database management system (RDBMS) stored procedures can be an efficient, productive approach to obtain data from a foreign data source. In general, using stored procedures can help reduce network traffic, promote better performance, and improve response time because the server-side processing of the stored procedure can be accomplished without communicating with the client; once this processing is completed, the server returns only the data results to the requesting client.

See RDBMS Stored Procedure Details on page 81 for a complete discussion of various RDBMS stored procedure and Send SQL statement techniques you can run against the Oracle DataServer.

3

RDBMS Stored Procedure Details

This chapter defines relational database management system (RDBMS) stored procedures and describes how to create and implement them in the OpenEdge environment. It discusses various RDBMS stored procedure and send-sql-statement option techniques you can employ. It also details how to execute an RDBMS stored procedure or send-sql-statement option against an Oracle Server, and load the result set directly into temp-tables.

For details, see the following topics:

- Overview
- RDBMS stored procedure basics
- RUN STORED-PROCEDURE details
- Data output and retrieval options
- Interfacing with RDBMS stored procedures
- Handling errors
- ROWID support

Overview

The DataServer allows you to call Oracle PL/SQL stored procedures, stored functions, and packages from within ABL procedures. *Stored procedures* and *stored functions* are self-contained groups of SQL statements that access the Oracle database. By executing a stored procedure or function, you can execute these statements or programs without having to enter their individual statements or code at each execution. Stored procedures and functions can return values or result codes. A *package* is a group of stored procedures, functions, and related program objects.

Note: This chapter uses the phrases RDBMS stored procedure and stored procedure interchangeably to refer to stored procedures and stored functions.

Enhancements to stored procedure and function techniques also allow you to extend your code in new ways and effectively employ more ABL features and functionality. For example, you can retrieve data from a foreign data source through a stored procedure and load this result set into a temp-table. Also, this technique is especially useful if you elect to further manage and manipulate your result set using ProDataSet capabilities.

Stored procedures can allow you to improve your DataServer's performance because they utilize the foreign data source's native capabilities. Stored procedures can be an efficient, productive approach to obtain data from a foreign data source or execute native business rules and procedures. In general, using stored procedures can help reduce network traffic, promote better performance, and improve response time because the server-side processing of the stored procedure can be accomplished without sustained communication with the client. Once this processing is completed, the server returns the data results to the requesting client.

The first task to effectively employ any stored procedure technique or approach is to determine what information you want to receive or process you want to run. Once you know your data requirements, you can proceed to define the stored procedure details.

Note: A complete discussion of creating and using native stored procedures is beyond the scope of this chapter. For this type of information, see your Oracle Server documentation.

Defining native stored procedures to ABL

The first time you run a stored procedure, the data-source management system creates an execution plan for it and stores the plan in the database. The next time you run the stored procedure, it runs the precompiled procedure, assuming it is still cached in the database. This makes access to the database quicker and more efficient than when you access it with new queries each time.

The DataServer allows you to use ABL to run Oracle stored procedures. All pre-defined stored procedures initiated on behalf of the Oracle are executed from within ABL, using the `RUN STORED-PROCEDURE` statement. You define specific ABL language elements to the `RUN STORED-PROCEDURE` statement to match the profile or signature of your native stored procedure. You can also provide additional ABL statements subsequent to the `RUN STORED-PROCEDURE` statement to handle result sets from stored procedures.

RDBMS stored procedure basics

In the OpenEdge environment, you can think of a stored procedure definition as having two basic, interrelated parts:

- **Execution controls to run and close a store procedure** — Comprise the information needed to execute a stored procedure request against the Oracle data source. At a minimum, all stored procedures discussed in this guide are assessable using the `RUN STORED-PROCEDURE` statement.
- **Language elements that enable access to specific data results** — Qualify the retrieved data, or *result set*, that the stored procedure's execution returns. Various keywords, phrases, statements, and syntax elements support different retrieval options for stored procedure output. This part of the stored procedure implementation reflects your analysis of your data needs; based on this analysis, you determine the additional syntax elements you need to define the output elements and data results you want retrieved.

The following table identifies and briefly introduces the elements that comprise a stored procedure definition. These elements are discussed later in this chapter.

Table 19: Stored procedure language elements

ABL	Description
`RUN STORED-PROCEDURE` statement	Executes a stored procedure
`PROC-HANDLE` phrase	Allows you to specify a handle to identify a stored procedure
`PROC-STATUS` phrase	Reads the return value
`LOAD-RESULT-INTO` phrase	Allows data from a result set that is returned for a foreign data source to be put into one or more temp-tables
`PARAM` phrase	Identifies run-time parameters to be passed to and/or from the stored procedure
`CLOSE STORED-PROCEDURE` statement	Enables the values to be retrieved from the output parameters that you defined for the stored procedure, finalizes result sets data processing, and tells OpenEdge that the stored procedure has ended
`NO-ERROR` phrase	Allows native database errors (that occur during stored procedure execution) to be evaluated and handled by the application. **Note:** This can also be achieved by a `CATCH` end block in the stored procedure. For more information on handling errors using the NO-ERROR option or a CATCH end block, see Error handling on page 69.

Note: You can substitute the abbreviations `RUN STORED-PROC` and `CLOSE STORED-PROC` for the full names `RUN STORED-PROCEDURE` and `CLOSE STORED-PROCEDURE`, respectively. The remainder of this guide generally uses the abbreviated form.

See RUN STORED-PROCEDURE details on page 86 for more details about the reference entries presented in the table above.

As noted in the table above, you can pass data types in the RUN STORED-PROCEDURE statement using the PARAM phrase. The following table lists issues that occur when you pass certain data types as parameters.

Table 20: Argument data types for stored procedures

OpenEdge	Oracle data source
DECIMAL INTEGER INT64	The DataServer converts each if these data types in the schema image to the equivalent OpenEdge data type as follows: • DECIMAL = NUMBER with precision and scale • INTEGER = CURSOR • INT64 = NUMBER with precision To preserve the scale and precision of these data types, you must manually update the information in the schema image for these parameters. Use the Data Dictionary to update the data type and format information in the Field Property Sheet for the parameter.
VARCHAR2	In Oracle, VARCHAR2 parameters cannot be greater than 4000 characters. If the VARCHAR2 parameter exceeds this limit, it causes an error. **Note:** If you use a parameter that is larger than 255 characters, you need to change the _for-maxsize value for this parameter.
DATE DATETIME DATETIME-TZ	You can specify a DATE, DATETIME or DATETIME-TZ data type as a parameter when using RUN STORED-PROCEDURE. You can also specify a DATETIME or DATETIME-TZ data type in a temp table used to receive results from a stored procedure using the LOAD-RESULTS-INTO phrase.
MEMPTR	You can specify a MEMPTR data type in a Paramphrase as an INPUT or an OUTPUT parameter to receive results from a corresponding BLOB data type parameter to or from a RUN STORED-PROCEDURE statement.
LONGCHAR	You can specify a LONGCHAR data type in a Paramphrase as an INPUT or an OUTPUT parameter to receive results from a corresponding CLOB data type parameter to or from a RUN STORED-PROCEDURE statement.

Note these stored procedure points:

- Input and output parameters are displayed as fields.
- If you are running several stored procedures, run them serially and process all the results from one stored procedure and close the procedure before you run a second one. By default, the DataServer allows one active request for running a stored procedure. It is not necessary to specify the PROC-HANDLE phrase when procedures are run serially.
- Stored procedures implemented through an Oracle DataServer already implicitly supports a large value (that is, greater than 32 bits) because the Oracle NUMBER data type supports 64-bit binary values.
- When you run stored procedures concurrently, the DataServer uses one connection to the data source per procedure. If different stored procedures attempt to update the same record from a single client's requests, the connections could block each other or a deadlock might occur.

Note:

You must define a PROC-HANDLE phrase for each stored procedure phrase that is simultaneously active. This technique provides a CLOSE STORED-PROC statement that can identify the targeted open procedure and close it.

In contrast, since a stored procedure executed with the LOAD-RESULT-INTO phrase implicitly closes the procedure once the execution ends and the data retrieved is placed into temp tables, it essentially runs serially and has no use for a PROC-HANDLE.

- When you create or update your schema image, the stored procedures appear in the list of accessible objects along with tables, view, and sequences. OpenEdge allows you to run the stored procedures that you create in data sources using the procedure definitions in your schema image. See your Oracle documentation for complete information about creating and using stored procedures.
- If ABL that executes a stored procedure is already within a transaction block, the stored procedure becomes an extension of that transaction and will not commit to the database until the ABL transaction is completed. However, because the stored procedure does not execute as part of an ABL client process, it cannot be rolled back by ABL.
- The DataServer cannot roll back sub-transactions in the stored-procedure context since it has no control over what the stored procedure executes.
- If you pass a DATE data type as an input parameter and use it in an equality test, the test might fail. In this case, use the trunc function in the stored procedure to isolate parts of the date structure for which you might want to test. For example:

```
procedure x_date (indate in date, outdate out date)
  as begin
  select date_terminate into outdate from datetbl
     where trunc(hire_date) = trunc (indate);
  end;
```

- For backward compatibility, Oracle DATE columns are pulled as OpenEdge DATE and OpenEdge INTEGER into the schema holder by default. If you create a temp-table using the LIKE phrase with a table in the schema holder that has mapped the Oracle DATE to OpenEdge DATE and INTEGER fields, the temp table definition will include both columns. If you use the LOAD-RESULTS_INTO phrase to receive the results set from a SEND-SQL-STATEMENT stored procedure or a non-ABL stored procedure, then the PL/SQL statement in the stored procedure that produces the resultant records should be constructed in such a way as to select the Oracle DATE into the two separate columns of the date and time portion that will match the temp table columns. Not mapping the temp table properly will result in a "Number of fields" mismatch error. Oracle functions TRUNC and EXTRACT can be used to extract the date and time portions from the Oracle Date into separate columns for the results buffer.

The following example loads the results of an Oracle DataServer schema definition created with the LIKE phrase into a temp-table using SEND-SQL-STATEMENT. Oracle table **mydate** with two columns, **dt_num** and **dt_date** is utilized in the schema holder to create a temp-table where stored-procedure results are loaded. The SQL is modified to produce the expected resultant.

```
DEFINE VARIABLE tth AS HANDLE NO-UNDO.
DEFINE TEMP-TABLE tt NO-UNDO LIKE mydate.
DEF VAR timeinsec AS CHAR NO-UNDO.

timeinsec = "EXTRACT(hour from to_timestamp(to_char(dt_date),
'yyyymmddhh24miss'))*3600 + EXTRACT (minute from
to_timestamp(to_char(dt_date), 'yyyymmddhh24miss'))*60".

DEF VAR sql-string AS CHAR NO-UNDO.

sql-string = "select dt_num, TRUNC(dt_date)," + timeinsec + " from
mydate".
tth = TEMP-TABLE tt:HANDLE.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO tth
NO-ERROR(sql-string).
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = tth.

FOR EACH tt.
DISP tt.
END.
```

The RDBMS Stored Procedure Details on page 81 presents more details about the use of the RUN STORED-PROC statement.

RUN STORED-PROCEDURE details

This section provides:

- A complete, detailed reference for all the syntax expressions and options you can use to define the RUN STORED-PROCEDURE and the associated language elements that comprise a stored procedure definition.
- Syntax to use the RUN STORED-PROCEDURE statementwith and without the LOAD-RESULT-INTO phrase.
- Detailed information about using the RUN STORED-PROCEDURE statement with the send-sql-statement option. Examples to use the RUN STORED-PROCEDURE statement with and without the LOAD-RESULT-INTO phrase are provided.

Syntax reference for RUN STORED-PROCEDURE

The following syntax shows the RUN STORED-PROC statement and all options that can be used with the statement. However, keep in mind that not all options shown can be used simultaneously.

```
RUN STORED-PROCEDURE procedure-name

[ [LOAD-RESULT-INTO <handle> [<int> = PROC-STATUS] ] |
     [<int> = PROC-HANDLE] ]
     [ NO-ERROR ]     [ ( [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name
=] expression, . . .
        [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression ) ]
```

Note: In the RUN STORED-PROCEDURE syntax, note that the <*handle*> is either a pure handle or a handle that points to a temp-table. For more information about using the temp-table handle with the LOAD-RESULT-INTO phrase, see CLOSE STORED-PROCEDURE statement on page 89.

```
CLOSE STORED-PROCEDURE procedure-name
     [integer-field = PROC-STATUS ]
     [ WHERE PROC-HANDLE = integer-field]
```

The CLOSE STORED-PROCEDURE is not used when the LOAD-RESULT-INTO keyword is used with the RUN STORED-PROCEDURE statement. See the .

For valid combinations and usage of these syntax elements, see:

- RUN STORED-PROC statement execution without the LOAD-RESULT-INTO phrase on page 90
- RUN STORED-PROC statement execution with the LOAD-RESULT-INTO phrase on page 91
- RUN STORED-PROC statement with send-sql-statement option on page 92
- RUN STORED-PROC statement with special internal option on page 93

The following sections describe the syntax elements and options in detail. These sections also present examples that describe typical situations in which you might use the various options that the RUN STORED-PROC statement supports.

RUN STORED-PROCEDURE statement

The RUN STORED-PROC statement runs an RDBMS stored procedure or allows you to send PL-SQL to an Oracle Server based data source using an OpenEdge DataServer. It contains a procedure-name which is either the:

- Name of the RDBMS stored procedure that you want to run
- OpenEdge built-in procedure name, send-sql-statement, to send PL/SQL to an Oracle Server based data source

LOAD-RESULT-INTO phrase

The LOAD-RESULT-INTO function loads the result set into a temp-table only where <handle> is a variable of type handle. Note that handle can also be defined as extent to enable you to pass more than one temp-table handle in those instances where the SQL statement is defined to return more than one result set.

When used with the send-sql-statement or stored procedure to load a result set into a temp-table, this function carries an implicit CLOSE-STORED PROCEDURE statement. However, the use of this function is optional so existing applications will not be affected.

Note: The OpenEdge client issues an error at runtime if the variable of type handle with the LOAD-RESULT-INTO function does not point to a temp-table.

When used with the LOAD-RESULT-INTO phrase, the temp-table handle identifies the temp-table to which the result set will be loaded.

You can specify an array of one or more temp-table handle elements to retrieve stored procedure result sets and have the DataServer load the result set data directly into the associated temp-table(s). This approach allows you to have direct access to the fields defined in the temp-table.

The following types of temp-tables can support result sets:

- **Static** — A temp-table whose schema is defined at compile time.
- **Dynamic** — A temp-table whose schema is defined at run time. There are two types of dynamic temp-tables: dynamic-prepared and dynamic-unprepared.

For additional details about using the LOAD-RESULT-INTO phrase with the temp-table handle, see Loading results into a temp-table on page 95.

PROC-STATUS phrase

The PROC-STATUS phase returns the return status from an Oracle Server stored procedure. The *return status* is an integer value that indicates whether a stored procedure succeeded or failed; if it failed, a code indicates why it failed. See your Oracle DataServer documentation for descriptions of the possible values for the return status.

PROC-HANDLE phrase

The PROC-HANDLE phrase allows you to specify a handle to act as a unique identifier for an Oracle DataServer stored procedure. For example, the PROC-HANDLE assigns a value to the specified integer field or variable (*integer-field*) that uniquely identifies the:

- Stored procedure that is returning results from Oracle
- SQL cursor that is used to retrieve results from an Oracle Server data source

Note these additional points about the PROC-HANDLE:

- Progress Software Corporation recommends that you specify a procedure handle for each stored procedure that you run.
- You do not have to specify a handle if there is only one active stored procedure and you do not include SQL statements in the OpenEdge application.

NO-ERROR option

The NO-ERROR option specifies that any ERROR condition that the RUN STORED-PROCEDURE statement produces is suppressed. Before you close a stored procedure, check the ERROR-STATUS handle for information on any errors that occurred. You receive an error when you attempt to close a stored procedure that did not start.

For more information on handling errors using the NO-ERROR option or a CATCH end block, see Error handling on page 69. For more information on the NO-ERROR option, CATCH statement or CATCH end block, in OpenEdge ABL, see *Develop ABL Applications*.

Note: This option must appear before any runtime parameter list.

PARAM phrase

The Param phrase identifies a run-time parameter to be passed to the stored procedure. A *parameter* has the following syntax:

```
[ ( [ INPUT | OUTPUT | INPUT OUTPUT ] [ PARAM parameter-name = ] expression, . . .
    [ INPUT | OUTPUT | INPUT OUTPUT ] [ PARAM parameter-name = ] expression ) ]
```

An *expression* is a constant, field name, variable name, or expression. INPUT is the default. OUTPUT and INPUT-OUTPUT parameters must be record fields or program variables.

If you run send-sql-statement for an Oracle-based data source, you must pass a single character expression *parameter* containing the SQL statement you want the data source to execute.

Note: The Oracle DataServer only supports one SQL statement with the send-sql-statement option.

If you do not specify *parameter-name* (the name of a keyword parameter defined by the stored procedure), you must supply all of the parameters in correct order. If you do specify *parameter-name*, you must precede your assignment statement with the keyword PARAM. If you do not supply a required parameter, and no default is specified in the stored procedure, you receive a run-time error.

CLOSE STORED-PROCEDURE statement

For a non-Progress stored procedure, PROC-STATUS indicates that the procedure has completed execution and retrieves any return status. For a send-sql-statement stored procedure, CLOSE STORED-PROCEDURE marks the stored procedure handle "reusable" and does not close the SQL cursor associated with the proc handle. When we get subsequent RUN STORED-PROC request, DataServer reuses the proc-handle if the SQL statement matches the existing statement in the cache. if not, it allocates a new proc-handle.

```
CLOSE STORED-PROCEDURE procedure-name
    [ integer-field = PROC-STATUS ]
    [  WHERE PROC-HANDLE = integer-field ]
```

`procedure-name`

The name of the stored procedure that you want to close or the built-in procedure name, `send-sql-statement`

`integer-field` = PROC-STATUS

Assigns the return value from a stored procedure to the specified integer field or variable (*`integer-field`*).

WHERE PROC-HANDLE = *`integer-field`*

An integer field or variable whose value uniquely identifies the stored procedure that produces the results returned from the data source or the SQL cursor of a `send-sql-statement` stored procedure.

Consider these points concerning the `CLOSED STORED-PROCEDURE` statement:

- If you specified a `PROC-HANDLE` when you ran a stored procedure, you must specify the `PROC-HANDLE` when you close the stored procedure.
- If you do not specify a `PROC-HANDLE`, the `CLOSE STORED-PROCEDURE` statement closes the procedure if there is only one stored procedure running. If there is more than one stored procedure running, an error is returned.

RUN STORED-PROC statement execution without the LOAD-RESULT-INTO phrase

To implement most data retrieval options associated with a stored procedure, (excluding those that involve loading retrieved data into temp-tables) you must define the `RUN STORED-PROC` statement and explicitly end the stored procedure using the `CLOSE STORED-PROC` statement.

You must define the `RUN STORED-PROC` and explicitly complete access to the stored procedure `OUTPUT` with the `CLOSE STORED-PROC` syntax expressions for a stored procedure implementation that retrieves the following types of data from the foreign data source:

- proc-text-buffer
- user-defined views

For more information about these data retrieval options, see Data output and retrieval options on page 94.

Note: You can also implicitly or explicitly close a stored procedure to retrieve these types of data: return code or output parameters. For more information about each of these data retrieval options, see Data output and retrieval options on page 94.

The syntax details presented in this section describe how you to run and close a stored procedure in the Oracle data source.

This is the syntax for the `RUN STORED-PROC` statement to execute an RDBMS stored procedure **without** the `LOAD-RESULT-INTO` phrase:

Syntax

```
RUN STORED-PROC procedure-name [<int> = PROC-HANDLE]
     [ NO-ERROR ]
   [ ( [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression, ...
     [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression ) ]
```

This type of syntax expression requires the explicit `CLOSE STORED-PROC` statement. This is the partial syntax for the `CLOSE STORED-PROC` statement:

Syntax

```
CLOSE STORED-PROC procedure-name
[integer-field = PROC-STATUS ]
    [ WHERE PROC-HANDLE = integer-field]
```

RUN STORED-PROC statement execution with the LOAD-RESULT-INTO phrase

OpenEdge also supports a stored procedure implementation you can execute to obtain result sets and load these results into temp-tables. You must define a result set target as part of the `RUN STORED-PROC` statement. You do not explicitly close a `RUN STORED-PROC` defined this way as you do with buffer methods of retrieving result sets; the `LOAD-RESULT-INTO` function achieves this goal implicitly.

This is the syntax for the `RUN STORED-PROC` statement implementation when loading result sets to temp-tables:

Syntax

```
RUN STORED-PROC procedure-name

[ [LOAD-RESULT-INTO <handle> [<int> = PROC-STATUS] ]
     [NO-ERROR]
    [ ( [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression, ...
        [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression) ]
```

The `RUN STORED-PROC` statement sets up the execution of the stored-procedure, retrieves the result set, and loads it into the temp-table(s) provided you use the `LOAD-RESULT-INTO` function. The `CLOSE STORED-PROC` statement is automatically applied when the result set is loaded into temp-tables because using temp-tables must consume all result sets of the procedure.

RUN STORED-PROC statement with send-sql-statement option

ABL also allows you to use stored-procedure syntax to send SQL statements and their native language extensions directly to a data source. The DataServer uses the RUN STORED-PROCEDURE statement with the send-sql-statement option to pass SQL statements to the data source. However, the Oracle DataServer only supports one SQL statement or PL/SQL block with the send-sql-statement option.

Like the option to define a RUN STORED-PROC statement either with or without the LOAD-RESULT-INTO options, you can define a send-sql-statement option with or without the LOAD-RESULT-INTO option. The following section presents and briefly describes an example use of each technique.

Without the LOAD-RESULT-INTO option

You use the RUN STORED-PROC statement with the send-sql-statement option and pass the PL/SQL statements as a parameter. The syntax of the statement must be valid PL/SQL syntax. The following example shows how the code presented passes a SELECT statement as a parameter.

Passing a SELECT statement as a parameter

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.

RUN STORED-PROC send-sql-statement iHandle = PROC-HANDLE NO-ERROR
 ("select cust_num,name like 'A_'").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = iHandle:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle.
```

This example returns the name of all customers whose name begins with A. You must read the results into a buffer as you would with a stored procedure called by an ABL procedure. You can read the results into the proc-text-buffer as defined in the example above. Alternatively, you can define your own buffer from within your data source that can accept other data type as well as the CHARACTER data type.

With the LOAD-RESULT-INTO option

The following example shows how to use the send-sql-statement option **with** the LOAD-RESULT-INTO option. It also shows that the PROC-STATUS phrase must be defined as part of the RUN STORED-PROC statement because of the implicit CLOSE STORED-PROC that is associated with the LOAD-RESULT-INTO phrase.

Using the send-sql-statement option with LOAD-RESULT-INTO option

```
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO.

DEFINE TEMP-TABLE tt1
  FIELD f1 AS INTEGER
  FIELD f2 AS CHARACTER.

ttHandle = TEMP-TABLE tt1:HANDLE.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHandle
  ("select cust_num, name like 'A_'").
```

In the previous example, also note that the PROC-STATUS phrase does not need a PROC-HANDLE phrase because it is retrieved using the RUN STORED-PROC statement; although the PROC-HANDLE is typically used after the execution of the RUN STORED-PROC statement, it is not needed in this context.

RUN STORED-PROC statement with special internal option

The following section describes the special internal option, ds-close-cursor.

ds-close-cursor option

The OpenEdge DataServer for Oracle supports a special built-in stored procedure call implementation that routes requests directly to the DataServer rather than routing request through the DataServer and onto the foreign data source. This special call enables you to close the inactive cursors—associated with the FIND statements that have already been executed in your application—that are either no longer (or minimally) required for future application use, thereby making more Oracle cursors available to your DataServer application. The close inactive cursor statement closes inactive cursors that are associated with FIND statements in your current application—directly from the SQL statement cache.

Syntax

```
RUN STORED-PROC ds-close-cursor ("FIND" [, number_of_cursors])
```

ds-close-cursor

The RDBMS special stored procedure call used to close inactive cursors in the Oracle DataServer.

FIND

Indicates the RDBMS stored procedure to close only the cursors associated with the FIND statement.

number_of_cursors

An optional parameter used to indicate the number of inactive cursors to be closed from the cursor cache.

Specify a value between 1 to 9999 for the number of inactive FIND cursors you want to close or specify no value to close all inactive FIND cursors from the application's SQL statement cache.

The following are examples of how to run stored procedures in ABL to close inactive cursors associated with the FIND statement.

RUN STORED-PROC for closing inactive cursors

```
RUN STORED-PROC ds-close-cursor ("FIND")
RUN STORED-PROC ds-close-cursor ("FIND",20)
```

The first and the second example in closes all the inactive cursors associated with the FIND statement.

The cursor closure operation can be initiated at any point in the run-time duration of the application. An entry is logged to `dataserv.lg` file each time the cursor closer operation is performed, provided the `cursor` log-entry-type is active.

Note: For more information on closing inactive cursors, see Closing inactive cursors on page 128.

- The special built-in procedure call is not a stored procedure call; it does not adhere to the rules and practices of using stored procedures.
- You do not need to acquire a handle to a special built-in procedure (ds-close-cursor) as there is no result set associated with the procedure call.
- You do not need to close the procedure call with CLOSE STORED-PROCEDURE.
- There is no error handling associated with the call, therefore, the NO-ERROR phrase is not applicable.

Data output and retrieval options

This section briefly highlights the data retrieval options that you can initiate through the execution of a stored procedure. See Interfacing with RDBMS stored procedures on page 96 for more information and examples of each option.

Return codes

This value might be a success code or a value returned by the stored procedure (as defined by the data source).

Values of output parameters you define when you create a procedure

When you call a stored procedure, you can specify the ordered list of positional parameters, or you can name them individually. To retrieve output parameter values from a stored procedure, request them with the keyword `OUTPUT` or `INPUT-OUTPUT` when you execute the procedure; your selected `INPUT` and `OUTPUT` parameter options must match the corresponding signature of the stored procedure at the data source.

Results retrieved from a database

This section identifies and briefly defines three coding techniques that ABL supports to retrieve data source results. In general, you can think of these techniques as handling complex query results for which each technique has its own purpose and benefits.

Using the OpenEdge-supplied proc-text-buffer for row results

The `Proc-text-buffer` is part of the schema holder, making this result set option independent of the foreign data source.

This technique:

- Allows you to access result sets available through a pseudo table for which each row is a long character string
- Requires you to parse each long string to access specific fields
- Referencing `Proc-text-buffer` with the `-q` extension enables you to access the character-converted results in quoted text (" ") from a data source. That is, `proc-text-buffer-q` causes quoted values in the character string and retains embedded quotes within the character result.

You can also override the default behavior of `proc-text-buffer` so that it behaves like `proc-text-buffer-q` with the help of a connection time DataServer option `PRGRS_QUOTE_PROCTEXT`. For more information on the `PRGRS_QUOTE_PROCTEXT` (`-Dsrv`) option, see Setting the data-source result set retrieval option on page 123.

Defining a special view on the Oracle data source to use as a buffer for row results

This technique involves some administration on the foreign datasource and therefore is considered a result set option that is dependent on the foreign data source.

This technique allows you to:

- Define views within the foreign data source
- Use the result set buffer configuration of the view to access the data returned by the stored procedure

Loading results into a temp-table

The temp-table approach maintains some separation between the data and the foreign data source which allows you to process the result sets quite independent of its database origin. Also, the result set definition is not contained within the schema holder.

This technique allows you to:

- Specify one or more temp-table handle elements to retrieve stored procedure result sets and have the DataServer load the result set data directly into the associated temp-table(s)
- Have direct access to the fields defined in the temp-table

Unless you pass unprepared dynamic temp-table handles into which the result sets are to be loaded, you still need to ensure that the temp-table record definition matches that of the stored procedures result set(s).

Passing empty temp-tables with the `LOAD-RESULT-INTO` phrase generates temp-table definitions dynamically based on a default mapping of data types. This method provides a data retrieval method completely independent of the foreign data source.

The `LOAD-RESULT-INTO` method represents a highly efficient means to transfer stored procedure result sets into temp-tables and provides significant performance gains over other techniques used to accomplish this same goal.

Note: The `proc-text-buffer` and the special native views techniques use the basic `RUN STORED-PROCEDURE` statement and are backward compatible to earlier versions of ABL. The `LOAD-RESULT-INTO` phrase provides a new technique to populate temp-tables directly in ABL.

Interfacing with RDBMS stored procedures

As previously mentioned, the RUN STORED-PROCEDURE statement is the initial statement required for the execution of all stored procedures you define and initiate through OpenEdge. It can run an RDBMS stored procedure bound by the data source, or allow you to send SQL to an Oracle Server data source using an OpenEdge DataServer. The specific, additional keywords and elements you must include in a stored procedure or define for a send-sql statement depend on the stored procedure characteristics including its signature, data results, and methods chosen to retrieve output to ABL procedure.

This section identifies and describes the syntax elements, keywords, and other requirements associated with obtaining stored procedure output from a stored procedure data request.

The following sections describe how to run Oracle stored procedures and retrieve return codes, output parameter values, and results sets.

Oracle stored procedure pcust

This example is the basis for the following examples of how to run stored procedures in ABL using this stored procedure created in Oracle.

```
CREATE PROCEDURE pcust (num IN INT, orders OUT INT, states OUT INT)
AS BEGIN
IF num IS NULL THEN
  raise_application_error (-20101, 'Cust Num is missing');
ELSE
  SELECT COUNT (*) INTO orders FROM customer, order_
    WHERE customer.Cust_num = order_.Cust_num AND customer.Cust_num > num;
SELECT count(*) INTO states FROM customer WHERE cust_num > num;
END IF;
END;
```

This PL/SQL code creates the stored procedure pcust and defines three parameters: num, orders, and states. The orders and states parameters are output parameters, which means that the procedure returns values for these parameters to the caller. All the parameters are of the data type INTEGER.

Note: Typically, you can have only fifty stored procedures running at one time. This number, however, is further restricted by the number of open cursors you specified for your Oracle database or for the current session. See Index cursors on page 197 for information on specifying open cursors. Cursor limitations also vary across platforms. See your Oracle documentation for more information.

Retrieving return codes

You can create a stored procedure that provides return code information. For example, it might indicate whether the stored procedure was successful or whether it encountered an error condition.

This example shows how ABL code runs the stored procedure pcust. It uses the PROC-STATUS function and the CLOSE STORED-PROC to retrieve the return code and assign the value to the variable stat.

ABL code running the stored procedure pcust

```
/* Return status */
DEFINE VARIABLE iStat AS INTEGER NO-UNDO.
RUN STORED-PROC pcust
  (PARAM num = ?, OUTPUT PARAM orders = 0, OUTPUT PARAM states = 0).
CLOSE STORED-PROC pcust iStat = PROC-STATUS.
IF iStat = 0 THEN
  DISPLAY pcust.orders pcust.states.
ELSE
  DISPLAY iStat.
```

The Oracle return codes have a range of values between -20000 and -20999. These values are user defined and you can test for them with the `PROC-STATUS` function.

Retrieving output parameter values

When you call a stored procedure, you can specify the ordered list of positional parameters or you can name them individually. To retrieve output parameter values from a stored procedure, request them with the keyword `OUTPUT` or `INPUT-OUTPUT` when you execute the procedure. When you run a stored procedure in a DataServer application, the parameters are supplied and passed using OpenEdge data types.

This example shows how the following ABL procedure uses the second option for passing parameters; it passes them by name with the `PARAM` option.

Passing parameters by name

```
/* Parameters by name */
RUN STORED-PROC pcust
  (PARAM num = 20, OUTPUT PARAM states = 0, OUTPUT PARAM orders = 0).
CLOSE STORED-PROC pcust.
DISPLAY pcust.orders pcust.states.
```

When you use `PARAM` to specify parameter names, you do not have to specify all parameters for the stored procedure. Instead, you can include only those parameters you want to use, in any order you choose. If the stored procedure names a default value for the parameter, you do not have to name that parameter at run time. However, you must explicitly name parameters that do not have defaults or name them when you want to pass values that are different from the default.

Retrieving results with cursor arguments

You can return result rows from stored procedures using named cursors as arguments. Cursor parameters are `OUTPUT` parameters only.

Use the following syntax to retrieve result rows:

Syntax

```
RUN STORED-PROC procedure-namevariable = PROC-HANDLE ( parameter-list ).
```

For example, the following code example returns rows from the customer table using the cursor named CUST_CURS:

```
RUN STORED-PROC open_cust h1 = PROC-HANDLE (CUST_CURS = ?, WITCH_V =1).
```

The DataServer retrieves the result rows and places them in a buffer. Specify the Oracle cursor where you want to fetch and process result rows by using the CURSOR option, as the following syntax and code example show:

Syntax

```
FOR EACH buffer-name WHERE PROC-HANDLE = variable  AND CURSOR =
[ [ db-name. ] procedure-name. ] parameter-name :
  DISPLAY buffer-name.
END.
```

```
RUN STORED-PROC open_cust h1 = PROC-HANDLE (CUST_CURS = ?, WITCH_V1).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = h1
  AND CURSOR = open_cust.CUST_CURS:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC open_cust.
```

The previous example code runs the stored procedure, open_cust, and displays the results fetched from the CUST_CURS cursor.

Note: If multiple cursors are associated with a stored procedure, you must specify a cursor by name when fetching results, otherwise the DataServer returns a run-time error. Always specifying PROC-HANDLE and cursor parameters ensures that your code continues to run if another cursor parameter is added to a stored procedure.

Retrieving results sets using proc-text-buffer

The proc-text-buffer technique offers one approach to access results returned from a data source. The following information is the partial syntax for the DEFINE BUFFER statement that you use to create a buffer with the same characteristics of the proc-text-buffer:

```
DEFINE BUFFER buffer-name FOR proc-text-buffer
```

For a complete description, see the "DEFINE BUFFER statement" reference entry in *ABL Reference*.

Technique to use proc-text-buffer

This example illustrates returning database results into the proc-text-buffer and converting the results to the INTEGER data type.

Returning database results into the proc-text-buffer and converting the results to the INTEGER data type

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.
DEFINE VARIABLE iMax    AS INTEGER NO-UNDO.

RUN STORED-PROC send-sql-statement iHandle = PROC-HANDLE
  ("SELECT max (custnum) FROM customer").
FOR EACH proc-text-buffer:
    iMax = INTEGER(SUBSTRING(proc-text, 1, 3)).
     DISPLAY iMax.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle.
```

The DataServer passes the PL/SQL statement directly to the Oracle data source. The ABL compiler does not process it, so errors occur only at run time and not when you compile a procedure.

Assessing a result set obtained from the proc-text-buffer technique

The advantage of using the `proc-text-buffer` is that you do not have to worry about what kind of data the procedure returns. The buffer accepts any type of data, in any order, and converts it to the character equivalent. Another benefit of the `proc-text-buffer` is that it can be used to hold the results from all of the SQL statements included in a stored procedure. However, a buffer that you create can hold the results of only one result set record form one result set at a time.

However, a disadvantage in using the `proc-text-buffer` technique is that it is much more difficult to manipulate the data after you receive it as it requires parsing the data. To act on anything but `CHARACTER` data, you must extract the data from the buffer and convert it to its original data type before you can use it.

Defining a view to use as a buffer

As an alternative to using the buffer `proc-text-buffer` defined by ABL, you can define a view in the data source that can serve as a buffer allowing you to retrieve database results in their original data types. However, keep in mind that using views creates a database dependency beyond the stored procedure itself.

Technique to define a view to use as a buffer

While a stored procedure can include multiple SQL statements, a buffer that you define contains the format of only a single result set. You need to define multiple views to accommodate multiple result sets.

To define a buffer:

1. Define a view in the Oracle data source with the following characteristics:

   - The naming convention `BUFFER_buffername`
   - The same number of columns and data types that the stored procedure returns in the result set
   - The columns in the order that the stored procedure returns them

   For example, to return two columns with two types of values, an integer and a character string, use an SQL utility to define the following view in the data source:

```
CREATE VIEW BUFFER_custlist AS SELECT customer.custnum,
  customer.name FROM customer WHERE 1 = 0
```

Notice that these views are defined to ensure that they never return any results. This helps to indicate that the purpose of the view is its buffer content and not its SQL capabilities. It is not necessary to define views that you will use as buffers this way, but it does allow you to distinguish quickly between views and buffers.

2. Update your schema image using the Update/Add Table Definitions DataServer utility. The utility adds the view to the list of accessible objects in the schema holder. The DataServer defines the view as a buffer that ABL can use. See Updating a schema image on page 221 for instructions on using this utility.

Assessing result sets obtained by defining a view as buffer technique

The buffer in the previous procedure defines two returned values for a stored procedure—an `INTEGER` and a `CHARACTER` value—in that order. If the data types do not match those returned by the stored procedure, the procedure returns more than two values, or returns the values in a different order than you specified, you receive a run-time error.

The easiest way to create a buffer that accepts data from stored procedures is to use the text of the SQL `SELECT` statement from the stored procedure. This ensures that you define your data types correctly and in the correct order.

Examples based on views created in the data source

The examples in this section do not use the supplied `proc-text-buffer` buffer. Instead, they show how to define formatted buffers by creating views in the data source, using the following syntax:

Syntax

```
CREATE VIEW BUFFER_buffer-name
```

The following two examples show the views created in your Oracle data source that you can use as buffers to store the results from the stored procedure `pcust`:

```
CREATE VIEW BUFFER_pcust_orders AS SELECT customer.custnum,
  customer.name, order.ordernum FROM customer, order WHERE 1 = 0
```

```
CREATE VIEW BUFFER_pcust_states AS SELECT customer.custnum, state.state
  FROM customer, state WHERE 1 = 0
```

Running the send-sql-statement option twice

This example runs the `send-sql-statement` option twice; procedure handles (through the `PROC-HANDLE` function) identify the different results from the Oracle database.

Note: This example is not intended to illustrate the use of the previous syntax.

```
DEFINE VARIABLE cSelect  AS CHARACTER NO-UNDO.
DEFINE VARIABLE iHandle1 AS INTEGER   NO-UNDO.
DEFINE VARIABLE iHandle2 AS INTEGER   NO-UNDO.

DEFINE BUFFER bfCustomer FOR customer.

RUN STORED-PROC send-sql-statement iHandle1 =
 PROC-HANDLE ("SELECT custnum, state FROM customer").

FOR EACH bfCustomer WHERE PROC-HANDLE = iHandle1:
  cSelect = "SELECT state, statename, region FROM state WHERE state = '" +
             bfCustomer.state + "'".

  RUN STORED-PROC send-sql-statement iHandle2 = PROC-HANDLE (cSelect).
  FOR EACH bfCustomer WHERE PROC-HANDLE = iHandle2:
    DISPLAY bfCustomer.
  END.
  CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle2.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle1.
```

If you use more than one `send-sql-statement` at a time to send `SELECT` statements, you must explicitly define procedure handles for each.

A stored procedure returning multiple result sets and how to access the result sets using cursor arguments and buffers

The following example is a two-part example. The code in the first part shows a stored procedure. The code in the second part shows the ABL procedure that is associated with the stored procedure.

```
/* First part of example - Stored procedure code */

create or replace package cv_types as
  type GenericCurType is ref cursor;
End cv_types;
/CREATE or replace PROCEDURE pcustorder (num IN INT, c1 out cv_types.
   GenericCurType, c2 out cv_types.GenericCurType)
AS BEGIN
IF num IS NULL THEN
raise_application_error (-20101, 'Cust Num is missing');
ELSE

open c1 for
SELECT cust_num,name FROM customer WHERE cust_num > num;

open c2 for
SELECT order_num,odate FROM order_WHERE order_.Cust_num > num;

END IF;
END;
/
```

The ABL procedure code in this example assumes that the pcust_buffer and porder_buffer exist in the schema and that they match the result-sets schema.

```
/* Second part of example - ABL procedure code*/

DEFINE VARIABLE hProc AS HANDLE NO-UNDO.

RUN STORED-PROC pcustorder
  (INPUT 10, OUTPUT 10, OUTPUT 10, OUTPUT 10, OUTPUT 10).
FOR EACH pcust_buffer WHERE PROC-HANDLE = hProc AND CURSOR = pcustorder.c1:
  DISPLAY pcust_buffer.
END.
FOR EACH porder_buffer WHERE PROC-HANDLE = hProc AND CURSOR = pcustorder.c2:
  DISPLAY porder_buffer.
END.
CLOSE STORED-PROC pcustorder.
```

As the above example shows, the stored procedure pcustorder returns multiple result-sets. The second code box shows the procedure used to access the result sets; it uses cursor arguments and buffers.

Because two different buffers have been defined, the returned values maintain their data types instead of being converted to character strings and stored in an ABL-defined buffer proc-text-buffer. You can then use the returned values in calculations without first converting them back to their original data types. In addition, the two separate buffers make your output look cleaner, allowing ABL to build a new default frame for the two different types of output. Reading your results into an explicitly defined buffer also allows you to manipulate the data just as you would manipulate data from an OpenEdge database; for example, with Frame phrases and FORM statements.

Loading a result set into a temp-table

Enhancements implemented through changes to the RUN STORED-PROC statement allow you to retrieve a result set from a foreign data source and load the result set, for which a temp-table handle is defined, into its own temp-table. The LOAD-RESULT-INTO function enables data retrieved to be loaded into temp-tables where the data can then be manipulated, employing all characteristics inherent to temp-tables. The capability to load result sets into temp-tables is not limited by the parsing requirements associated with the proc-text-buffer nor the database dependencies associated with views.

Temp-tables can provide data management capabilities associated with ABL directly to the result sets of a stored procedure, but completely independent of the foreign data source from which it was populated and/or derived. Temporary tables are effectively database tables in which OpenEdge stores data temporarily. Because temp-tables have the same support features that actual OpenEdge databases use, you can take advantage of almost all the OpenEdge database features that do not require data persistence and multi-user access. For example, you can define indexes for fields in the temp-table. For more information about temp-tables, see *Develop ABL Applications*.

Using the RUN STORED-PROC statement with a LOAD-RESULT-INTO phrase

This example introduces how to use the `RUN STORED-PROC` statement with the `LOAD-RESULT-INTO` phrase with a single dynamic temp-table. It highlights the coding techniques discussed in the RUN STORED-PROC statement with send-sql-statement option on page 92and introduces the dynamic temp-table topic further discussed in this section.

```
DEFINE VARIABLE bh      AS HANDLE NO-UNDO.
DEFINE VARIABLE hBrowse AS HANDLE NO-UNDO.
DEFINE VARIABLE hQuery  AS HANDLE NO-UNDO.
DEFINE VARIABLE tt1     AS HANDLE NO-UNDO.

DEFINE FRAME BrowseFrame WITH SIZE 80 BY 10.

CREATE TEMP-TABLE tt1.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO tt1
  ("SELECT * from CUSTOMER").

bh = tt1:DEFAULT-BUFFER-HANDLE.
CREATE QUERY hQuery.
hQuery:SET-BUFFERS (bh).
DISPLAY tt1:name.
hQuery:QUERY-PREPARE("FOR EACH" + tt1:name).
hQuery:QUERY-OPEN.

CREATE BROWSE hBrowse ASSIGN
  ROW         = 1
  COL         = 1
  WIDTH       = 79
  DOWN        = 10
  FRAME       = FRAME BrowseFrame:HANDLE
  QUERY       = hQuery
  SENSITIVE   = TRUE
  SEPARATORS  = TRUE
  ROW-MARKERS = FALSE
  VISIBLE     = TRUE.

hBrowse:ADD-COLUMNS-FROM(bh).
ENABLE ALL WITH FRAME BrowseFrame.
WAIT-FOR CLOSE OF CURRENT-WINDOW.
```

Getting started

If you are using static temp-tables, you must define the temp-table layout in your program to accommodate a specific stored procedure result set before attempting to populate these tables. Once this prerequisite task is done, however, temp-tables can also be automatically populated, offering a potential performance gain in most instances.

Unlike the proc-text-buffer technique, you do not have to parse the strings from the proc-text-buffer pseudo table buffer where each row is a character string. Similarly, you do not need to perform any administration to maintain views in the foreign data source or their definitions in the schema holder. For more details about planning your temp-table layout, see the Creating a temp-table layout plan on page 104 .

Employing additional enhancements

The temp-table technique offers even greater programming benefits as it extends both send-sql-statement options and stored procedures through the result processing techniques previously described in this chapter. For example, by mapping the PROGRESS_RECID to the ROWID field in temp- tables, you can easily support KEY definitions required by the ProDataSets to ensure that your data mappings between the foreign data source and the temp-table are accurate. Accurate data mappings are essential for sending data back from the temp-table to the data source. Non-ROWID key definitions can also be described with a unique key.

ProDataSets functionality is based on one or more temp-tables that share and extend basic temp-table functionality. For more information about ROWID field and using the send-sql-statement with the LOAD-RESULT-INTO option, see ROWID support on page 113. For in-depth discussions of temp-tables and more information about ProDataSets, see *Use ProDataSets*.

The following table highlights additional language elements you can use with the stored procedure and the send-sql statement language to use ROWID.

Table 21: Returning result sets and loading the data into temp-tables

ABL	Description
RUN STORED-PROCEDURE statement	Executes the stored procedure or send-sql-statement option and tells OpenEdge that the stored procedure has ended.
LOAD-RESULT-INTO phrase	Allows data from a result set that is returned for a foreign data source either through a stored procedure or a send-sql- statement option to be put into one or more temp-tables. Static, unprepared dynamic, and prepared dynamic temp-tables are supported. Only one result set can be returned when when using the send-sql-statement option. **Note:** When using SQL statement(s) through a send-sql-statement option or stored procedure to load result sets into temp-tables, RUN STORED-PROC carries an implicit Run CLOSE-PROC statement. (The stored procedure's output parameters are available after the RUN STORED-PROC executes and closes the procedure.)
PROC-STATUS phrase	Reads the return value (optional).

Creating a temp-table layout plan

You must define the temp-table layout in your application program to accommodate specific result sets **before** you attempt to populate the temp-tables with data. If a SQL statement retrieves more than one result set, you must define multiple temp-tables to be able to retrieve all the data. Therefore, the success of this approach depends to a large extent on your:

- Understanding of the specific data your foreign data source is providing you through a given stored procedure
- Ability to correctly define temp-tables

The following types of temp-tables can support result sets:

- **Static** — A temp-table whose schema is defined at compile time.
- **Dynamic** — A temp-table whose schema is defined at run time. There are two types of dynamic temp-tables: dynamic-prepared and dynamic-unprepared.

Keep in mind that you can pass handles of temp-tables that contain a mixed array. A *mixed array* is one in which some of the temp-table handle elements can be static while others can be dynamic. Also, note that a stored procedure supports the use of an `INT64` data type in static and dynamic temp tables when the `LOAD-RESULT-INTO` phrase processes the procedure's result set on the `RUN-STORED-PROC` statement.

The following table identifies the temp-table options for which you can plan and the requirements you must fulfill for each option.

Table 22: Options to plan the temp-table layout for result sets

To return a result set to this type of temp-table . . .	Then the layout definition is . . .
StaticDynamic-prepared state	Defined by you; you must base the layout on the expected fields to be returned and each of these fields' data types so that the first field defined in the temp-table corresponds to the first column of the result set. This column matching and data type matching must be repeated successfully for each temp-table and its corresponding result set.
Dynamic - unprepared state	Not defined by you; the schema of the temp-table is based on the result-set schema and a mapping of default ABL data types for each SQL type. For more information, see Details about a dynamic temp-table in an unprepared state on page 106.

Note: Once the data is loaded into the temp-table, any updates made to the records in the temp-table are not propagated back to the foreign database. Result sets are available through temp-tables for the purpose of obtaining a snapshot of the data. For example, you can use this technique to populate a browser from a temp-table. You must re-read the record using the proper lock mechanism to actually update the record.

Using a temp-table handle with an unprepared dynamic temp-table

When a temp-table handle points to an unprepared dynamic temp-table, the Oracle Server DataServer defines the temp-table schema in the form of the result sets record structure which is passed back to the DataServer from the foreign data source. The data types defined for the temp-table schema are determined based on the default data type mapping that exists between the SQL data type and its equivalent ABL default data type. Once the temp-table schema is dynamically established by the DataServer, the result set begins to populate it.

Recognize that there is the possibility of a small performance price to be paid when you build dynamic temp-tables. However, considering the database independence that this technique affords over building static temp-tables, you might consider the price of dynamically built temp-tables to be a small, reasonable one.

Details about a dynamic temp-table in an unprepared state

A dynamic temp-table is considered to be in an unprepared state after the first definitional method is called until the temp-table is prepared. If a clear dynamic temp-table handle is passed, the DataServer populates the temp-table schema based on the result-set schema and prepares the temp-table. A *clear dynamic temp-table* is a table that is in an unprepared state where definitional methods have not yet been called. The DataServer then executes the temp-table handle: `ADD-NEW-FIELD` (field name, data type) internally for each one of the columns of the result set. If a column in the result set from the foreign schema does not have a name (for example, an array element field or a SQL-derived field), the DataServer assigns a name to it based on the column position on the row.

For example, if you run the following statement, then the temp-table contains columns custnum, name and column4:

```
/* */
SELECT "mytag", custnum, name, (salary * 0.10) FROM <anytablename>
```

The data type associated with each column follows the mapping rules that exist between OpenEdge and the foreign data source's data types. For more information about data types and default mapping, see Initial Programming Considerations on page 35

Note: Since a stored procedure can return multiple result sets, the DataServer prepares the temp-table as "`result<n>`" where <`n`> is the result-set sequence number that corresponds to its element position in the temp table handle array, starting with 1. Therefore, if there are 3 result sets and 3 clear dynamic temp-tables are passed, the temp-tables are called result1, result2, and result3.

Note the following error conditions as they specifically apply to a dynamic temp-table:

- If an error occurs during the schema population or during the prepare of a temp-table, the DataServer raises an error condition.
- The dynamic temp-table must be either already prepared or clear with no defined fields in it. If fields have already been added to a dynamic temp-table before the `RUN STORED-PROC` statement is executed and the temp-table is not prepared, the DataServer raises an error condition due to the invalid state of the temp-table.

Details about a Dynamic temp-table in a prepared state

This example shows multiple dynamic temp-tables in a prepared state. It is based on the stored procedure code presented in which shows the basics of executing a call to a stored procedure that returns multiple result sets using the functionality that the LOAD-RESULT-INTO phrase supports. Each result set will be loaded into a separate temp-table.

Dynamic temp-tables shown in a prepared state

```
/* Second part of this example - ABL code */

DEFINE VARIABLE tt1      AS HANDLE NO-UNDO.
DEFINE VARIABLE tt2      AS HANDLE NO-UNDO.
DEFINE VARIABLE tt-array AS HANDLE NO-UNDO EXTENT 2.

CREATE TEMP-TABLE tt1.
tt1:ADD-NEW-FIELD("custnum", "integer").
tt1:ADD-NEW-FIELD("name", "character").
tt1:TEMP-TABLE-PREPARE("custx1").

CREATE TEMP-TABLE tt2.
tt2:ADD-NEW-FIELD("ordernum", "integer").
tt2:ADD-NEW-FIELD("order-date", "date").
tt2:TEMP-TABLE-PREPARE("ordx1").

ASSIGN tt-array[1] = tt1
       tt-array[2] = tt2.

RUN STORED-PROC pcustorder LOAD-RESULT-INTO tt-array
  (INPUT 10, OUTPUT 0, OUTPUT 0).
```

Additional temp-table examples

This section presents more examples that show various techniques to code temp-tables.

Executing a stored procedure call using the LOAD-RESULT-INTO phrase

This example shows the basics of executing a call to a stored procedure using the functionality that the `LOAD-RESULT-INTO` phrase supports. Note that the code works with the stored procedure in Example comparable to the way you can see it works with ABL code.

```
/* Calling a stored procedure, using a static temporary table */

DEFINE VARIABLE ttArray AS HANDLE EXTENT 2.

DEFINE TEMP-TABLE tt1
  FIELD custnum AS INTEGER
  FIELD name    AS CHARACTER.

DEFINE TEMP-TABLE tt2.
  FIELD ordernum  AS INTEGER
  FIELD orderdate AS DATE.

ASSIGN ttArray[1] = TEMP-TABLE tt1 HANDLE
       ttArray[2] = TEMP-TABLE tt2:HANDLE.

RUN STORED-PROC pcustorder LOAD-RESULT-INTO ttArray
  (INPUT 10, OUTPUT 0, OUTPUT 0).
```

Using an existing temp-table without the TEMP-TABLE-PREPARE ( ) method

This example shows the basics of using an existing dynamic temp-table without the TEMP-TABLE-PREPARE () method. In this instance, the send-sql-statement option is used rather than a predefined stored proc. In contrast, the third example code that appears later in this section shows the same approach, but explicitly defines the existing dynamic temp-table with the TEMP-TABLE-PREPARE() method.

```
/* Calling a stored procedure, using an existing temp-table without temp table
   prepare */

DEFINE VARIABLE ttHndl AS HANDLE NO-UNDO.

CREATE TEMP-TABLE ttHndl.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHndl
  ("select * from customer").
```

Using an existing temp-table with the TEMP-TABLE-PREPARE ( ) method

This example shows the basics of using an existing dynamic temp-table with the TEMP-TABLE-PREPARE() method.

```
/* Calling a stored procedure, using an existing temp-table with temp table
   prepare*/

DEFINE VARIABLE ttHndl AS HANDLE NO-UNDO.

CREATE TEMP-TABLE tt1Hndl.
ttHndl:ADD-NEW-FIELD("custNum","integer").
ttHndl:ADD-NEW-FIELD("name","char").
ttHndl:TEMP-TABLE-PREPARE("ordX").

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHndl
  ("selec custNum, name from myDB.customer").
```

Note these points as they relate to the above example:

- As a prerequisite for creating the code shown in the previous example, the developer would need to define the schema for the table.
- Once the temp-table schema begins preparation from the clear state, the temp-table must be defined to the exact specifications for the result sets as generated by the RUN STORED-PROC statement or send-sql-statement option. Otherwise, the RUN STORED-PROC will end in failure. Also, note the TEMP-TABLE-PREPARE must be called at the completion of the temp-table definition associated with the stored procedure results.

A stored procedure that uses the LOAD-RESULT-INTO phrase with a single temp table and the send-sql-statement option

This example shows the syntax for the stored procedure with the LOAD-RESULT-INTO phrase with a single static temp-table and the send-sql-statement option.

```
/* Calling a stored procedure that uses the LOAD-RESULT-INTO phrase with a
   single temp table and the send-sql-statement option */

DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO.

DEFINE TEMP-TABLE tt1
  FIELD f1 AS INTEGER
  FIELD f2 AS CHARACTER.

ttHandle = TEMP-TABLE tt1:HANDLE.
RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHandle
  ("Select cust_num, name from customer").
```

Using the PROC-STATUS phrase

This example shows the use of a PROC-STATUS phrase. The PROC-STATUS phrase must be defined as part of the RUN STORED-PROC statement because of the implicit CLOSE STORED-PROC that is associated with the LOAD-RESULT-INTO phrase.

```
/* Example of the implicit close stored-proc and use of LOAD-RESULT-INTO */
 DEFINE VARIABLE iStat AS INTEGER NO-UNDO.
DEFINE VARIABLE ttHandle AS HANDLE  NO-UNDO.

CREATE TEMP-TABLE ttHandle
RUN STORED-PROCEDURE pcust LOAD-RESULT-INTO ttHandle
   iStat = PROC-STATUS (20, OUTPUT 0, OUTPUT 0).
DISPLAY iStat.
```

In this example, note that the PROC-STATUS phase does not need a PROC-HANDLE phrase because it is retrieved using the RUN STORED-PROC statement and not after this statement's execution as it typically is used.

Executing a call to a stored function that returns a result set using the LOAD-RESULT-INTO phrase.

This example is a two-part example that shows the basics of executing a call to a stored function that returns a result set using a cursor and the functionality that the LOAD-RESULT-INTO phrase supports. The result set will be loaded into a temp-table.

```
/* First part of the procedure - Stored-function code*/

create or REPLACE function myfunc_2 (num in number) return cv_types.
GenericCurType is c1 cv_types.GenericCurType;
begin

OPEN c1 FOR
   select *
     from customer where cust_num>=num;

return c1;
end;
/
```

```
/* Second part of the procedure - ABL code*/

DEFINE VARIABLE tth AS HANDLE NO-UNDO.

CREATE TEMP-TABLE tth.
RUN STORED-PROC myfunc_2 LOAD-RESULT-INTO tth (OUTPUT ?, INPUT 10).
```

Handling errors

The RUN STORED-PROC statement supports the NO-ERROR option. and these examples show how to trap errors within a procedure using the NO-ERROR option.

How to trap errors within a stored procedure using NO-ERROR option

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

RUN STORED-PROC send-sql-statement NO-ERROR
  ("select count (*) from customer where name between a and 'z' ").
/*Select statement syntax is incorrect due to type-mismatch*/
IF ERROR-STATUS:ERROR THEN DO:
  DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE "error" ERROR-STATUS:GET-NUMBER(ix)
    ERROR-STATUS:GET-MESSAGE(ix).
  END.
  RETURN.
END.
CLOSE STORED-PROC send-sql-statement.
```

Trapping errors within a procedure using NO-ERROR option

This example assumes that you used the `customer` table from the demo database to create `sp_write_cust` procedure that inserts two records into the `customer` table:

```
create or replace procedure sp_write_cust(
sp_cust_num in out number,
sp_address in out varchar2
)as
begin
  insert into customer(cust_num,address) values(sp_cust_num, sp_address);
  insert into customer(cust_num,address) values(sp_cust_num, sp_address);
  commit; /* Procedure accidently commits unique cust_num twice */
  select cust_num, address into sp_cust_num, sp_address from customer
    where   cust_num= sp_cust_num;
end sp_write_cust;
/
```

In this example, `sp_write_cust` procedure execution causes a unique index error because it tries to insert two values for `cust_num`, a primary key of the `customer` table. Then, the unique index error is handled by `ERROR-STATUS` handle in the `IF` block.

```
DEF VAR intvar AS INTEGER.
DEF VAR ix AS INTEGER.
DEF VAR num AS INTEGER.
DEF VAR myaddr AS CHARACTER.
num =1.
myaddr = '21 Main Street'.
RUN STORED-PROC sp_write_cust intvar = PROC-HANDLE (PARAM sp_cust_num = num,
                                      PARAM sp_address = myaddr).

IF ERROR-STATUS:ERROR AND ERROR-STATUS:NUM-MESSAGES > 0 THEN DO:
  DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE ERROR-STATUS:GET-NUMBER(ix) ERROR-STATUS:GET-MESSAGE(ix).
  END.
  RETURN.
END.
CLOSE STORED-PROCEDURE sp_write_cust WHERE PROC-HANDLE = intvar.
DISPLAY "Customer " + string(num) + " at address " + myaddr + " was added to the
 database".
```

The DataServer internally closes the stored procedure handle when the stored procedure fails to execute. You must ensure that the error handling code block immediately follows the `RUN STORED-PROCEDURE` call so that errors captured using `NO-ERROR` are not lost by the execution of another statement. Including the `CLOSE STORED-PROCEDURE` statement in the error handling block causes a run-time error because closure is implicit. Only if the procedure is run successfully will a procedure handle be established, and only then do you need to execute `CLOSE STORED-PROCEDURE` on the open handle.

The error in this example can also be handled using a `CATCH` end block in the procedure block that performs the `RUN STORED-PROC` statement.

Trapping errors within a procedure using CATCH option

This example shows how to trap errors within a procedure using a CATCH end block.

```
DEF VAR intvar AS INTEGER.
RUN STORED-PROC sp_img_out intvar = PROC-HANDLE (3, OUTPUT ?).
CLOSE STORED-PROCEDURE sp_img_out WHERE PROC-HANDLE = intvar. /* This statement
 will not be executed on error */

CATCH eSysError AS Progress.Lang.ProError:

  MESSAGE eSysError:GetMessageNum(1) SKIP
    eSysError:GetMessage(1) SKIP
    eSysError:GetMessage(2) SKIP
    eSysError:GetMessage(3) VIEW-AS ALERT-BOX BUTTONS OK.
    /* Handler code for SysError condition. */
END.
```

For more information on the NO-ERROR option, CATCH statement or CATCH end block, in OpenEdge ABL, see *Develop ABL Applications*.

Trapping errors when using LOAD-RESULT-INTO

By properly positioning the NO-ERROR in the RUN statement, error information can also be retrieved from attempted SQL execution at the data source when using LOAD-RESULT-INTO. shows how errors are trapped after LOAD-RESULT-INTO stored procedure execution.

```
DEFINE VARIABLE tables AS HANDLE EXTENT 1 NO-UNDO.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO tables NO-ERROR
  ("Invalid SQL Statement").

IF ERROR-STATUS:ERROR THEN
  MESSAGE ERROR-STATUS:GET-MESSAGE(1) VIEW-AS ALERT-BOX.
ELSE
  MESSAGE "SQL call was successful." VIEW-AS ALERT-BOX.
```

Error messages related to using the LOAD-RESULT-INTO phrase

Note the following situations in which error conditions related to the use of LOAD-RESULT-INTO are reported:

- The ABL compiler issues an error if you try to specify PROC-HANDLE and LOAD-RESULT-INTO on the same Run STORED-PROC statement.
- The ABL compiler issues an error if you use PROC-STATUS without specifying the LOAD-RESULT-INTO function.
- The compiler issues error messages when the number of handles is not equal to the number of result-sets specified in the statement. For example:
  - **Error "Only 2 handles supplied but there are at least 3 result-sets (12209)"** — This error message appears when two handles and three result-sets are specified in a statement.

- **Error "Received an invalid handle (element 3) when processing stored procedure's result-set (12212)"** — This error message appears when four handles and three result-sets are specified in a statement.

- If you try to execute the `CLOSE STORED-PROC` statement after the `Run stored-proc statement` for which you have defined the `LOAD-RESULT-INTO` function, the following error message appears:

  - **Error "No active stored procedures. (2101)"** — This error message occurs when no handle is specified in the statement.
  - **Error "No active procedure for specified proc-handle. (2475)"** — This error message occurs when a handle is specified in the statement.

  The system generates either of the previous error messages as it assumes a second `CLOSE STORED-PROC` statement has been incorrectly executed in the current implementation.

ROWID support

This section presents details about `ROWID` function.

- Understanding the ROWID Implementation on page 113
- ROWID characteristics on page 113
- Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO on page 114
- Additional ProDataSet support on page 117

Understanding the ROWID Implementation

The OpenEdge `ROWID` function returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type `ROWID`, which is supported for OpenEdge and all other DataServer databases.

The dataserver designates a column to support the `ROWID` column. It evaluates the indexes available for a table and selects one in the following order:

- `PROGRESS_RECID` column
- Unique index on a single, mandatory, `NUMBER` with precision that is less than 10 or undefined, and scale less than or equal to zero or undefined
- The Oracle native `ROWID`

ROWID characteristics

If you migrated your database to Oracle and elected to use the `CREATE RECID FIELD` option during the migration, a unique 8-byte Integer column named `PROGRESS_RECID` will have been automatically generated in your database tables along with a corresponding Index containing the `PROGRESS_RECID` value as its key.

Databases that are only pulled from the native environment or are migrated without the CREATE RECID FIELD option must either choose a unique index key from each table or the native ROWID to represent the PROGRESS_RECID to support the RECID and ROWID functions, and forward and backward scrolling within their ABL sessions. OpenEdge will internally map the selected unique key or native ROWID to the PROGRESS_RECID and ROWID functions. It is important to note that the unique index key used to derive the OpenEdge ROWID must be a single component index for it to map to the ROWID of a temp-table as described in the following discussion. For more details, see the ROWID function discussion in Initial Programming Considerations on page 35.

Also, before discussing ROWID as it relates to the RUN STORED-PROC statement, it is important to understand an important property of the OpenEdge ROWID. See ROWID: Standard ABL behavior on page 114 for more information.

ROWID: Standard ABL behavior

The ROWID value of a temp-table buffer will be different than the ROWID value of a record BUFFER even if the underlying data is identical. This difference exists because the ROWID function relies on the record buffer name.

Expected ABL behavior—ROWID value of a Temp-Table buffer

This example shows the standard expected ABL behavior.

```
DEFINE VARIABLE rid-1    AS ROWID  NO-UNDO.
DEFINE VARIABLE rid-2    AS ROWID  NO-UNDO.
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO EXTENT 1.

DEFINE TEMP-TABLE ttCust LIKE Sports.Customer
  FIELD tRecid       AS INTEGER
  FIELD tRECID_ident AS INTEGER.

ttHandle[1] = TEMP-TABLE ttCust:HANDLE.

FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-1 = ROWID(customer).

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHandle
  ("select * from customer where custnum=1").
rid-2 = ROWID(ttCust).

IF rid-1 <> rid-2 THEN
  MESSAGE "The same record but different ROWID's".
```

The following sections illustrate the differences between the ROWID value of a temp-table buffer and the ROWID value of a record BUFFER so that you can best understand, program for, and leverage the use of the ROWID function with the LOAD-RESULT-INTO clause of the RUN STORED-PROCEDURE command.

Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO

The example in this section assumes you migrated your database to Oracle using the CREATE RECID option, as discussed in the ROWID characteristics on page 113 . This example and all subsequent examples show ROWID as being represented by the 4-byte integer value of the PROGRESS_RECID column as opposed to some other unique single-component index designated in your database to be the Progress RECID.

Note: If you used a different single-component index to load the ROWID of a temp-table or native ROWID, you would need to map it accordingly, just as the example maps PROGRESS_RECID.

The RUN STORED-PROC command has no native awareness that the Oracle Database Table is being queried for the result set(s) it generates. Therefore, to allow DataServer technology to convert the stored PROGRESS_RECID value into a native Progress ROWID value, the physical name of the target database table must be known. To achieve this bond, the temp-table that the stored procedure populates must be associated with an OpenEdge ProDataSet object.

Simple ProDataSet code

This example shows an ABL query filling the temp tables of a ProDataSet. It is used as the baseline code which is referenced throughout the remainder of this section.

```
DEFINE VARIABLE phDataSet AS HANDLE NO-UNDO.

DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
  FIELD iRecid AS INTEGER.

DEFINE DATASET dsCust FOR ttCustomer.
DEFINE QUERY qCustomer FOR Customer.

phDataSet = DATASET dsCustomer:HANDLE.
DEFINE DATA-SOURCE srcCustomer FOR QUERY qCustomer.
BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE
  (DATA-SOURCE srcCustomer:HANDLE,?,?,?).
QUERY qCustomer:QUERY-PREPARE("FOR EACH customer").
DATASET dsCustomer:FILL().

FOR EACH ttCustomer:
  DISPLAY ttCustomer.name ttCustomer.iRecid.
END.
```

Using the LOAD-RESULT-INTO technique to populate the underlying temp-table of a ProDataSet

This example combines code from the above codes applying the results of the `RUN STORED-PROC [LOAD-RESULT-INTO]` technique, rather than an ABL query, to fill the temp-table associated with a ProDataSets. Key points about this example are presented following the example.

```
DEFINE VARIABLE hSendSQL  AS HANDLE NO-UNDO EXTENT 1.
DEFINE VARIABLE phDataSet AS HANDLE NO-UNDO.
DEFINE VARIABLE rid-1     AS ROWID  NO-UNDO.
DEFINE VARIABLE rid-2     AS ROWID  NO-UNDO.

DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
  FIELD rRecid AS ROWID. /* MUST BE CHANGED TO ROWID TYPE */

hSendSQL[1] = TEMP-TABLE ttCustomer:HANDLE.

DEFINE DATASET dsCustomer FOR ttCustomer.
DEFINE QUERY qCustomer FOR Customer.

phDataSet = DATASET dsCustomer:HANDLE.

DEFINE DATA-SOURCE srcCustomer FOR QUERY qCustomer.
BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE
  (DATA-SOURCE srcCustomer:HANDLE,?,?,?).

FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-1 = ROWID(customer).

/* Populate the ttCustomer temp-table */
RUN STORED-PROC send-sql-statement
  LOAD-RESULT-INTO hSendSQL ("select * from customer").

FIND FIRST ttCustomer WHERE ttCustomer.custnum = 1 NO-LOCK.
rid-2 = ttCust.rRecid.

If rid-1 <> rid-2 THEN
  MESSAGE "The same record but different ROWID's".
IF rid-1=rid-2 THEN
  MESSAGE "Congratulations - we have the same ROWID's".

MESSAGE STRING(ttCustomer.rRecid) VIEW-AS ALERT-BOX.
```

Keep the following key points in mind as you review this example:

- The `TEMP-TABLE` field that is mapped to the `PROGRESS_RECID` column should be changed from its standard definition of `INTEGER` to `ROWID`. In this example, the result column location where `PROGRESS_RECID` is being returned has been named `rRecid` in the temp-table. This renaming occurs in Example 3-26 because of the following line:

```
FIELD rRecid       AS ROWID /* must be changed to ROWID type */
FIELD iRECID_ident AS INTEGER.
```

- The `TEMP-TABLE` must be defined to the ProDataSet. The following line, another excerpted line of code from this example, shows this definition:

```
DEFINE DATASET dsCustomer FOR ttCustomer.
```

- The technique, demonstrated in this example, does not change the default behavior of the ROWID function, but provides a mechanism for the ROWID value to be stored along side its corresponding result rows; therefore, using the ROWID value to access database rows is unconventional with respect to the normal, or more typical, association between ROWID and a database table row. The following code demonstrates this difference.

  Default use of ROWID function on a record buffer, as excerpted from this example:

```
FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-1 = ROWID(customer).
```

In contrast, the following excerpt from this example demonstrates an alternative use of the ROWID value with a temp-table:

```
FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-2 = ttCustomer.Recid.
```

Additional ProDataSet support

As demonstrated in the above example, the LOAD-RESULT-INTO functionality provides a very fast and efficient way to populate the temp-table(s) of a ProDataSet Object.

Using the LOAD-RESULT-INTO technique with BEFORE-FILL method to fill the temp-table(s) of a ProDataSet

This example builds on the techniques demonstrated in , but the previous example shows the ProDataSet object `BEFORE-FILL` procedure can be modified to provide a single ProDataSet data access object that can be used against native OpenEdge or against Oracle and other DataServer data sources.

```
DEFINE VARIABLE hSendSQL  AS HANDLE NO-UNDO EXTENT 1.
DEFINE VARIABLE phDataSet AS HANDLE NO-UNDO.
DEFINE VARIABLE rid-1     AS ROWID  NO-UNDO.
DEFINE VARIABLE rid-2     AS ROWID  NO-UNDO.

DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
  FIELD rRecid AS ROWID.

hSendSQL[1] = TEMP-TABLE ttCust:HANDLE.

DEFINE DATASET dsCustomer FOR ttCustomer.
DEFINE QUERY qCustomer FOR Customer.

phDataSet = DATASET dsCust:HANDLE.
phDataSet:SET-CALLBACK-PROCEDURE
  ("BEFORE-FILL", "preDataSetFill", THIS-PROCEDURE).

DEFINE DATA-SOURCE srcCustomer FOR QUERY qCustomer.
BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE
  (DATA-SOURCE srcCustomer:HANDLE,?,?,?).
DATASET dsCustomer:FILL().

FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-1 = ROWID(customer).

FIND FIRST ttCustomer WHERE ttCustomer.custnum = 1 NO-LOCK.
rid-2 = ttCustomer.rRecid.

IF rid-1 <> rid-2 THEN
  MESSAGE "The same record but different ROWID's".
IF rid-1=rid-2 THEN
  MESSAGE "Congratulations - we have the same ROWID's".

MESSAGE STRING(ttCustomer.rRecid) VIEW-AS ALERT-BOX.
```

4

Additional Features to Enhance DataServer Performance

This chapter presents additional features to enhance the Oracle DataServer performance.

For details, see the following topics:

- Enhancements overview
- SQL and ABL queries
- Bind variables
- Caching records
- Oracle hints
- Using field lists when updating records
- Setting the data-source result set retrieval option
- Join by SQL DB
- Skipping schema verification
- Writing queries for performance
- Bulk-insert with ABL
- Closing inactive cursors
- Query tuning

Enhancements overview

You can design a DataServer application that emphasizes portability across data sources or one that optimizes the strengths of the DataServer's interaction with a particular aspect of the Oracle RDBMS. For example, you can write a query that gives you consistent results across databases or that takes advantage of Oracle's cursor management.

In addition to influencing how DataServer applications perform through queries, you can control how the DataServer processes queries on a statement-by-statement basis. Some of the DataServer's default behavior might not be optimal for the application you are designing. The `QUERY-TUNING` phrase and startup and connection parameters give you the ability to control query processing. In addition to reading the following sections, see Query tuning with connection and startup parameters on page 178 for information on tuning queries at compile time and run time. See Sample Queries on page 261 for examples of how the DataServer handles queries with and without query-tuning options.

SQL and ABL queries

The DataServer allows you to use different approaches for querying an Oracle database. Your application might be able to take advantage of the strengths of the different approaches depending on the kind of query you are writing and the kind of data you are accessing. The approaches are:

- **ABL** — The DataServer generates optimal SQL for `DEFINE QUERY` and `FOR EACH` statements, but you can use the `QUERY-TUNING` option to customize the queries that the DataServer passes to Oracle.
- **ABL SQL SELECT** — When you use an SQL `SELECT` statement in an OpenEdge procedure, the DataServer passes the SQL to Oracle. This approach can improve performance, especially when counting records, and allow you to access certain types of data more effectively, such as aggregates.
- **OpenEdge SQL-92** — Do not use SQL-92 syntax in applications that access the DataServer. The OpenEdge SQL Engine (which compiles SQL-92) is not integrated into the DataServer architecture.
- **Oracle SQL** — If you want to use specialized query syntax supported only by SQL, you can use the `RUN-STORED-PROC send-sql-statement` to send the syntax to Oracle. If you want to use `BEGINS` as a search criterion, an SQL query can result in better performance. Note, however, that ABL and SQL queries produce different results when accessing `CHAR` data because ABL uses bind variables. Use the `QUERY-TUNING NO-BIND-WHERE` option in the ABL query for results that are more similar to results from an SQL query.

Bind variables

The DataServer has enhanced memory management for bind variables, which causes variables in SQL statements to be rebound less frequently.

SQL `SELECT` statements use bind variables. When you use the SQL `SELECT` statement in an ABL procedure, the OpenEdge DataServer for Oracle uses bind variables and generates reusable SQL.

For ABL queries that reference a `DATE` in the `WHERE` clause, the DataServer uses a bind variable instead of a literal for the `DATE` value and generates reusable SQL.

Note: The DataServer does not substitute bind variables for values in SQL statements that you send directly to Oracle using the ABL `RUN STORED-PROC send-sql-statement` syntax.

Caching records

The DataServer caches results sets from the Oracle database to enhance performance. It caches as much data as fits in its allocated cache size. Depending on what kind of cursor a query is using, the DataServer caches row identifiers or records:

- **Standard cursors** — The DataServer caches row identifiers for the results set. If the database table is using the native `ROWID` as the row identifier, each identifier requires 18 bytes of cache. If the table is using a `PROGRESS_RECID` column or another index as the row identifier, each identifier requires 4 bytes of cache. Therefore, a results set of 100 records requires either 1800 or 400 bytes of cache.
- **Lookahead cursors** — The DataServer caches complete records or partial records as specified by a field list. The DataServer uses the maximum length allowed for a row as defined in the Oracle database to calculate the record length, not the actual contents of the record. In addition to the defined row length, the record consists of a row identifier field. Therefore, a row with a defined maximum length of 100 bytes and a native `ROWID` field (used by the DataServer as the row identifier) requires 118 bytes of cache. The DataServer counts a `LONG` or `LONG RAW` column as being 256 bytes long. If a `LONG` or `LONG RAW` column is longer than 256 bytes, the DataServer refetches it.

In the case of joins, each record in the cache is a result of the fields selected in the join. In addition to the record, there is a row identifier field (4 or 18 bytes) for each table involved in the join. For example, a three-way join for tables that use the native `ROWID` as a row identifier, adds 54 bytes to the cache for each result row.

You can affect the performance of a query by controlling the size of the cache. As queries generate different results, they benefit from different cache sizes. Generally, the larger the cache, the faster the performance. However, you must balance cache size against other memory requirements for your system. Consider also that continually adjusting cache size in an application might decrease performance, as each adjustment requires the DataServer to make several calls to the OCI.

To determine the optimal cache size for a query, experiment with different values for `CACHE-SIZE` and use `DEBUG EXTENDED` to generate cursor statistics in the `dataserv.lg` file that you can examine. Aim for minimal cursor activity. You might also want to lower the cache size for queries that typically fetch only a row or two. This makes memory available for other, more productive uses.

The following statement is an example of setting an optimal cache size for a particular query against the Sports database:

```
FOR EACH customer, EACH order OF customer WHERE order.ordernum > 20
  QUERY-TUNING(CACHE-SIZE 20 ROW DEBUG EXTENDED):
```

Oracle hints

The DataServer issues hints to Oracle in the following cases:

- For deletions and updates.
- For queries that use the `USE-INDEX` phrase if the DataServer determines that a hint would ensure that the order of the report is consistent with ABL. The index that you specify in the `USE-INDEX` phrase must have

a FOREIGN_NAME. That is, it must be an index defined in the Oracle database and in the schema holder. It cannot be a field that you define as an index in the schema holder only, nor can it be a function-based index because function-based index definitions do not get described to the schema holder.

Note: When a query specifies the EXCLUSIVE-LOCK condition, the hint is applied to the first SQL request executed against Oracle for the ABL query which is responsible for producing a key cache of the result set. Subsequent SQL requests, that position the query on individual records to be locked, do not apply the index hint.

- For queries that use the native Oracle ROWID. Note that you must specify that a table use the native ROWID in the schema holder using the Data Dictionary.

If you create your Oracle database using the OpenEdge DB-to-Oracle migration utility and choose the **Create Progress RECID** option, your tables must use the PROGRESS_RECID column instead of the native ROWID or your applications will not benefit from this performance enhancement. The combination of this enhancement and using the native Oracle ROWID results in performance gains when your application holds exclusive locks or upgrades locks.

In general, using the native ROWID tends to help performance, though you lose the following functionality:

- Support for FIND PREV and FIND LAST statements
- Ability of FIND statements to reposition each other
- Support for the RECID function, although you can still use the OpenEdge ROWID function

Using field lists when updating records

You can instruct the DataServer to use field lists when your application needs to update records. The DataServer uses the technique of optimistic updates to allow you to retrieve and update only the fields you need. Oracle handles the record-locking when the update occurs. Activate this feature by specifying -Dsrv optimistic when you start the DataServer.

Note: Optimistic updates are not allowed for LONG, RAW, and LONG RAW columns.

Typically, an application has to obtain a record with an EXCLUSIVE-LOCK (either by explicitly specifying the EXCLUSIVE-LOCK or by a SHARE-LOCK that is upgraded to EXCLUSIVE-LOCK). Optimistic updates allow changes to be made to records that you retrieve NO-LOCK. Since field lists require that you obtain records with NO-LOCK, you can use field lists combined with optimistic updates to perform updates without retrieving the entire record. For example, the following code is acceptable if you specify the -Dsrv optimistic startup parameter:

```
FOR EACH customer FIELDS (cust_num name) NO-LOCK:
  UPDATE name.
```

The DataServer generates SQL similar to the following:

```
SELECT cust_num, name FROM sports.customer
UPDATE sports.customer SET name=:x1 WHERE cust_num=:rid AND name=:o1
```

The bind variable :x1 represents the new value for the name column and :o1 supplies the old value. The clause, WHERE cust_num=:rid, specifies which row to update (in this example cust_num supports the OpenEdge ROWID function). The name=:o1 portion of the WHERE clause prevents the UPDATE from taking place if another client has changed the name column while your client was holding it NO-LOCK.

The DataServer instructs Oracle to compare the old value of name to its present value. If the values are the same (indicating that no one changed the record while your client held the record NO-LOCK), Oracle updates the field. This feature enhances performance by reducing concurrency problems resulting from locks held for long periods and by reducing network traffic, as you can send only those fields you want to update.

Caution: Using optimistic locking can cause the application to appear to hang. If one user already has a record locked prior to a second user executing ABL, as previously shown with optimistic locking, the second user's application will appear to hang waiting for the first user to release the record. The standard table in use message is not displayed.

Setting the data-source result set retrieval option

The DataServer provides a -Dsrv option, PRGRS_QUOTE_PROCTEXT, to specify how the proc-text-buffer pseudo table must access result sets in which each row is a long character string, that is, it specifies how to use proc-text-buffer for row results. For more information on proc-text-buffer and proc-text-buffer-q (with **quoted text**), see Using the OpenEdge-supplied proc-text-buffer for row results on page 94.

You can activate or deactivate the quoting behavior using the PRGRS_QUOTE_PROCTEXT option. The default value of this option is 0 (without **quoted text**). You can activate it by setting PRGRS_QUOTE_PROCTEXT option to 1.

Join by SQL DB

For queries that include joins issued in FOR EACH and OPEN QUERY statements, the DataServer evaluates the queries and, in some cases, instructs the Oracle DBMS to perform the joins, thereby improving performance. However, when Oracle performs a join, you receive results in an order consistent with Oracle, not with ABL. To get results that are consistent with ABL, turn off join by SQL DB with the QUERY-TUNING phrase at the query level or with the -nojoinbysqldb startup parameter.

In DataServers, JOIN-BY-SQLDB is the default behavior for outer join operations.

For each join, the DataServer evaluates whether it is possible to have the Oracle RDBMS perform it and estimates whether doing so improves performance. The DataServer uses the following criteria to determine whether a join by SQL DB is possible:

- All tables in the join are in the same logical OpenEdge database, that is, they are contained in the same DataServer schema. The tables can be in distributed Oracle databases as long as they are represented in a single DataServer schema.
- Every table, except the innermost one, has a unique record identifier (ROWID) or RECID support.
- There is no USING phrase for any of the inner tables. For example, join by SQL DB will not occur for this query:

```
FOR EACH customer, EACH order OF customer USING ordernum:
```

- There is no BY phrase that contains expressions or array fields.
- There is no request for an EXCLUSIVE-LOCK on any of the tables in the join.
- The join does not exceed 10 levels.

The DataServer uses the following criteria to estimate whether performing a join by the Oracle RDBMS might improve performance:

- The join uses an OF clause or WHERE clause for each of the inner table loops. For example, the following query requires a field-to-field correspondence between two tables:

```
FOR EACH customer, EACH order OF customer:
```

- The WHERE clause includes the equals operator (=) and the AND option, as in the following example:

```
FOR EACH customer, EACH order
  WHERE customer.cust_num = order.cust_num AND customer.cust_num > 100:
```

Note:

By default, dynamic, inner, and outer joins are performed by the oracle database server, provided the criteria set by the DataServer is fulfilled. For more information on joins, see *Develop SQL for OpenEdge*.

An Unknown value (?) in OpenEdge is mapped to a NULL value in MS SQL Server; When you perform a JOIN-BY-SQLDB, MS SQL does not join the records in the parent table to the records in the child table on NULL join key values. For an outer join performed with JOIN-BY-SQLDB, if the results set contains a parent record with a NULL join key value and a non-matching child record, all field values in the child record are set to NULL. In contrast, when you perform a join by client, records that are joined on the Unknown value (?) (to which NULLs correspond), the results set includes any matching parent and child records in the join results of both inner and outer joins.

By default, the DataServer instructs Oracle to perform a join when possible and when desirable. However, you can control the default behavior by using the QUERY-TUNING [NO-]JOIN-BY-SQLDB phrase or the -nojoinbysqldb startup parameter. The QUERY-TUNING phrase controls the behavior for a single query. The -nojoinbysqldb controls it at the session level. The query-level setting overrides the session-level setting. The following table describes how these controls interact and affect the behavior.

Table 23: Controlling join by SQLDB

QUERY-TUNING	Startup parameter	Behavior
JOIN-BY-SQLDB	-nojoinbysqldb	Oracle performs the join if possible
JOIN-BY-SQLDB	None	Oracle performs the join if possible
NO-JOIN-BY-SQLDB	-nojoinbysqldb	The client performs the join
NO-JOIN-BY-SQLDB	None	The client performs the join

QUERY-TUNING	Startup parameter	Behavior
None	-nojoinbysqldb	The client performs the join
None	None	Oracle performs the join if possible and if the join is a true join

Join by SQL DB does not occur by default for the following query:

```
FOR EACH customer, EACH order:
```

You receive a warning if you specify JOIN-BY-SQLDB when it is impossible to have Oracle perform the join, and the DataServer performs the join instead. You receive a warning at compile time if you specify JOIN-BY-SQLDB when it is not optimal to have Oracle perform the join.

Improving join performance for queries with REVERSE-FROM option

If a join does not perform as well as you expect, try using the REVERSE-FROM option for the QUERY-TUNING phrase. The REVERSE-FROM option causes the DataServer to generate an SQL FROM clause that lists the tables in reverse order. In some cases, reversing the order in which tables are joined might improve Oracle DBMS performance.

Improving join performance for queries with FIRST or LAST criteria

In versions prior to OpenEdge 11.3, processing an ABL query with multiple tables using the ABL FIRST and LAST keywords was significantly slow compared to processing a similar ABL query against a single table or against a join query that did not use FIRST or LAST criteria. The performance degradation was because the server join was unable to reduce the FIRST or LAST criteria to a single server join record during ABL to SQL translation. This widened the result set and forced client selection, and/or the client was forced to retrieve server records key by key using multiple queries.

The join performance has been improved by ensuring that all joins with FIRST or LAST criteria translate ABL to a single server join SQL statement, thus maximizing server optimization, minimizing client selection, and reducing round trips to the foreign data source.

The ABL uses an index to evaluate the “FIRST” or “LAST” criteria. By default, the primary index is selected in determining the “MIN” or “MAX” column criteria to be applied in the server sub-query that handles the FIRST or LAST join criteria. When USE-INDEX clause is used, the index specified in the USE-INDEX clause will be used in place of the default primary index. Please visit Database index key widths section to view examples that illustrate the behavior of the index definition if the single-shot query mechanism is used.

Note:

Single-shot Server side Join selection is excluded for the scenarios listed below:

- When the ABL queries have the BY clause in it.
- When the ABL queries have multiple single shot statements.
- When the ABL queries have OUTER-JOIN.

Skipping schema verification

When r-code runs each time a table, view, or buffer is opened, the DataServer checks the data definitions of the Oracle database to make sure they match the schema definitions in the schema holder. If they do not match, the DataServer returns an error.

Unmatched definitions can result in the corruption of your Oracle database. However, verifying the definitions is time consuming in a production scenario. In a production environment, you might consider using the `-Dsrv skip-schema-check` startup parameter to increase performance, but only when you are certain that the data definitions in the Oracle database, at least for the tables you are accessing, have not changed.

Caution: If you use this parameter, and the DataServer skips the schema check, it will not detect discrepancies between the schema in the schema holder and the data definitions in the Oracle database. If it continues to process queries, inserts, and deletions, your Oracle database might become corrupted.

Even if you specify the `skip-schema-check` option, the DataServer does not skip the schema check if one of these cases is true:

- The schema holder does not have enough information on the table being queried to determine the maximum size of a column. In Version 8, the DataServer did not store information in the schema holder on maximum sizes of `CHARACTER` and `RAW` columns, and a migrated schema holder might not have this information. For example, there might not be maximum size information if you migrated a Version 8 schema holder that you dumped and loaded into a Version 9 schema holder and then forward to an OpenEdge 10 schema holder. To make sure that the schema holder has the required maximum size information, use the **Update/Add Table Definitions** utility, found on the OpenEdge **Oracle Utilities** > **DataServer** > **Data Administration**menu, to update the schema holder after using the dump and load utilities.
- A view does not have a field selected to serve as a record identifier (`PROGRESS_RECID`).

The `dataserv.lg` log file notes when the DataServer skips the schema check.

Measure carefully the performance benefit against the risk to your database before deciding to use `-Dsrv skip-schema-check`.

Writing queries for performance

This section provides a collection of tips and guidelines to follow when writing queries. For example, a query that processes a large number of rows performs best if it uses `NO-LOCK`, lookahead cursors, a large cache size, and a small field list. The following suggestions might help improve the performance of your DataServer applications. Try some of the following and use the `DEBUG` diagnostic options to gather statistics on how your application runs:

- Use `FOR EACH`, `GET`, and `OPEN QUERY` statements as opposed to `FIND` statements, which generally perform more slowly. Consider using the `FOR FIRST` statement instead of `FIND FIRST`. The only exception is that `FIND LAST` is faster than `GET LAST` because `GET LAST` causes the client to process all the records. The `FIND LAST` statement allows the server to retrieve the last record.
- Take advantage of field lists.
- Take advantage of the `QUERY-TUNING` options.
- Use lookahead cursors.
- Use `NO-LOCK` where possible.

- Avoid specifying lock upgrades. Allow the DataServer and Oracle to handle lock upgrades.
- Do not ask for a particular ordering of results with `USE-INDEX` or `BY` clauses, unless your application requires it. Allow the DataServer and Oracle to determine which index (if any) is most efficient for processing a query and avoid the overhead of sorting results.
- For aggregates, use the `RUN-STORED-PROC send-sql-statement` syntax or use an OpenEdge SQL statement. If you use an OpenEdge SQL statement with a cursor, declare the cursor read-only.
- If you are testing for the existence of a record, use the `CAN-FIND` function, which does not retrieve the record if the DataServer passes the entire `WHERE` clause to Oracle for processing. However, avoid nesting `CAN-FIND` functions.
- Avoid using the `RECID` function. Use the `ROWID` function.

See Analyzing application execution with Enhanced Logger on page 182 for information on collecting statistics.

Bulk-insert with ABL

Bulk-insert is used for creating bulk records by means of a programmatic syntax. The DataServer client component caches the record(s), and then send it to the Oracle database once the cache is full. The following statements can be used with `send-sql-statement` when you want records to be created in Bulk-insert mode:

- `("--Bulk-insert Start <n>”)`: This statement is used to begin the Bulk-insert process, where `<n>` is optional.
- `CREATE`
- `("--Bulk-insert End")`: This statement is used to end the Bulk-insert process.

For example:

```
RUN STORED-PROC send-sql-statement h1 = PROC-HANDLE ("--Bulk-insert Start
        10").
CLOSE STORED-PROC send-sql-statement r1 = PROC-STATUS WHERE PROC-HANDLE = h1.
  DO I = 1001 to 1040:
  CREATE customer.
  ASSIGN cust-num = i.
  ASSIGN NAME = "Bulk cust " + STRING (i).
  IF I > 1020 THEN
  comments = "asjkasdjkakjdjaksdjaks jkasdjk".
  IF I = 1035 THEN
  comments = FILL("asdf ", 1000).
  END.
  RELEASE customer. /* Flush out the last one before --Bulk-insert End */
RUN STORED-PROC send-sql-statement h2= PROC-HANDLE ("--Bulk-insert End").
CLOSE STORED-PROC send-sql-statement r2 = PROC-STATUS WHERE PROC-HANDLE = h2.
DISPLAY ERROR-STATUS:ERROR.
DISPLAY ERROR-STATUS:GET-MESSAGE(1) FORMAT "x(60)" ERROR-STATUS:GET-NUMBER(1).
```

The value after `Bulk-insert Start` is the number of records after which records will be created on **Oracle** Database. In case, the number of records is not specified, Bulk-insert will fill up a buffer that ABL Client uses for keeping the records. When the buffer is filled up, records will be created. In that case, `Bulk-insert End` will just write the remaining records in the buffer that haven’t been created.

Batch size is minimum of number you specify on `Bulk-insert Start` or the number of records that can be put it in the bulk insert cache(which is 30k).

Note: Whenever there is a LOB operation inside Bulk Insert then, Bulk Insert will be downgraded to single record insert.

Closing inactive cursors

In a use case where the OpenEdge application faces the possibility of exceeding the limit on available cursors, it is recommended that you increase the limit of the OpenEdge index cursor (`-c`) parameter. For more information on OpenEdge Index cursors, see Index cursors on page 197. Note that the maximum value you can specify for the `-c` parameter depends on the maximum value you specified for the `OPEN_CURSORS` parameter in the `init.ora` file.

If there is a chance that you will exceed the maximum number of cursors specified by the `-c` parameter, then you must close inactive cursors using the RDBMS stored procedure statement, RUN STORED-PROC ds-close-cursor. For more information on the cursor closer operation, see RUN STORED-PROC statement with special internal option on page 93.

Query tuning

In addition to the standard approach of using selection criteria to refine access to data, you can further influence how the DataServer executes a query through the ABL `QUERY-TUNING` phrase. How you structure a query determines how efficiently you access a database. Efficient use of the Oracle RDBMS enhances the performance of DataServer applications. You can include the `QUERY-TUNING` phrase for the following ABL statements:

- `FOR EACH`

```
FOR EACH table QUERY-TUNING
  ( query-tuning-optionquery-tuning-option . . . )
```

- `OPEN QUERY`

```
OPEN QUERY query QUERY-TUNING
  ( query-tuning-optionquery-tuning-option . . . )
```

- `DO PRESELECT`

```
DO PRESELECT table QUERY-TUNING
  ( query-tuning-optionquery-tuning-option . . . )
```

- REPEAT PRESELECT

```
REPEAT PRESELECT table QUERY-TUNING
  ( query-tuning-optionquery-tuning-option . . . )
```

Place the QUERY-TUNING phrase after the last record phrase. For example, place it near the end of the statement where you also place block modifier phrases such as BREAK, ON ERROR, and TRANSACTION. Separate multiple query-tuning options by a single space. The QUERY-TUNING options have equivalent startup parameters. You cannot use the startup parameters to override the QUERY-TUNING settings.

The following table describes the query-tuning options.

Table 24: Query-tuning options

Option	Description
ARRAY-MESSAGE NO-ARRAY-MESSAGE	Specifies whether the DataServer sends multiple result rows in a single logical network message, thereby reducing network traffic. **Default**: ARRAY-MESSAGE, if the query uses a lookahead cursor.
BIND-WHERE NO-BIND-WHERE	Specifies whether the DataServer uses Oracle bind variables for values in WHERE clauses. Using bind variables typically improves performance, but Oracle provides unexpected results for some operations, such as a MATCHES on an indexed field and a trailing wild card or comparisons of CHAR fields that use Oracle's blank-padding rules. Specify NO-BIND-WHERE to use literals. Using NO-BIND-WHERE in queries that do comparisons (MATCHES, BEGINS on an indexed field) can improve performance. **Default**: BIND-WHERE.
CACHE-SIZE *integer* BYTE CACHE-SIZE *integer* ROW	Specifies the size of the cache for information (in bytes or records) used by lookahead or standard cursors. If you have two ABL statements that cause the DataServer to generate identical SQL code except that the second statement specifies a smaller cache size, the DataServer reuses the larger cache from the first statement if the cursor is still available. Reusing cache and cursors improves performance. Byte maximum: 65535 bytes. Byte minimum: Specify the number of bytes contained in a single record. For joins, specify the number of bytes contained in two joined records. By default, the DataServer sizes the cache to accommodate one record or, for a join, two joined records. For example, if a join returns a 500-byte record, you need a cache of at least 1000 bytes. **Default**: 1024 bytes with standard cursors; 8192 with lookahead cursors. Row maximum: the number of records that can fit in 65535 bytes. See Caching records on page 121 for more information.Row minimum: 1 for a single table; 2 for a join.

Option	Description
DEBUG EXTENDED DEBUG SQL NO-DEBUG	Specifies whether the DataServer should print debugging information that it generates for the query to the dataserv.lg file. Specify DEBUG SQL to print the SQL that the DataServer executes against the Oracle DBMS. Specify DEBUG EXTENDED to print additional information, such as cursor statistics. There are additional options for collecting advanced statistics with DEBUG. See Analyzing application execution with Enhanced Logger on page 182 for more information. **Default**: NO-DEBUG.
HINT string1 string2 string3	Specifies the Oracle hint syntax that the DataServer passes directly to the Oracle DBMS as part of the query. This allows you to control which hints are passed as opposed to the index hints that the DataServer passes when appropriate. When you have to specify an index name in the hint syntax, use the name defined in the Oracle database. Because the DataServer generates aliases for Oracle tables using names from T0 through T9, use these aliases to refer to tables in the hint syntax. The DataServer passes the opening symbols (/*+) and closing symbols (*/). For example, to pass the /*+ORDERED*/ hint syntax, you specify only HINT "ORDERED". Passing incorrect hint syntax, inappropriate hints, or conflicting hints will not return an error but might give you unpredictable results. See your Oracle documentation for information on hint syntax.
INDEX-HINT NO-INDEX-HINT	Specifies whether the DataServer should provide index hints to the Oracle DBMS. Generally, index hints improve performance, but Oracle's responses to hints vary between releases. Specify NO-INDEX-HINT to test whether performance for a query improves when the DataServer executes it without hints. See Indexes on page 41 for more information on index hints. By default, the DataServer passes index hints. You can turn off the default globally at compile time or run time by specifying the -noindexhint startup parameter when you start an OpenEdge session. Use INDEX-HINT to retain the behavior for individual queries. **Default**: INDEX-HINT.

Option	Description
JOIN-BY-SQLDB NO-JOIN-BY-SQLDB	Specifies whether the DataServer allows the Oracle DBMS to perform a join, which usually improves performance. **Default**: JOIN-BY-SQLDB.You can turn off the JOIN-BY-SQLDB default globally at compile time by specifying the -nojoinbysqldb startup parameter when you start an OpenEdge session. The -nojoinbysqldb parameter does not override the explicit use of JOIN-BY-SQLDB in the QUERY-TUNING phrase. **Note:** JOIN-BY-SQLDB is the default behavior for all outer join operations.
LOOKAHEAD NO-LOOKAHEAD	Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as fit in the allocated cache (CACHE-SIZE), which limits the number of database accesses, thereby improving performance. Using lookahead cursors results in behavior that is different from ABL because the client does not see any changes made to the records in the cache. Specify NO-LOOKAHEAD for behavior that is consistent with ABL. **Default**: LOOKAHEAD, except with FIND statements and statements that use an EXCLUSIVE lock.
NO-QUERY-ORDER-ADDED	Specifies that OpenEdge should not choose an index in the absence of a USE-INDEX or BY clause in the query request. OpenEdge may otherwise select an index if it is needed to provide ABL language compatibility. **Note:** If you elect to use this option to omit index selection on the query, you might see better performance using the optimizer's sort selections. However, compatibility with OpenEdge forward/backward scrolling and reposition capability might be lost. Only use this option when compatibility is not required and can be overlooked for the sake of better performance.

Option	Description
NO-QUERY-UNIQUE-ADDED	Specifies that OpenEdge should omit the record identifier from the end of the query's generated ORDER BY clause, or a join query's join crieteria, when trying to obtain record uniqueness from a selected non-unique index. A sort order that is modified to derive uniqueness may produce a query that can't find a useful index to perform sorting thus impacting query performance. **Note:** If you elect to use this option, the query may find an index match to provide better performance. However, turning off uniqueness in a query where scrolling is required may result in behavior that is incompatible with the OpenEdge ABL. Only use this option when compatibility is not required and can be overlooked for the sake for better performance.
ORDERED-JOIN	Specifies that the DataServer embed the ORDERED hint syntax in the SQL it generates.
REVERSE-FROM	Specifies that Oracle join tables in the reverse order in which they appear in the FROM clause. The DataServer generates a new SQL FROM clause with the tables in reverse order. REVERSE-FROM is independent of the ORDERED-JOIN option.
SELECT-ON-JOIN JOIN-ON-SELECT	When performing an outer join operation, SELECT-ON-JOIN specifies that any search condition separate from the join predicate be bracketed **after** the join is performed. This option eliminates all the records with NULL values contained in the non-matching results, if the columns with the NULL values appear in the join search criteria. When performing an outer join operation, JOIN-ON-SELECT specifies that any search condition separate from the join predicate be bracketed **before** the join is performed. This option includes all the records with NULL values contained in the non-matching results and causes records that join on NULL to be included in the results of an OUTER-JOIN which is consistent with OpenEdge join handling of the unknown value. **Default**: JOIN-ON-SELECT.

Note: All of the query-tuning options take effect at both compile and runtime except for the INDEX-HINT, JOIN-BY-SQLDB, and NO-JOIN-BY-SQLDB options, which apply only at compile time.

The following example shows how to use the QUERY-TUNING phrase to enhance performance. It includes a join which the DataServer instructs Oracle to perform by default. The QUERY-TUNING options specify that no lookahead cursors will be used. In addition, the DataServer will write an extended report on the SQL statements it executes, as shown:

```
FOR EACH customer, EACH order OF customer WHERE order.ordernum > 20
 BY custnum QUERY-TUNING(NO-LOOKAHEAD DEBUG EXTENDED) TRANSACTION:
```

This example shows how to use the QUERY-TUNING phrase to manage cache size so that the DataServer can reuse cursors and cache, thereby improving performance. The phrase also passes a hint to the Oracle optimizer to choose the cost-based approach to optimize the statement for best response time. Finally, the DEBUG EXTENDED option causes the DataServer to report on the SQL statements it executes, as shown:

```
FOR EACH customer, EACH order OF customer WHERE order.ordernum > 20
  QUERY-TUNING(CACHE-SIZE 20 ROW HINT "FIRST_ROWS" DEBUG EXTENDED)   TRANSACTION:
```

5

Configuring the DataServer

Building the OpenEdge DataServer for Oracle involves creating executables for several processes. This chapter provides step-by-step instructions for initially setting up the DataServer.

For details, see the following topics:

- DataServer modules
- Configuring Windows modules
- Configuring UNIX modules
- Configuring an international environment
- Configuring the DataServer in the Unified Broker framework
- Creating, Maintaining, and Deploying a schema holder
- Configuring a self-service DataServer agent in the OpenEdge Appserver and Progress Application Server

DataServer modules

The OpenEdge DataServer for Oracle can run in a variety of configurations. Some configurations involve a single process running on one machine. Others involve multiple processes running on different machines across multiple platforms.

To set up a DataServer configuration, determine which modules you need on which platforms. Then, set up the appropriate executables on those platforms. The following table lists the possible combinations and describes which executables you must set up on each machine. The term *local* indicates that a DataServer runs on the same machine as the client. *Remote* indicates that a DataServer runs on a different machine than the client.

Table 25: DataServer Configurations

Client	DataServer	Installing and configuring
Windows	Local Windows	Use the default OpenEdge client executable (`prowin32.exe`). This executable also allows you to use Oracle networking to access Oracle.
Windows	Remote Windows	On the client machine, use the default OpenEdge client executable (`prowin32.exe`).In the Windows host machine, use either the default broker executable (`_probrkr.exe`) or configure the broker using OpenEdge Management or OpenEdge Explorer.
Windows	Remote UNIX	On the client machine, use the default OpenEdge client executable (`prowin32.exe`).On the UNIX host machine, use the default broker executable (`_probrkr`) and set `ORASRV` to point to the `_orasrv` DataServer executable. This executable supports OpenEdge networking.
UNIX	Local UNIX	On the UNIX client machine, use the default client executable (`_progres`). This executable also allow you to use Oracle networking to access Oracle.
UNIX	Remote UNIX	On the UNIX client machine, use the default client executable (`_progres`).On the UNIX host machine, use the default broker executable (`_probrkr`) and set `ORASRV` to point to the `_orasrv` DataServer executable. These executables support OpenEdge networking.
UNIX	Remote Windows	On the UNIX client machine, use the default client executable (`_progres`).In the Windows host machine, use either the default broker executable (`_probrkr.exe`) or configure using OpenEdge Management or OpenEdge Explorer.

Note: If you are using a DataServer on a UNIX platform where Oracle does not support shared objects, you must build your DataServer executables instead of using the default client and DataServer executables. See Building DataServer Executables on page 275.

For instructions on setting up your DataServer configuration, see the section or sections that apply to the platform you are considering. For example, if you are configuring a local DataServer where all the OpenEdge modules run on the same UNIX machine, see Configuring UNIX modules on page 138. If you are building a remote DataServer configuration with a UNIX client accessing a Windows host, see Configuring UNIX modules on page 138 for instructions on setting up the client and Configuring Windows modules on page 137 for instructions on setting up the host.

Configuring Windows modules

A DataServer configuration that includes Windows can be a local configuration, where all the components run on the same machine, or a remote, client/server configuration. It can also include clients or servers running on different platforms. For example, a Windows client can access a DataServer running remotely on a UNIX host. See The following table for a list of the possible configurations for Windows. It describes what you must do to set up the client and server modules.

In a remote configuration, the following two DataServer modules run on the host even though you use only one executable:

- **Broker** — The `_probrkr` executable starts the broker, which spawns the DataServer process.
- **DataServer** — The broker spawns this process, so you do not interact with it directly. However, you must make sure that the broker uses the `_orasrv` DataServer executable.

Note: Only one broker can run in a directory at a time. If you want to run multiple brokers, run each in a separate directory.

The following sections explain how to configure and run the local DataServer and how to configure and run the DataServer modules in the Windows host machine. For instructions on setting up the client side of remote configurations:

- **On UNIX** — See Configuring UNIX modules on page 138 for information.
- **In Windows** — See Configuring the DataServer in the Windows host on page 137 for information.

Configuring the local DataServer in Windows

Before you can run the DataServer modules in the Windows client, you must set environment variables., and verify access to Oracle files.

To configure the local DataServer in a Windows client:

1. Make sure the `Oracle_HOME` and `Oracle_SID` variables are set correctly. You can set variables in the registry (Oracle sets `Oracle_SID` at installation time), at the system level or in a `.ini` file. See the following table for information on these variables.
2. If you plan to use Oracle Networking to access the Oracle database, make sure that the client executable has read access to the `tnsnames.ora` file in the directory indicated by the `TNS_ADMIN` variable in the `oracle.ini` file. Settings for your instance in the `tnsnames.ora` file will override values for `Oracle_HOME` and `Oracle_SID`.

Configuring the DataServer in the Windows host

Before you can run the DataServer modules in the Windows host, you must set environment variables. OpenEdge Management or OpenEdge Explorer utility provides a graphical interface that allows you to manage and configure the DataServer host environment. See Configuring the DataServer in the Unified Broker framework on page 142 for instructions.

If you do not choose to use OpenEdge Management or OpenEdge Explorer and want to set environment variables manually, see the table below.

To use the command line to configure the DataServer modules:

1. Open the **Proenv** window from your installation folder.
2. Set the required variables, as described in the following table.

Table 26: DataServer for Oracle environment variables

Variable	Value
`Oracle_HOME`	The pathname of the directory where you installed Oracle.
`Oracle_SID`	The identifier for the Oracle instance you are accessing.
`ORASRV`	The name of the executable (including the path) of the Oracle DataServer. If you are not using the default DataServer executable, `_orasrv.exe`, set `ORASRV` to point to the appropriate executable.
`PROSTARTUP`	The pathname of your default OpenEdge startup (`.pf`) file.To set the Message Buffer Size (`-Mm`) startup parameter for the broker to a value different from the 1924 default buffer size, you must set the `PROSTARTUP` environmentvariable equal to the path and name of the .pf file that contains the `-Mm` parameter. This approach ensures that your value is recognized and therefore used by the broker when it starts. Only Windows clients must use this technique. **Note:** If you use the command line utility and intend to set the `-Mm` startup parameter to a value other than 1024, you must use the `.pf` file with the `PROSTARTUP` variable set.

3. Start the broker. See Starting the DataServer processes on page 152 for instructions to start the DataServer processes from the command line.

   Once the broker is running, you can build the schema holder for the Oracle database. See Creating, Maintaining, and Deploying a schema holder on page 149 for instructions.

Configuring UNIX modules

A DataServer configuration that includes UNIX can be a local configuration, where all the components run on the same machine, or a remote, client/server configuration. It can also include clients or servers running on different platforms. For example, a UNIX client can access a DataServer running remotely in a Windows host.

In a client/server configuration, the following two DataServer modules run on the host:

- **Broker** — The DataServer broker on the host machine determines the type of requests coming over the network and starts the appropriate server for the client process.
- **DataServer** — The DataServer on the host machine accesses the Oracle database and communicates with the client process.

Note: Only one broker can run in a directory at a time. If you want to run multiple brokers, run each in a separate directory.

See the DataServer Configurations table for a list of the possible configurations for UNIX. It describes what you must do to set up the client and server modules.

Configuring a DataServer Module on UNIX involves setting up its environment.

Configuring the local DataServer on UNIX

To configure the local DataServer on UNIX, you set environment variables. Be sure to set the variables in the same environment (UNIX shell) from which you plan to run the DataServer. The following table describes the environment variables and how to set them for the local DataServer.

Table 27: Environment variables for the local DataServer

Variable	How to set it
DSLOGDIR	The pathname to the directory where the DataServer creates the log file, dataserv.lg, to track processes and error messages. By default, the DataServer creates this file in the current directory. (Optional)
Oracle_HOME	The pathname of the directory where Oracle is installed. (Required)
Oracle_SID	The identifier for the Oracle instance you are accessing. (Required)

If you plan to use Oracle Networking to access the Oracle database, make sure that you have configured the tnsnames.ora file as your Oracle documentation instructs. The OpenEdge client executable must have read access to the tnsnames.ora file in the directory indicated by the TNS_ADMIN variable in the oracle.ini file.

Once you have set up your environment, you are ready to create a schema holder. See Creating, Maintaining, and Deploying a schema holder on page 149 for instructions.

Configuring the DataServer on the UNIX host

To configure the remote DataServer modules on UNIX, you set environment variables on the host machines, as described in the following table. In a remote DataServer configuration, you set all necessary variables on the host machine. Be sure to set the variables in the same environment (UNIX shell) from which you plan to run the DataServer. No configuration is required on the client machine.

The only environment variable in the following table that you set on the client is PROEXE.

Table 28: Environment variables for the remote DataServer

Variable	How to set it
DSLOGDIR	The pathname to the directory where the DataServer creates the log file, dataserv.lg, to track processes and error messages. By default, the DataServer creates this file in the current directory. (Optional)
Oracle_HOME	The pathname of the directory where Oracle is installed. (Required)

Variable	How to set it
Oracle_SID	The identifier for the Oracle instance you are accessing. (Required)
ORASRV	The name of the executable (including the path) of the Oracle DataServer. Set ORASRV to point to the appropriate DataServer executable. (Required)
PROBRKR	The pathname to the executable of the broker. (Optional)
PROSTARTUP	The pathname of your default OpenEdge startup file. To set the Message Buffer Size (-Mm) startup parameter for the broker to a value different from the 1024 default buffer size, you must set the PROSTARTUP environment variable equal to the path and name of the .pf file that contains the -Mm parameter. This approach ensures that your value is recognized and therefore used by the broker when it starts. Only Windows clients must use this technique. **Note:** If you use the command line utility and intend to set the -Mm startup parameter to a value other than 1024, you must use the .pf file with the PROSTARTUP variable set.

Once you have set up your environment, you are ready to create a schema holder. See the Creating, Maintaining, and Deploying a schema holder on page 149 for instructions.

Configuring an international environment

If you plan to access an Oracle database that uses a Unicode double-byte character set or that relies on non-Western European languages, you must configure your DataServer and your Oracle installation so that they run with the appropriate locale settings. For more information on DataServer for Oracle's Unicode implementation, see Support for Unicode on page 39. For more information on the capabilities OpenEdge has for internationalized applications, see *Internationalize ABL Applications*.

Note: By default, the OpenEdge DataServer for Oracle is double-byte enabled (DBE) and Unicode-enabled. Use or build the standard executables.

The following examples describe a Unicode or double-byte environment, but the instructions apply to other locales as well.

To configure your international environment:

1. Make sure that your installation of Oracle has been configured properly for the language required. See your Oracle documentation for details.
2. After setting the environment variables required for a standard DataServer configuration, set the DLCDB environment variable to the appropriate $DLC/prolang/ subdirectory. Setting DLCDB ensures that empty OpenEdge databases you create are appropriate for the locale.

For example, running the following command on UNIX ensures that empty databases support Japanese using the EUCJIS code page:

```
DLCDB=$DLC/prolang/jpn/eucjis ; export DLCDB
```

Note: When you set DLCDB to a prolang/ subdirectory, you cannot create a copy of the Sports database. Before copying the Sports database, unset DLCDB.

3. Create a schema holder. See Creating, Maintaining, and Deploying a schema holder on page 149 for instructions. Make sure that the code page of the empty database matches the code page that the OpenEdge client uses. Setting DLCDB addresses this if you are working in a single-language environment.
4. When you create the schema holder, make sure that the code page associated with the schema image matches the code page used by the Oracle database. (The default is ibm850, which is not appropriate for Unicode or double-byte enabled applications or for applications running in many non-Western European locales.) Specify the OpenEdge name for the equivalent Oracle code page in the **Code Page** field. For a complete list of code pages and collation names supported by OpenEdge, see the README file in the $DLC/prolang directory.
5. Run your OpenEdge application.

Note: You must specify the Internal Code Page (-cpinternal) and Stream Code Page (-cpstream) parameters when you start the OpenEdge client. The values that you specify for these parameters must match the code page that the Oracle database uses. For example, if your Oracle database uses EUCJIS, specify EUCJIS for -cpinternal and -cpstream.

In Windows, run this command to run a procedure against a local instance of Oracle:

```
prowin32 -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -U userid -P password
  -p procedure-name
```

In Windows, run this command to connect to and run a procedure in batch mode against a remote instance of Oracle through Oracle Networking:

```
_prowin32 -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -U userid@service-name
  -P password -b -p procedure-name
```

On UNIX, run this command to connect to and run a procedure against a local instance of Oracle:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -1 -db oracle-dbname
  -ld oracle-logical-dbname -U userid -P password
  -p procedure-name
```

On UNIX, run this command to connect to and run a procedure against a remote instance of Oracle through Oracle Networking:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -1 -db oracle-dbname
  -ld oracle-logical-dbname -U userid@service-name
  -P password -p procedure-name
```

On UNIX, run this command to connect to and run a procedure in batch mode against a remote instance of Oracle through OpenEdge networking after starting the broker on the host machine:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -S service-name -H host
-U userid -P password -b -p procedure-name
```

Configuring the DataServer in the Unified Broker framework

Integrating a DataServer into the Unified Broker framework affects how you configure the remote DataServer broker. It does not affect configuring the client; client configuration is the same as in a standalone DataServer architecture, although connection parameters differ. This section provides instructions on configuring the DataServer for Oracle. For instructions on configuring the other products that the Unified Broker framework supports, see *OpenEdge Application Server: Developing AppServer Applications* and *OpenEdge Application Server: Developing WebSpeed Applications*.

Configuring the Unified Broker framework in Windows

In Windows, you use OpenEdge Management or OpenEdge Explorer tool to configure the OpenEdge DataServer for Oracle in the Unified Broker framework. The Explorer has a default broker already defined, orabroker1.

To define and configure the OpenEdge DataServer for Oracle:

1. Make sure that the OpenEdge AdminServer is running.
2. Start the OpenEdge Management or OpenEdge Explorer.
3. Connect to the local/remote AdminServer.
4. From the left pane of OpenEdge Management or OpenEdge Explorer, select the **Oracle DataServer** folder and double-click. The list of existing DataServer brokers for Oracle appears in the right pane.
5. Select the DataServer instance whose properties you want to create or edit, and right-click. A pop-up menu appears.

Note: The OpenEdge DataServer for Oracle installation provides one predefined DataServer Broker (`orabroker1`) and one predefined NameServer (`NS1`). You can use these predefined components as a starting point for creating and configuring additional DataServer Brokers, and, if needed, NameServers. Each broker is referred to as an instance. Progress Software Corporation recommends using a NameServer configuration at all times. In such cases, the DataServer client's initial connection is to the NameServer. However, you can alternatively connect the DataServer directly to the broker instance by setting the `-DataService` value to none in the connection parameters of your schema holder or by adding the `-DirectConnect` parameter which will override your `-DataService` parameter. If you will always bypass the controlling NameServer, you should remove the controlling NameServer from your broker instance definition. See Starting the DataServer in the Unified Broker framework on page 167 for more information about connecting the DataServer to the NameServer and the Broker.

6. Choose the **Properties** option from the pop-up menu. The **Properties** dialog box for that instance appears.
7. Configure the DataServer Broker by setting general DataServer properties, owner information properties, logging settings, environment variables, and others. For descriptions of relevant parameters to be set in the ubroker.properties file for DataServers, see Configuring the Unified Broker framework on UNIX on page 143 and Establishing the SSL protocol in a DataServer broker instance of the Unified Broker on page 147.

   By default in the Property Editor Server category, the pathname of the DataServer executable that the broker runs is `@{Startup\DLC\}bin\_orasrv.exe`.

   It is not recommended that you simultaneously run some DataServers for Oracle under brokers with controlling NameServers and others directly under brokers (that is, without controlling NameServers). This defeats the purpose of using a NameServer to control brokers. If you do this, the benefits of the NameServer are lost and load balancing is ineffective. Progress Software Corporation recommends that you always use a NameServer, with one exception: you can choose initially to connect directly to a broker to simplify confirming an initial connection. Once you establish a connection, Progress Software Corporation recommends that you reintroduce the NameServer into your configuration.

   For more information on using OpenEdge Management or OpenEdge Explorer and details on the required settings, see the OpenEdge Management or OpenEdge Explorer online help.

Configuring the Unified Broker framework on UNIX

On UNIX, you configure the Unified Broker framework by editing the `$DLC/properties/ubroker.properties` file. In this file, you provide configuration information for the AppServer, WebSpeed, and DataServers. For general information on how to work with the `ubroker.properties` file, see *Configure OpenEdge*.

When you configure the OpenEdge DataServer for Oracle, you are providing information that the Unified Broker framework can use to make sure that it is running the correct DataServer broker instance, that the broker is spawning the appropriate DataServer process, and that broker and servers are running in the correct environment.

The following table describes the sections in the `ubroker.properties` file that apply to the OpenEdge DataServer for Oracle. The file configures a default broker, `orabroker1`, which you can use as is or use as a template for your own configuration specifications.

Table 29: OpenEdge DataServer for Oracle sections

Section	Description
[UBroker.OR]	Parent entity of the OpenEdge DataServer for Oracle brokers. It defines default property settings for all OpenEdge DataServer for Oracle Broker instances.
[UBroker.OR.orabroker1]	Sample property entity of an OpenEdge DataServer for Oracle Broker instance. It defines property settings for the OpenEdge DataServer for Oracle Broker named orabroker1. Note that although many of the settings in this section might work in your environment, some of the settings are for demonstration purposes only. You must specify the appropriate settings for: • **srvrExecFile** — Specify the pathname of DataServer executable you want to use. The default is $DLC/bin/_orasrv. • **srvrStartupParam**— In general terms, you should not modify the parameters in this section. Note that the values assigned to the parameters are for the broker's internal use only. For example, the values of the code-page parameters do not correspond to the values that your client connection might require. You can add startup parameters, however. One parameter you may wish to add, for instance is the –ssl parameter in order to make your DataServer instance SSL enabled. **Note:** For more information on setting the -ssl parameter in the srvrStartupParam value, see Establishing the SSL protocol in a DataServer broker instance of the Unified Broker on page 147.
[Environment.orabroker1]	Environment settings that apply to the DataServer for Oracle Broker named orabroker1. Be sure to set the variables to the appropriate values; the values in the file are for demonstration purposes only. ORACLE_HOME and ORACLE_SID are required when not using SQL*NET to connect to Oracle. Add any environment variables that you want to apply to the DataServer's environment to this section. Environment variables defined here will be made available to the AdminService and broker at run-time and will override any values that may have been set previously (such as in the Windows registry or in a UNIX shell script).

Here are the OpenEdge DataServer for Oracle sections in the ubroker.properties file:

```
#
#  Default properties for broker instances serving Oracle DataServers
#
#
 #  Default properties for broker instances serving Oracle DataServers
 #
 [UBroker.OR]
     srvrExecFile=$DLC/bin/_orasrv
     srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
```

```
     operatingMode=State-aware
     classMain=com.progress.ubroker.broker.ubroker
     portNumber=4445
     defaultService=0
     initialSrvrInstance=0
     minSrvrInstance=0
     maxSrvrInstance=256
     brkrLoggingLevel=3
     description=Oracle DataServer Broker
#
# Sample Oracle DataServer Broker definition
#[UBroker.OR.orabroker1]
     srvrExecFile=$DLC/bin/_orasrv
     srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
     srvrLogFile=$WRKDIR/orabroker1.server.log
     brokerLogFile=$WRKDIR/orabroker1.broker.log
     portNumber=4445
     defaultService=1
     appserviceNameList=orabroker1
     controllingNameServer=NS1
     environment=orabroker1
     uuid=172.18.103.53:1f415c:d6330e5d24:-7f4d
     description=A sample Oracle DataServer Broker

##
Environment for Oracle Dataserver Broker: orabroker1
##
Connection to an Oracle database through the Progress
# dataserver requires that environment variables for
# ORACLE_HOME and ORACLE_SID are set in the context of
# a running dataserver broker instance. These values
# must either be set in the shell from which the Admin
# Service is started or they must be explicitly set here
#
for the broker instance.
#[Environment.orabroker1]
# ORACLE_HOME=/u01/oracle/client
# ORACLE_SID=OR

#
# Sample Oracle DataServer Broker definition using SSL
#[UBroker.OR.oraSSLbroker1]
     srvrExecFile=$DLC/bin/_orasrv
     srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
     -ssl
     srvrLogFile=$WRKDIR/orasslbroker1.server.log
     brokerLogFile=$WRKDIR/orasslbroker1.broker.log
     portNumber=4446
     defaultService=0
     appserviceNameList=orasslbroker1
     controllingNameServer=NS1
     environment=orasslbroker1
     uuid=172.18.103.53:1f415c:d6330e5d24:-7f4d
     description=A sample Oracle DataServer SSL Broker

#
# Environment for Oracle Dataserver Broker: orasslbroker1
#
# Connection to an Oracle database through the Progress
# dataserver requires that environment variables for
# ORACLE_HOME and ORACLE_SID are set in the context of
# a running dataserver broker instance.  These values
# must either be set in the shell from which the Admin
# Service is started or they must be explicitly set here
#
 for the broker instance.
#[Environment.orasslbroker1]
#   ORACLE_HOME=/u01/oracle/client
#   ORACLE_SID=ORCL
```

For a complete description of the parameters included in each of these sections, see the comments in the `$DLC/properties/ubroker.properties` file. Be sure to preserve the original `$DLC/properties/ubroker.properties` file. Work with a copy of the file. You must name the copy of the file `ubroker.properties`. After creating your own version of `ubroker.properties`, you can verify its settings by using the `oraconfig` utility. See DataServer Command Line Utilities and Startup Parameters on page 283 for more information.

The `ubroker.properties` file is read on startup of the AdminServer process. For changes in any used environment variables to take effect, you must restart the AdminServer.

Validating ubroker.properties content

Whenever you create your own version of the `ubroker.properties` file, you should use the relevant validation utility to validate your changes and make sure that there are no syntax errors or conflicts. When configuring the DataServer for Oracle, you validate the file by using the `oraconfig` utility. For information about the ORACONFIG utility, see DataServer Command Line Utilities and Startup Parameters on page 283 for more information.

Configuring multiple brokers

You can configure multiple brokers by adding more `[UBroker.OR.`*`broker-name`*`]` and `[Environment.`*`broker-name`*`]` sections. Each broker instance must have a unique name. The brokers can share the properties that you define in the parent entity `[UBroker.OR]` section.

If you want to access multiple Oracle databases, you must configure one broker per database, and each broker must run in a separate directory and a distinct environment. For example, each broker must have a unique setting for `ORACLE_SID`.

Using the NameServer

By default, your DataServer for Oracle broker instances are defined with a controlling NameServer and are provided with a default Data Service. Progress Software Corporation recommends using a NameServer configuration at all times. In such cases, the DataServer client's initial connection is to the NameServer. However, you can alternatively connect the DataServer directly to the broker instance by setting the `-DataService` value to **none** in the connection parameters of your schema holder. If you will always use a `-DataService` value of **none**, you should remove the controlling NameServer from your broker instance definition. See *Configure OpenEdge* for more information about the NameServer's role in a configuration.

Note: Do not simultaneously run some DataServers for Oracle under brokers with controlling NameServers and others directly under brokers (that is, without controlling NameServers). This defeats the purpose of using a NameServer to control brokers. If you do this, the benefits of the NameServer are lost and load balancing is ineffective. Progress Software Corporation recommends that you always use a NameServer, with one exception: you can choose initially to connect directly to a broker to simplify confirming an initial connection. Once you establish a connection, Progress Software Corporation recommends that you reintroduce the NameServer into your configuration.

Establishing the SSL protocol in a DataServer broker instance of the Unified Broker

The following set of SSL properties are used to establish SSL protocol communication using OpenEdge networking from a unified broker configuration using the ubroker.properties file:

Table 30: SSL properties used to establish SSL protocol communication

Property	Default Value	Description
sslEnable	0	Enables SSL-secure connections to a DataServer broker instance from remote DataServer clients. To enable SSL security for incoming client connections, set the property value to *1*. While the sslEnable switch enables the infrastructure needed to make SSL connections, it does not initiate the start of an SSL connection dialogue.To start the dialog, the srvrStartupParam must be appended with the "-ssl" startup parameter.
keyAlias	default_server	Specifies the alias set for a private/public key. If this property is not set, the server certificate alias that is provided by OpenEdge is used.
keyAliasPasswd	20333c34252a2137	Specifies the encoded password set for the alias of the private/public-key to access the server Private Key / Digital Certificate key store entry. You can use the OpenEdge provided genpassword utility for encoding your passwords
keyStorePath	$DLC/keys	Specifies the key store location.
keyStorePasswd	NA	Specifies the password used for accessing the custom keystore. This password is expected to be encoded. You can use the OpenEdge provided genpassword utility for encoding your passwords.
noSessionCache	0	0 - enable SSL session caching (stores SSL information) 1 - disable SSL session caching (no SSL information persisted)
srvrStartupParam	*-svub -S X -N TCP -U X -P X -hs 0 -s 40*	Specifies server/agent startup parameters that each process uses when the broker starts it. The default startup parameters can be appended with the "-ssl" parameter in order to initiate server/agents in SSL secure connection mode as follows:-svub -S X -N TCP -U X -P X -hs 0 -s 40 –ssl. When agents are started with the -ssl switch, they can only accept SSL-enabled client connections. This property can be dynamically updated. Dynamic changes affect only new agents that are started after this value is changed.
sslCiphers	AES128-SHA256	The SSL algorithm list to use in arbitrating an SSL session with an SSL client. Also referred as server Ciphers
certStorePath	@{Startup\DLC}\certs\	The path to the trusted certificate store. The root certificate for the certificate authority that was used to sign the server certificate should be specified by this path. Client can mention Cert Store path on the command line i.e. _progres -b -p client.p -certstorepath $RDLQA/tests/SHA2/$alg/certs

When a DataServer instance of the Unified Broker is configured for SSL, the server/agent does not attempt to auto-correct the client's connection attempt when the client connects in non-SSL mode. Instead, the connection request is rejected by the server/agent and an exception is logged. Client users must explicitly initiate their server connection in SSL mode in order to connect to an SSL-enabled DataServer broker instance of the Unified Broker.

Note: This is in contrast to the ProBroker configuration where a DataServer client connection attempted in non-SSL mode receives a return message from the SSL-enabled ProBroker that auto-corrects the connection request and retries it in SSL mode.

Creating, Maintaining, and Deploying a schema holder

The schema holder is the next component of the DataServer architecture that you have to build. The schema holder holds the data definitions of the supporting Oracle database.

To create a schema holder:

1. Start an Oracle instance for the supporting database.
2. Start the OpenEdge client. If you are using a remote DataServer, you must start both the broker and client processes.
3. Create and connect an empty OpenEdge database.
4. Create a schema holder.

The following sections describe these steps in more detail.

Schema holder security

When you use the DataServer to create a schema holder, it accesses the Oracle database. This section describes the Oracle permissions required when you create a schema holder or connect to one.

Permissions for creating a schema holder

You must have privileges to use the `SELECT` statement on certain data source objects to perform certain tasks. In this chapter, these privileges are referred to as `SELECT` privileges.

For example, when you create or update a schema holder for an Oracle database, you must be able to connect to the database and have `SELECT` privileges on specific system objects. `SELECT` privileges on system objects are required because the Data Dictionary cannot access the Data Dictionary tables in the Oracle database without them; it must access those tables to create a schema holder.

The following table describes the permissions that you need for creating a schema holder.

Table 31: Required Oracle permissions

Permission	Object
CREATE SESSION	Database
SELECT	System objects: sys.argument$ sys.ccol$ sys.cdef$ sys.col$ sys.com$ sys.con$ sys.dual sys.icol$ sys.ind$ sys.link$[1] sys.obj$ sys.procedure$ sys.seq$ sys.syn$ sys.tab$ sys.ts$ sys.user$ sys.view$ 12

Authorization and Authentication for connecting a schema holder

When you use the DataServer to create a schema image in a schema holder, it accesses the associated MS SQL Server database. This section describes the data source permissions required when you create, update, or connect to a schema image.To connect a schema holder for an Oracle database, you must provide a valid user ID and password combination for Oracle at connection time. Use the User ID (-U) parameter to provide the user ID and the Password (-P) parameter to provide the password.

Once you have set up the schema holder, the required Oracle privileges vary among users depending on their applications. For example, the user running an OpenEdge application that queries, but does not update, a table in the Oracle database must connect to the Oracle database with a user and password that provides at least SELECT privileges on the table.

Note: In addition to the permissions required by the applications that users run, users must have SELECT permission on the sys.dual system table.

In summary, the user ID and password combination required to run an application depends on:

- The Oracle database tables the application accesses
- The type of access required on those tables

Note: A database administrator (DBA) must establish all user ID and password combinations within Oracle.

---

[12] SELECT privileges are typically not provided to the Database Administrator (DBA).

Database Authentication During Foreign Database Connect

The Oracle DataServer does not require its own authentication. Users connecting through Oracle client libraries from a DataServer application need to provide only their foreign data source credentials to make a connection.

Secure connection to a foreign database can be made in three different ways:

- Wait for the Data Administration tool to prompt you for your data source user ID and password
- Provide a -U and -P parameter during OpenEdge client start up
- Add user ID and password in the Connection Parameters field of your logical database and they will be picked up for processing with the foreign data source connection

The User Id and Password that is used to connect to the foreign database from OpenEdge can be encrypted by the user. The following example demonstrates a self-encryption method available to OpenEdge applications that uses an agreed upon encryption key. The user encrypts the message and the DataServer detects the encryption by virtue of an encryption key prefix that informs the DataServer to decrypt just prior to connection with the foreign data source. The ABL syntax to encrypt a connection string is shown in the following example:

```
DEFINE INPUT PARAMETER cUserID AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER cPasswd AS CHARACTER NO-UNDO.
CONNECT <schema-holder> -1 NO-ERROR.
/* self service */
CONNECT <logical-database> -ld <logical-database> -dt ORACLE
VALUE( " -U " + cUserID +
" -P " + "oech1::" +
AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cPasswd)) NO-ERROR.
/* client/server */
CONNECT <logical-database> -ld <logical-database> -dt ORACLE
VALUE( " -U " + cUserID +
" -P " + "oech1::" +
AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(cPasswd))
-H <host> -S <port> NO-ERROR.
```

User Id and Password are encrypted internally on behalf of the application so that clear test is never seen from the application, nor is it transmitted over a connection in clear text. This security policy is enabled by default.

Users are still able to store clear text credentials in the dictionary if they chose to do so. However, if they self-encrypt credentials placed in the dictionary, as described in the above example, the credentials will remain encrypted while in schema holder storage and will be automatically decrypted by the DataServer when authentication is needed for the foreign data source connection.

Note: The use of encryption requires that both the OpenEdge client and server are of the same OpenEdge version. Connecting a new OpenEdge 11.4 client with an old pre-11.4 server will fail to connect. Backward compatibility is lost if the new client utilizes encryption during connection to an old server.

Running Oracle

An instance must be running for the Oracle database whose definitions make up the DataServer schema image. Before starting Oracle, you must set environment variables on your host machine.

To set environment variables on your host machine and run Oracle:

1. Set the `ORACLE_HOME` variable to the main Oracle directory.
2. Set the `ORACLE_SID` variable to the Oracle system identifier.

Note: A process can have the ORACLE_SID set to only one value at a time. If you want to access multiple Oracle databases on the same machine, set an ORACLE_SID variable to a different value for each database. Then create a separate broker for each database with the ORACLE_SID variable set to the matching value for each database. For more information, see the Oracle documentation for your machine type.

3. Start an Oracle instance for the target Oracle database.

Starting the DataServer processes

Before you create a schema holder, you must start the DataServer processes. This section provides brief instructions for starting the processes that make up the local and remote DataServers on host and client machines. For more information on starting the DataServer processes, see Connecting the DataServer on page 165

The following instructions use default executable names. If you built custom executables, substitute those names. When creating a schema holder, the startup syntax is the same whether you are running a local DataServer or using Oracle Networking.

1. To start a remote DataServer process in Windows using a UNIX client:
   a) Start the broker you configured for the Oracle DataServer.
   b) Run the client executable (_progres) on the client machine.
2. To start a remote DataServer process in Windows using a Windows client:
   a) Start the broker you configured for the Oracle DataServer.
   b) Run the client executable (prowin32.exe) on the client machine.
3. To start a local DataServer on UNIX, run the client executable (_progres).
4. To start a remote DataServer on UNIX using a UNIX client:
   a) On the host machine, enter the following command at the system prompt. Choose the *service-name* from the available services listed in your /etc/services file:

```
_probrkr -S service-name -H host
```

   b) Run the client executable (_progres) on the client machine.
5. To start a remote DataServer on UNIX using a Windows client:
   a) On the host machine, enter the following command at the system prompt, choosing the *service-name* from the available services listed in your /etc/services file:

```
_probrkr -S service-name -H host
```

   b) Run the client executable (prowin32.exe) on the client machine.
6. Start a local DataServer in Windows, run the client executable (prowin32.exe).

Starting ProBroker distributed DataServer broker with SSL-enabled Progress Networking

The schema holder can run on the client or can optionally be located on any other host that is accessible on your network. The network can be configured to use an SSL or a non-SSL connection irrespective of remote or local DataServer configuration. The schema holder can also be established using SSL network protocol by using “-ssl” command line parameter as shown in the following example:

```
_mprosrv -S service-name -ssl
```

The dataServer client should also be instantiated with the “-ssl” parameter if you also wish to establish a DataServer broker connection over an SSL enabled Progress networking, as follows:

```
prowin32  -1 SHDB -S 12345 -ssl
db my_dsn -ld ldb -dt ORA -U mb -P mb -S ccall7sv1 -ssl
```

Note: The following SSL connections are established using the above instruction:

- Schema holder DB and Progress Client
- OE DataServer Client and DataServer Server

The following instructions use default executable names. If you built custom executables, substitute those names. When creating a schema holder, the startup syntax is the same whether you are running a local DataServer or using Oracle Networking.

Non-default SSL certificates

The Unified Broker Framework has the “certStorePath” SSL property to specify the path to the trusted certificate store which is the root certificate for the certificate authority that was used to sign the server certificate. Servers using this customized certificate can connect clients instantiated with a matching certificate specified using `-certstorepath` command line parameter of the client.

However, in a Probroker configuration, responsibility for spawning a ssl or a non-ssl DataServer servers rests with the Probroker based on whether it was instantiated with an SSL or a non-SSL command line parameter. An SSL-enabled ProBroker can include the following startup parameters to customize the brokers use of certificates and key stores.

Table 32:

Parameter	Description
-certstorepath	The path to the trusted certificate store. The root certificate for the certificate authority that was used to sign the server certificate should be specified by this path.
-keystorepath	Specifies the key store location.
-keyalias	Specifies the alias set for a private/public key. If this parameter is not provided, the server certificate alias that is provided by OpenEdge is used.
-keyaliaspasswd	Specifies the encoded password set for the alias of the private/public-key to access the server Private Key / Digital Certificate key store entry. You can use the OpenEdge provided genpassword utility for encoding your passwords

For example a Probroker can be started and customized in SSL mode as follows:

```
    _probrkr -S ccall7sv2 -ssl
                                          -certstorepath
E:\\users\\<currentuser>\\par\\SHA224\\certs
                                          -keystorepath
E:\\users\\<currentuser>\\par\\SHA224\\keys
                                          -keyalias localhost
                                          -keyaliaspasswd 20333c34252a2137
```

The above Probroker can establish connection to a client which uses a matching certificate:

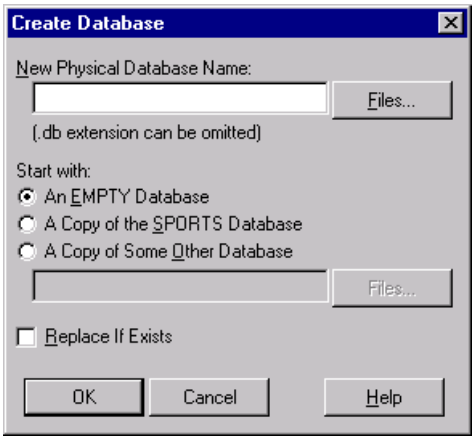
```
    prowin32  -pf parametersFile.pf
                      -certstorepath E:\\users\\ashukla\\par\\SHA224\\certs
                      -nohostverify
```

Creating an empty OpenEdge database

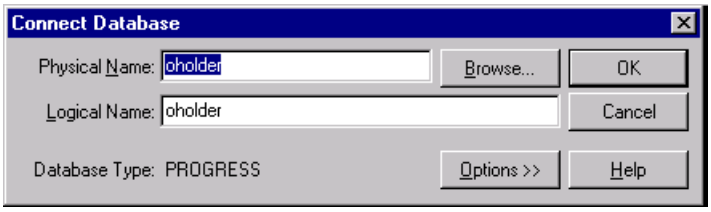
The DataServer uses the empty database as a schema holder for your Oracle data definitions. This section describes how to create a database with the Data Admin Tool. See *Manage the OpenEdge Database* for other techniques. This example uses Windows, but you can follow the same steps if you are using OpenEdge with a character-based interface, since the screens prompt for the same information. If you create a new OpenEdge application to be Unicode-enabled, ensure that you create the database from a Unicode source. A copy of the empty OpenEdge UTF-8 database can be obtained from `$DLC/prolang/utf/empty`.

To create and connect an empty OpenEdge database on a client machine:

1. Start the **OpenEdge Procedure Editor** with no database connected and access the **Data Admin Tool**.
2. Select **Database** > **Create**. The **Create Database** dialog box appears:



3. Type the schema-holder name, for example `oholder`, in the **New Physical Database Name** field.
4. Select **An EMPTY Database**.
5. Choose `OK`. The **Connect Database** dialog box appears. By default, the name of the newly created database appears in the **Physical Name** field:



Note: You do not have to provide any additional connection information. See *Manage the OpenEdge Database* for a complete description of the **Connect Database** dialog box.

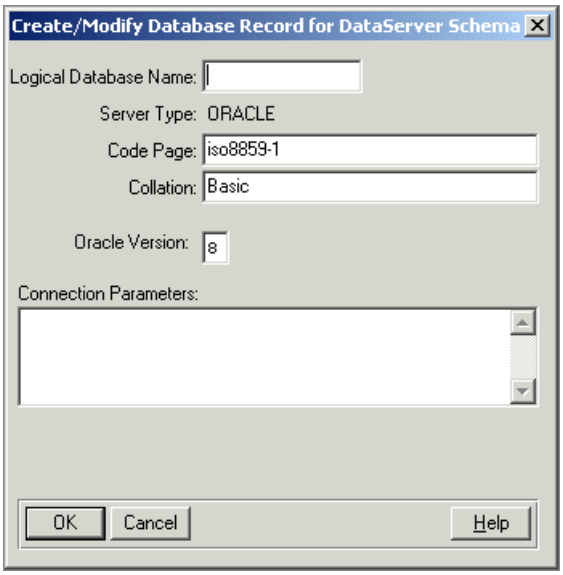
6. Choose `OK` to connect the empty OpenEdge database and return to the **Data Admin** main window.

Using the DataServer utility to create a schema image

Once you have started Oracle, started the DataServer processes, and created and connected to a local empty OpenEdge database, you can create the schema image. This example uses Windows; the character-based interfaces have the same sequence of screens.

To create a schema image:

1. From the Windows **Data Administration** main menu or the character-based **Data Dictionary** main menu, select **DataServer** > **Oracle Utilities** > **Create DataServer Schema** . The following dialog box appears:



2. Type a logical database name in the **Logical Database Name** field. You use the logical database name to connect to your Oracle database. You also use it to refer to the database in your programming applications. For more information on database names, see *Develop ABL Applications*.

   If you are building a schema holder for a distributed Oracle database, the logical name that you choose must be unique across the distributed database.

   Note: If you place the schema image from a second non-OpenEdge database into a schema holder, you must give the second schema image a different logical database name from the first schema. The schema holder has one physical name, but each schema image it contains must have a different logical name.

3. In the **Code Page** field, type the name of the code page for the schema image. The name must be the OpenEdge name for the code page. If you choose UTF-8 as the schema image code page, your schema holder's code page must also be UTF-8.

   If your Oracle environment uses a code page that OpenEdge does not support, you must supply a conversion table that translates between the OpenEdge client code page and the code page that Oracle uses.

   See Code pages on page 37 for more information on how the DataServer handles code pages and code-page conversion tables.

   For a complete discussion of code pages, see *Internationalize ABL Applications*.

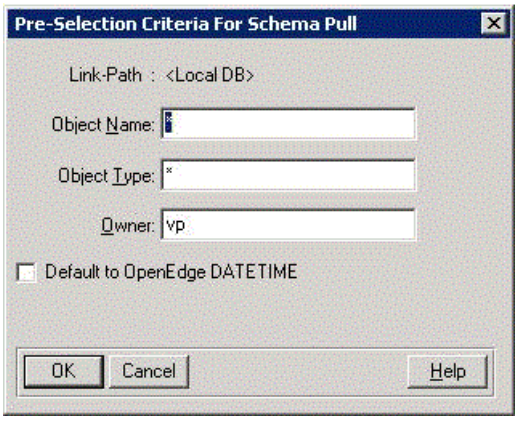
4. In the **Collation** field, type the name of the collation table.
5. In the **Oracle Version** field, type the version number of Oracle you are using. Possible values are 8 (the default), 9, or 10.
6. Type the required connection parameters in the **Connection Parameters** field.

   At a minimum, you need the User ID and Password parameters. Remote connections also require networking parameters. See Connecting a schema holder at startup on page 172 for a description of the required and optional connection parameters. This is how you identify the Oracle user in an Oracle Networking connection:

```
-U userid@service-name -P password
```

7. Choose **OK**. The utility prompts you to verify the Oracle user ID and password.

8. If you did not specify the User ID and Password parameters in the previous dialog box, type an Oracle user ID and password combination that has the privileges required for creating a schema holder. Required Oracle permissions table lists these permissions. Choose **OK**. The following dialog box appears:



9. Preselect the objects that the schema holder should contain. You can select these objects by object name, object type, or owner. By default, the wild card symbol (*) appears and specifies that the utility selects all objects. Typically, you should not include system-owned objects.

Note: Progress Software Corporation recommends that you do not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)

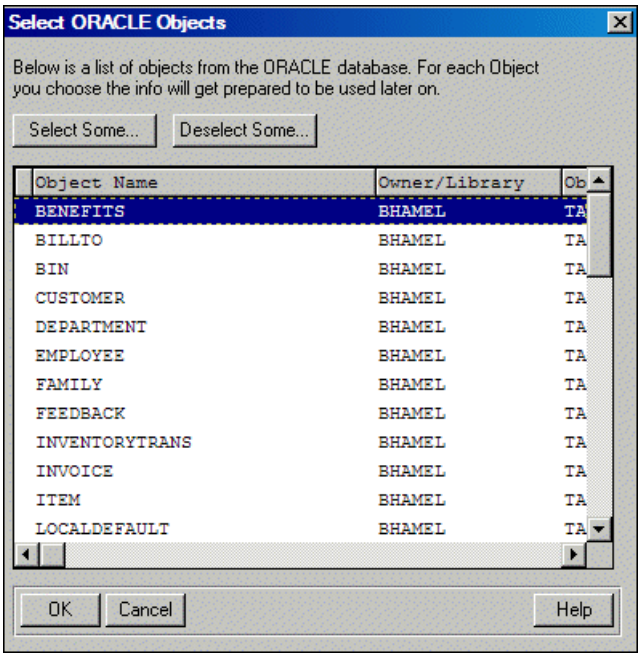
---

10. Choose **OK**.

Note: If your Oracle database is part of a distributed database, a dialog box listing the linked databases that make up the distributed database appears.

---

11. Select the linked databases whose objects you want to include in the schema holder, then choose `OK`. The **Pre-Selection** dialog box appears so that you can select objects by name, type, and owner.
12. Choose **OK**. The following dialog box appears:



13. Select the objects that you want to include in the schema holder.

    If object names repeat across a distributed database, the DataServer qualifies each name by adding `-n`. For example, if your database has two `INVOICE` tables, the schema image will list them as `INVOICE` and `INVOICE-1`.

14. Choose **OK**. The DataServer reads information about the database objects and loads the data definitions into the schema holder. The time this process takes depends on the size and number of the Oracle objects.

    For each table, the DataServer selects an index or the native `ROWID` to support the OpenEdge `ROWID`. If an Oracle object (such as some views) does not support the `ROWID` function, the DataServer issues the warning, "Please check warnings and messages in the file ds_upd.e." The `ds_upd.e` file lists the objects that do not support `ROWID`.

    You can change the DataServer's selection by using the Data Dictionary. For example, you can choose to use an alternate integer index or the native `ROWID`. See Defining the ROWID on page 231 for instructions. See ROWID function on page 72 for more information on `ROWID`.

    If the DataServer encounters Oracle errors while creating or maintaining schema information or loading data, it displays the Oracle message and message number on screen. See your Oracle documentation for complete descriptions of the error messages.

    The DataServer makes an additional connection to the schema holder when it creates, updates, or verifies a schema image. If you are connecting to multiple databases or schema images, make sure that you set the Number of Databases (`-h`) parameter to a value that accommodates the number of databases and schema images plus one.

Maintaining a schema holder

The OpenEdge Data Administration tool provides a set of Oracle utilities that you can use to maintain a schema holder. The DataServer Tutorial on page 205 describes these utilities.

If you are using non-OpenEdge applications to change the structure of the supporting Oracle database, be sure to update the schema image to reflect those changes. Some database operations require a certain technique to ensure that the schema image and the supporting database function together.

The following table lists common database operations and the required techniques.

Table 33: Database operations and the schema holder

Database operation	Technique
Modify a column in an Oracle table	Update the schema image by using the OpenEdge DataServer for Oracle utilities.
Add a column to an Oracle table	Update the schema image by using the OpenEdge DataServer for Oracle utilities.
Insert a row in an Oracle table	Use the Oracle sequence generator to generate a value. Assign that value to the `PROGRESS_RECID` column for that row. If you do not use the sequence generator, ABL might fail when it tries to access that table.
Add an index to an Oracle table **Note:** The index cannot be a function-based index.	Update the schema image by using the DataServer for Oracle utilities.
Modify data definitions in the schema image	Update the Oracle database to reflect your changes by using the Incremental Schema Migration utility.

Deploying a schema holder

The guidelines and techniques that apply to deploying an OpenEdge database apply to deploying a schema holder for an Oracle database. There is, however, an additional consideration. Make sure that you make changes to both the supporting Oracle database and the schema holder. For example, when you provide an SQL script to modify an Oracle database, you must also provide a new data definitions file or ABL code to update the schema holder. Keep in mind that the SQL script that you send to a customer's site must consider how the Oracle database is configured on their system, including table space and security issues.

There are two techniques for updating a deployed schema holder:

- Allow your users to use the DataServer Update/Add Table Definitions utility.
- Send a new data definitions file for the schema holder. Your users can load the `.df` file into an empty OpenEdge database.

To maintain a deployed DataServer application:

1. Make changes to the Oracle database.
2. Run the Update/Add Table Definitions utility on one schema holder.
3. Run the PROUTIL utility to truncate the before image (`.bi`) file.
4. Recompile code against the updated schema holder to build new r-code.
5. Send out copies of the new `.r` files and the schema holder `.db` and `.bi` files to your users.

Note: You can improve the performance of a deployed application by using the `-Dsrv skip-schema-check` startup parameter. See Skipping schema verification on page 126 for more information.

Setting up a schema holder as a Windows service

The schema holder can reside in the Windows host or on the client machine. A schema holder on a client machine (a local schema holder) performs better. However, a schema holder in the Windows host (a remote schema holder) might be easier to maintain in a multi-user environment. Setting up the schema holder so that it can run as a native Windows service means that the schema holder can run without users connected to it. Its availability is not dependent on any one user connection.

To set up a remote schema holder in a Windows host and make it accessible to multiple users, configure a database server using OpenEdge Management or OpenEdge Explorer. By default, the server will be started as a Windows service. See *Manage the OpenEdge Database* for a complete description of how to configure a database server.

Configuring a self-service DataServer agent in the OpenEdge Appserver and Progress Application Server

The following sections describe how to start self-service DataServer agent in the OpenEdge AppServer and Progress Application Server.

Configuring the OpenEdge AppServer agent as a self-service DataServer session

To configure the OpenEdge AppServer agent as a self-service DataServer session:

1. Make sure that the AdminService is running.

   Note: For more information on using the `PROADSV` tool, see the OpenEdge Application Server: Administration guide.

2. Add the following property in the ubroker.properties file under `UBroker.AS.asbroker1 section: srvrStartupParam= -pf "$WRKDIR/conn.pf"` where AS1 is the AppServer.
3. Start the Application Server broker using the following command:

```
asbman -i asbroker1 -start
```

   Note: For more information on using asbman commands and its parameters, refer to the *OpenEdge Application Server: Administration*.

4. Open an ABL session and execute the client.p to connect to the self-service DataServer through an OpenEdge Appserver instance as shown in the following example:

```
client.p
----------
DEFINE VARIABLE hServer AS HANDLE NO-UNDO.
DEFINE VARIABLE i AS logical.
CREATE SERVER hServer.
/*hServer:CONNECT('-AppService asbroker1 -S 5162 -H <hostname> -nsClientMinPort 5000
-nsClientMaxPort 50100') NO-ERROR.*/
hServer:CONNECT('-AppService asbroker1').
i = hServer:Connected().
message i.
MESSAGE "Error:" ERROR-STATUS:ERROR.
IF ERROR-STATUS:ERROR = FALSE THEN
DO:
RUN server.p ON hServer .
hServer:DISCONNECT().
END.
ELSE
MESSAGE ERROR-STATUS:GET-MESSAGE(1).
DELETE OBJECT hServer NO-ERROR.
output close.
quit.
```

Configuring the Progress Application Server agent as a self-service DataServer session

To configure the OpenEdge AppServer agent as a self-service DataServer session:

1. Run the following command from **Proenv** to create an instance of Progress Application Server for OpenEdge:

```
pasman create -p <http portnum>  -P <https portnum> -s <shutdown port>  <instance
name>
```

Note: In the above command `-p port_num` specifies the TCP port that listens for HTTP messages, the default is 8080, and `-P port_num` and specifies the TCP port that listens for HTTPS messages, the default is 8443. `-s port_num` specifies the TCP port used to stop an instance. `instance name` is the name of the Progress Application Server instance.

2. Edit the **OpenEdge.Properties** file located in the `conf` folder to update the `agentStartupParam` as follows:

Table 34:

```
agentStartupParam=-T "${catalina.base}/temp" -pf %WRKDIR%/ORA.pf
```

3. Create a schema holder as described in the Creating a schema holder section.
4. Start the schema holder using the Proserve command.
5. Start the Progress Application Server instance using the following command from the `cd pasds/bin` location to start :

Table 35:

```
tcman.bat start
```

6. To connect the client application to Progress Application Server, use one of the following methods:
   a. To connect using APSV, use the following sample client.p file to connect to the multi-threaded agent:

```
DEFINE var a as INTEGER.
DEFINE var b as INTEGER.
DEFINE var outmsg1 AS INTEGER.
DEFINE VARIABLE iCount              AS INTEGER      NO-UNDO.
DEFINE VARIABLE happsrv AS HANDLE NO-UNDO.

CREATE SERVER happsrv.
IF happsrv:CONNECT("-URL http://<machine name>:< http portnum >/apsv -sessionModel
 Session-Managed") THEN
 DO :
    a = 10.
    b = 2.
       RUN test.p on happsrv(INPUT-OUTPUT a ,INPUT-OUTPUT b ,INPUT-OUTPUT outmsg1).

        MESSAGE "Output Value: " outmsg1.
 END.
 happsrv:disconnect().
 DELETE OBJECT happsrv.
```

b. To connect using REST, run the `deployREST.bat` and `MultipleService.paar`. Run the `tcman.bat` from `http://localhost:8811/rest/MultipleOperationsService/test1/9`

c. To connect using SOAP, start the `tcman.bat` from the `cd pasds\bin` location. Run `deploySOAP.bat` and GetCustName.wsm ROOT from the `http://localhost:8811/soap/wsdl?targetURI=urn:getCustName`. From the location, **cd %WRKDIR%** run the commands:

Table 36:

```
prowin -b -p ClientGetCustNameSSL.p
$PROEXE -b -p client.p -param find_first.p &
$PROEXE -b -p client.p -param find_last.p &
$PROEXE -b -p client.p -param for_each.p &
$PROEXE -b -p client.p -param for_first.p &

OUTPUT AT
C:\OpenEdge\WRK117\pasds\work
dataserv.lg at
C:\OpenEdge\WRK117\pasds\work
input files
C:\OpenEdge\WRK117\pasds\openedge
```

6

Connecting the DataServer

This chapter provides instructions for running the DataServer after you have configured its modules and created a schema holder.

For details, see the following topics:

- Starting the local DataServer
- Starting the remote DataServer
- Connecting a schema holder
- Connection failures and OpenEdge responses

Starting the local DataServer

Starting the OpenEdge client starts the local DataServer. You include information about the Oracle database, user ID, and password in the startup command.

To start the local DataServer:

1. Set `ORACLE_HOME` and `ORACLE_SID` and start an instance for the supporting Oracle database.
2. Set the environment variables in the shell from which you are starting the DataServer. The following table describes how to set them:

For this variable . . .	Set . . .
DSLOGDIR	The pathname to the directory where the DataServer creates the log file, dataserv.lg, to track processes and error messages. By default, the DataServer creates this file in the current directory. (Optional)
ORACLE_HOME	The pathname of the directory where Oracle is installed.
ORACLE_SID	The identifier for the Oracle instance you are accessing.

Note: If you change the values of any environment variables, such as ORACLE_SID or ORACLE_HOME, you must shut down the DataServer processes and restart them.

3. Enter one of the following commands, depending on the platform:

   a) For a Local DataServer on UNIX, enter the following command:

```
pro schema-holder-name -db ora-dbname -dt ORACLE
  -ld oracle-logical-db-name -U userid -P password
```

   b) For a Local DataServer in Windows, enter the following command:

```
prowin32 schema-holder-name -db ora-dbname -dt ORACLE
  -ld oracle-logical-db-name -U userid -P password
```

   For example, the following command starts OpenEdge with the DataServer on a UNIX machine, connects a local schema holder oholder in single-user mode, and connects an Oracle database orademo with a user scott and the password tiger:

```
pro oholder -db oradb -dt ORACLE -ld orademo -U scott -P tiger
```

   For Oracle Networking startup information, see Connecting through Oracle Networking on page 175.

Starting the remote DataServer

This section describes how to start the processes that make up the remote DataServer for Oracle. First, you start the DataServer broker process on your host machine. Then you start an OpenEdge client process on a UNIX machine or on a PC running Windows that connects to the schema holder and the Oracle database.

Note: Only one broker can run in a directory at a time. If you want to run multiple brokers, run each in a separate directory.

Starting the DataServer in the Unified Broker framework

The Unified Broker framework administers only the server module (`_orasrv` in Windows and on UNIX) of the DataServer. After starting the server, start your OpenEdge client as you would in any remote DataServer configuration. The following sections describe how to run the DataServer in the Unified Broker framework in Windows and on UNIX.

In the Windows host

Use OpenEdge Management or OpenEdge Explorer tool to run DataServer processes in the Windows host. Optionally you can use the command-line technique described in On the UNIX host on page 167. To configure the DataServer, see Configuring the DataServer in the Unified Broker framework on page 142. This section presents basic start, stop, and status check procedures for the DataServer.

1. To start the DataServer:
   a) From the left pane of OpenEdge Management or OpenEdge Explorer, select the **Oracle DataServer** folder and double-click. The list of existing DataServer brokers for Oracle appears in the right pane.
   b) Select the DataServer broker you want to start and right-click. A pop-up menu appears.
   c) Choose the `Start` option from the pop-up menu. The **Status** in the right pane changes to **Running** for that DataServer broker.
2. To check the status of a DataServer broker:
   a) Select the DataServer broker andright-click.
   b) Choose the **Status** option from the pop-up menu.

   Note: See the OpenEdge Management or OpenEdge Explorer Online Help for more information the various **Status** tabs.
3. To stop the DataServer broker:
   a) Select the DataServer broker and `right-click`.
   b) Choose the `Stop` option from the pop-up menu. The **Status** in the right pane changes to **Not Running** for that DataServer broker.

   Note: For more information on using OpenEdge Management or OpenEdge Explorer tool, see the OpenEdge Management or OpenEdge Explorer Online Help.

On the UNIX host

Use the Unified Broker framework command-line interface to run DataServer processes on the UNIX host. To configure the DataServer, see Configuring the DataServer in the Unified Broker framework on page 142.

Start the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

```
oraman -name broker-name -start
```

Where *broker-name* is the name you gave the broker when you configured it.

You can run the broker from a remote machine. In that case, you specify additional options that identify the remote host. For example:

```
oraman -name broker-name -start -host host-name -user user-name
```

where *broker-name* is the name you gave the broker when you configured it; *host-name* is the name of the host machine on which you want the broker to run; and *user-name* is the user ID of the system account under which the AdminServer is running. This account can be different from the one that owns the DataServer broker, which in turn can be different from the user profile you use to connect to the Oracle database.

You can also use the `oraman` utility to check the status of a broker. To check the status of a broker, enter this command at the system prompt on the machine where the broker is running:

```
oraman -name broker-name -query
```

To stop the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

```
oraman -name broker-name -stop
```

To stop the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

You can stop a broker on a remote machine by adding the `-host` and `-user` options.

See DataServer Command Line Utilities and Startup Parameters on page 283 for a complete description.

On the client

After you have started the broker on the host machine, you can connect your UNIX or Windows client. Use the same parameters that you would use for connecting to the schema holder and Oracle in a standard configuration. In addition:

- Include the `-Dsrv SVUB,1` parameter. This parameter allows you to connect to the broker administered by the Explorer.
- Include the `-DataService`*data-service* parameter to connect through a NameServer to the broker. The value for *data-service* must specify a valid name from the DataService list registered to this NameServer as defined by your *appServiceNameList* entry in the broker properties file. If a default DataService has been defined for your broker instance, you can omit this parameter and connect using the default service.

  For diagnostic purposes, it is acceptable to bypass the NameServer connection and connect directly to the broker instance. To do this, specify the reserved DataService name "none" or add the `-DirectConnect` parameter as follows:

  ```
  -DataService none
  ```

Or

```
-DirectConnect
```

- Set the -S parameter to one of the following:
  - The port number assigned to the controlling NameServer (when the -DataService value is not "none") or the port number of the broker instance that you started in the Explorer (when the -DataService value is "none")
  - The service name in your service file, \windows-install-dir\system32\drivers\etc\services in Windows or /etc/services on UNIX, whose associated port matches the port of the controlling NameServer (when the -DataService value is not "none") or the broker instance that you started in the Explorer (when the -DataService value is "none")
- Set the -H parameter to the name of the machine where the NameServer and/or broker instance is running.

If you do not set the required -Dsrv SVUB,1 and optional -DataService *data-service* connection parameters as described in this section, the client is assumed to be configured for a standard OpenEdge broker and the -H and -S parameters are used to locate a Probroker executable on the appropriate host machine. By setting the SVUB parameter on, you redirect the -H and -S parameters to locate the appropriate NameServer and/or broker on the host machine.

The following example illustrates how to use these connection parameters for a client that connects to a NameServer:

```
CONNECT oraholder -db orademo -dt ORACLE -U bob -P bobpass -H host1
-S oraservice -DataService orabroker1 -Dsrv SVUB,1
```

Starting and stopping the ProBroker process from the command line

The broker can be started from a Windows or UNIX command line prompt in addition to the OpenEdge Management or OpenEdge Explorer interface.

To start the broker process from the command line:

1. Start an instance for the supporting Oracle database.
2. Set the environment variables in the shell from which you are starting the DataServer. The following table describes how to set them.

Table 37: Environment variables for the remote DataServer

Variable	How to set it
DSLOGDIR	The pathname to the directory where the DataServer creates the log file, dataserv.lg, to track processes and error messages. By default, the DataServer creates this file in the current directory. (Optional)
ORACLE_HOME	The pathname of the directory where Oracle is installed.

Variable	How to set it
ORACLE_SID	The identifier for the Oracle instance you are accessing.
ORASRV	The name of the executable (including the path) of the Oracle DataServer.
PROBRKR	The pathname to the executable of the broker.

Note: If you change the values of any environment variables, such as ORACLE_SID, ORACLE_HOME, or ORASRV, you must shut down the DataServer processes and restart them.

3. To start the DataServer broker process, enter the following command at the system prompt. Select the *service-name* from the list of available services:

```
broker-executable-name -H host-name -S service-name
```

For example, the following command uses the default broker executable:

```
_probrkr  -H paris -S demosv
```

This example starts an SSL –enabled ProBroker on the “demosv” port of the local host:

```
_probrkr  -S demosv -ssl
```

4. To stop the DataServer modules by shutting down the broker, enter this command at the system prompt of the client machine:

```
proshut -Gw -H host-name -S service-name
```

Starting the UNIX client process

After starting the remote broker, you start the OpenEdge client process on your UNIX machine by running this executable:

```
pro
```

You can supply the connection parameters required by the DataServer when you start the client process, or you can include them in the **Connection Parameters** field when you create a schema holder. For example, this command starts the OpenEdge client and connects to a read-only schema holder named `oholder` for an Oracle database named `orademo`:

```
pro oholder -RO -db oradb -ld orademo -H host1 -S oservice
-U scott -P tiger
```

See Connecting a schema holder at startup on page 172 Optional connection and startup parameters on page 177 and the for descriptions of the command line and the required and optional parameters.

Starting the Windows client process

Start the OpenEdge client process in Windows by running the `prowin32.exe` executable. The Windows executable includes support for the DataServer.

You can create a program icon for the OpenEdge client process. On the command line for the program icon, enter the following information:

1. The executable.
2. The schema holder name. If you have multiple schema holders, create a program icon for each schema holder.
3. The connection parameters required by the remote DataServer configuration.
4. This is a sample command line:

```
prowin32 oholder -RO -db oradb -H host1 -S oserviceA
    -U scott -P tiger
```

See Connecting a schema holder at startup on page 172 Optional connection and startup parameters on page 177 and the for descriptions of the command line and the required and optional parameters.

Connecting a schema holder

In addition to the automatic connection that can take place when you create a schema holder, OpenEdge provides these techniques for connecting a schema holder:

- Use the Data Dictionary or Data Administration tool. From the main menu, select **Database** > **Connect** and supply the schema holder's physical name and appropriate connection parameters. You connect to the Oracle database when you select it as your working database.
- Use the `CONNECT` statement (see the "CONNECT Statement" reference entry in *ABL Reference*). For example, this command connects a local read-only schema holder named `oholder`:

```
CONNECT oholder -RO -db oradb -dt ORACLE -ld orademo -S oserviceA
  -H doc4 -U scott -P tiger.
```

- Use connection parameters when you start OpenEdge. You can include these parameters in a parameter file that you specify when you start OpenEdge. A parameter file is portable and easy to maintain. For information on how to create a parameter file, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

You can use different connection techniques in combination with each other. For example, you can connect the schema holder at startup, then connect to the DataServer using an ABL `CONNECT` statement. Any combination of connection techniques works. You must connect to the schema holder in single user (`-1`) or read-only (`-RO`) mode. If you are using a remote DataServer, you must start a broker on the host machine before you can connect.

When you connect to the schema holder and the Oracle database in a single startup command or connection statement, be sure to specify parameters that affect the schema holder before the Database Name (`-db`) parameter. Specify only parameters that affect the Oracle database connection after the `-db` parameter.

The following sections explain how to connect both a schema holder and an Oracle database when you start up OpenEdge.

Connecting a schema holder at startup

OpenEdge supports connection parameters that you can use to connect the OpenEdge schema holder and the Oracle database. These parameters control how your system connects to a database. When the OpenEdge DataServer for Oracle runs in a remote configuration, your startup command or parameter file must include parameters that control networking options.

The following table describes the connection parameters that relate to the remote DataServer and networking options.

Table 38: Remote DataServer connection parameters

Parameter	Description
Host Name (`-H`)	Indicates the name of the host machine in the network.
Service Name (`-S`)	Indicates the name of the service you are calling. The service you are calling is the broker on the host machine. Use the same name you used when you started the broker.

The following command starts OpenEdge for Windows:

```
prowin32 schemaholdername -db fill-char -dt ORACLE
  -ld oracle-logical-db-name -H hostname -S service-name
  -U userID -P password
```

You can use the previous command to start OpenEdge for Windows in the following situations:

- In single-user mode
- In a remote-DataServer configuration
- With a local schema holder connected
- With an Oracle database connected

The following table describes the parameters required for connecting to a schema holder and an Oracle database.

Table 39: Required connection parameters

Parameter	Description
User ID (`-U`)	Supplies the user ID that the OpenEdge DataServer for Oracle uses to log into the Oracle RDBMS. Oracle allows you to pass the user ID and password as a single string to the `-U` parameter. If you specify `-U userid/password` or `-U userid/password@service-name`, the `USERID` function returns the password information or the password and service in addition to the user ID.
Password (`-P`)	Supplies the password that the OpenEdge DataServer for Oracle uses to log into the Oracle RDBMS unless it was specified with the `-U` parameter.
Physical Database Name (`-db`)	For OpenEdge databases, the string that follows this parameter is the physical name of the database you want to connect. However, for Oracle databases, this physical database name can be any fill characters. For example, use the logical database name after the `-db` parameter or any other designation, such as `oradb`.
Single-user Mode (`-1`)orRead-only (`-RO`)	Specifies that a database is used in single-user mode. When you connect the schema holder, you must connect either in single-user mode or as read-only. Specifies that a schema holder is read-only. Connecting a schema holder as read-only increases processing speed at compile time. It also allows multiple client processes on the same machine to access the schema holder without starting additional server processes.

You can create a parameter file with the required parameters for each database. You can add more startup parameters than the ones listed—these are just the required parameters. For information on how to create a parameter file, see *Startup Command and Parameter Reference*.

The following example commands demonstrate how to start DataServer clients that connect to the schema holder and Oracle database at startup.

Remote DataServer: Windows client

In this example, the schema holder's physical name is `oholder`; it is read-only; the Oracle database's logical name is `orademo`; the host name is `host1`; the service name is `oserviceA`; the user ID is `scott`; the password is `tiger`. This configuration assumes you started the remote DataServer broker using this command line interface:

```
prowin32 oholder -RO -db oradb -ld orademo -H host1
  -S oserviceA -U scott -P tiger
```

Remote DataServer via NameServer: Windows client

The following attributes pertain to the code in the example shown in this section:

- The schema holder's physical name is `oholder` and it is read-only.
- The Oracle database's logical name is `orademo`.
- The user ID is `scott`.
- The password is `tiger`.
- The host where the NameServer is running is `nshost`.
- The service is port number 5162, the default port of the Name Server.
- The `-Dsrv` switch indicates this connection is using a NameServer.
- The DataService is `orabroker1`.

```
prowin32 oholder -RO -db oradb -ld orademo -U scott -P tiger -H nshost
  -S 5162 -Dsrv SVUB,1 -DataService orabroker1
```

Local DataServer: Windows

In this example, the schema holder's physical name is `oholder`; it is read-only; the Oracle database's logical name is `orademo`; the user ID is `scott`; and the password is `tiger`:

```
prowin32 oholder -RO -db oradb -ld orademo -U scott -P tiger
```

The DataServer is local so there is no need for host and service parameters.

Remote DataServer: UNIX client

In this example, the schema holder's physical name is `oholder`; it is read-only; the Oracle database's logical name is `orademo`; the remote DataServer host is `host1`; the service name for the DataServer broker is `oserviceA`; the user ID is `scott`; and the password is `tiger`. This configuration assumes you started the remote DataServer broker using the command line interface, as shown:

```
pro oholder -RO -db oradb -ld orademo -H host1
  -S oserviceA -U scott -P tiger
```

Local DataServer: UNIX

In this example, the schema holder's physical name is `oholder`; it is read-only; the Oracle database's logical name is `orademo`; the user ID is `scott`; and the password is `tiger`:

```
pro oholder -RO -db oradb -ld orademo -U scott -P tiger
```

The DataServer is local, so there is no need for host and service parameters.

Connecting a Schema holder and a broker using SSL-enabled Progress Networking

The Schema holder can run on the client or can optionally be located on any other host that is accessible on your network. The network can be configured to use an SSL connection to your schema holder irrespective of whether the DataServer is configured remotely or locally.

The Schema holder connection can be established using the SSL network protocol by specifying the -ssl parameter from the command line as follows:

```
_mprosrv SHDB -ssl -S <schema-holder-port>

_probrkr -H <dataserver-host> -S <dataserver-port> -ssl
```

The corresponding DataServer client should also be started with the "-ssl" parameter.

The Progress client connects to the schema holder as follows:

```
prowin32 -1 SHDB -ssl -S <schema-holder-port>
```

The Progress client connects to the brokered Dataserver as follows:

```
-db my-dsn -ld my-ldb -dt ORA -U <user> -P <pwd> -H <dataserver-host> -S
<dataserver-port> -ssl
```

Connecting through Oracle Networking

Oracle network interfaces allow you to access an Oracle RDBMS instance running on a remote machine. You do not need any OpenEdge-supplied software on the machine where Oracle is running. However, you need Oracle Networking installed on the client machine and an Oracle Listener on the machine where the Oracle instance is running.

To access an Oracle distributed database, you connect through Oracle Networking, just as you would connect to a single instance. The DataServer detects all the links that make up the distributed database. DataServer access to a distributed database is transparent and requires no additional commands or startup parameters.

To connect to an Oracle instance through Oracle Networking:

1. Make sure that the remote Oracle instance is running.
2. Make sure that Oracle Networking is running on the client machine.
3. Supply the following connection parameters in your OpenEdge startup command or parameter file, depending on the platform and configuration:

a) For a Windows client on an Oracle Networking, enter this command:

```
prowin32 schema-holder-name -RO -db oracle-dbname
-U userid@service-name -P password
```

b) For a UNIX client on an Oracle Networking, enter this command:

```
pro schema-holder-name -db oracle-dbname
-U userid@service-name -P password
```

For example, the following command starts OpenEdge on a UNIX machine, connects a local schema holder `oholder` in read-only mode and connects the remote Oracle instance `orademo` and uses the service `oserviceA`. The username is `scott` and the password is `tiger`, as shown:

```
pro oholder -RO -db oradb -U scott@oserviceA -P tiger
```

If you are connected through Oracle Networking, the OpenEdge `USERID` function returns `scott@oserviceA`, instead of just `scott`.

If you use Oracle Networking, the service name must be part of the username or password. You can use one of these forms:

```
CONNECT <logical name> -U <user@service-name> -P <password>

CONNECT <logical name> -U <user/password@service-name>

CONNECT <logical name> -U <user> -P <password@service-name>
```

Connecting with external credentials

OpenEdge DataServer for Oracle supports external password credentials. Specifically, the DataServer for Oracle uses Lightweight Directory Access Protocol (LDAP) through the OCI driver to provide directory authentication through the global user management capabilities of the Oracle Internet Directory (OID).

The Oracle DataServer makes database connections through the bound OCI client library that is capable of interfacing with an Oracle Server using Oracle Advanced Security. When an Oracle database environment is configured to use external authentication, a centralized service is shared by Oracle and other application environments to provide security and single sign-on capability. For instance, an LDAP directory service can be configured to provide global user authentication and authorization using the `ldap.ora` or `tnsnames.ora` configuration files. LDAP directory services can be configured in many different ways with Oracle databases to provide centralized enterprise-level authentication services.

An OpenEdge DataServer client can obtain external authentication by sending a connection request with only a user id and password delimiter, absent the actual user id and password. For instance:

```
/@<tnsnames-server>.<ora-instance>
```

Depending on the mechanism used, external credentials may or may not be provided independent of the global user management features of the OID. For instance, public key infrastructure (PKI) digital certificates can be passed directly to a target Oracle Enterprise server. Oracle Wallets can also be provided in an external password store and may perform authentication in conjunction with a particular external authentication server.

Optional connection and startup parameters

You can use other OpenEdge startup and connection parameters with the DataServer.

The following table describes parameters that are especially relevant to using the OpenEdge DataServer for Oracle.

Table 40: Optional connection and startup parameters

Parameter	Description
Database Type (`-dt ORACLE`)	Specifies that the non-OpenEdge type of the target database is Oracle.
DataServer (`-Dsrv`)	Specifies parameters that allow you to control how the DataServer processes queries. `-Dsrv` is a connection parameter. See Query tuning with connection and startup parameters on page 178 for more information.
Logical Database Name (`-ld`)	Specifies the logical name for the Oracle database. This is the name by which you refer to the database in your applications.
Index Cursors (`-c`)	Specifies the maximum number of Oracle cursors per user that OpenEdge opens. This parameter is valid only for connecting to Oracle databases. It has no effect when connecting to the schema holder. See Index cursors on page 197 for more information.
Schema Cache File (`-cache`)	Specifies the name of a binary file that contains the schema for a database. Reading schema from this file is more efficient than reading from a remote schema holder. See Local schema caching on page 199 for more information.
Data Service(`-DataService`)	Specifies the data service the NameServer uses. For more information, see Starting the DataServer in the Unified Broker framework on page 167. **Note:** This parameter is required for OpenEdge Management or OpenEdge Explorer connections and must be used in conjunction with the `-Dsrv SVUB,1` option.
Direct Connect(`-DirectConnect`)	When using OpenEdge Management or OpenEdge Explorer, this parameter instructs the client to bypass the controlling NameServer and connect directly to the Oracle DataServer Broker.

Parameter	Description
Internal Code Page (-cpinternal)	Specifies the name of the code page that OpenEdge uses in memory. This code page must match the code page that the Oracle database uses. Required when using a code page other than ibm850 or iso8859-1, such as in a DBE DataServer configuration.
Stream Code Page (-cpstream)	Specifies the name of the code page that OpenEdge uses for stream I/O. This code page must match the code page that the Oracle database uses. Required when using a code page other than ibm850 or iso8859-1, such as in a DBE DataServer configuration.

Query tuning with connection and startup parameters

In addition to controlling aspects of how the DataServer handles queries programmatically within ABL statements, you can control the same aspects through startup and connection parameter options.

Note: Startup and connection parameters override query-tuning defaults. Options set in the QUERY-TUNING phrase take precedence over startup and connection parameters. For example, if you specify "qt_debug,SQL" at connection it will override the NO-DEBUG default. If you additionally specify NO-DEBUG for a specific query, the NO-DEBUG for the query will override the connection setting, and no SQL will be written to the log file for that query. In this example, the DataServer will write a report that includes all the SQL it generates for the application, except for the query with the NO-DEBUG option. See Query tuning on page 128 for more information.

You override query-tuning defaults with the DataServer (-Dsrv) connection parameter when you connect to the Oracle database. The -Dsrv switch accepts a comma separated list of options and values. To guarantee correct parsing, there can be no spaces in the list, as shown in the following syntax:

Table 41: Syntax

```
CONNECT db-name -U user-name -P password
  -Dsrv query-tuning-option1,value1,
  query-tuning-option2,value2.
```

The following table describes the query-tuning options that you can specify with the -Dsrv parameter. Unless otherwise indicated, these options apply at compile and run time.

Table 42: Connection query-tuning options

Option	Description
qt_bind_where qt_no_bind_where	• Specifies whether the DataServer uses Oracle bind variables for values in WHERE clauses. • Specify qt_no_bind_where to use literals. • Use at run time only. • **Default**: qt_bind_where.

Option	Description
qt_cache_size,*integer*,QT_BYTE qt_cache_size,*integer*,QT_ROW	• Specifies the size of the cache (in bytes or records) for information used by lookahead or standard cursors. • Byte maximum: 65535 bytes. • Byte minimum: Specify the number of bytes contained in a single record. For joins, specify the number of bytes contained in two joined records. If a join returns a 500-byte record, you need a cache of 1000 bytes. • Row maximum: The number of records that can fit in 65535 bytes. See Caching records on page 121 for more information. • Row minimum: 1. If the server performs the join, the minimum is 2. • If QT_BYTE or QT_ROW is omitted, QT_BYTE is assumed. • **Default**: 1024 bytes with standard cursors; 8192 with lookahead cursors.
qt_lookahead qt_no_lookahead	• Specifies whether the DataServer uses lookahead or standard cursors. • Specify qt_no_lookahead for behavior that is consistent with OpenEdge. • **Default**: qt_lookahead, except with statements that use an EXCLUSIVE lock.

Option	Description
Get Previous(`-Dsrv srv-get-prev`)	• The addition of this parameter allows for queries defined with `qt_no_lookahead` to correctly execute a `GET PREV` after a reposition. • Without this switch, when a reposition fails to find the matching record in either the client or server cache, and new SQL is generated to satisfy the query, the resulting record is reset to the starting point of the query, rendering the `GET PREV` meaningless.
Warning suppression(-Dsrv PRGRS_WARNING_SUPPRESSION)	• Specifies the Oracle warnings to be suppressed by the DataServer at connection time. The DataServer treats and handles any warnings received from Oracle as errors. If you want your DataServer application to ignore Oracle warnings, you execute `RUN STORED-PROCEDURE` in your ABL editor to alter your Oracle session attributes to disable **all** the warnings or just the informational warnings in your DataServer connection. For more information on `RUN STORED-PROCEDURE` statement, see RUN STORED-PROCEDURE statement on page 87. However, if you want your DataServer application to overlook only **certain** warning conditions returned from Oracle, you can use the PRGRS_WARNING_SUPRESSION switch. • In the PRGRS_WARNING_SUPRESSION switch, you can specify a maximum of 25 error codes as a comma-separated list ending with a semi-colon. Refer to the example in the section. • The error codes specified in the PRGRS_WARNING_SUPRESSION switch are suppressed by the DataServers **only if** Oracle treats those codes as warnings. • Errors codes added to the PRGRS_WARNING_SUPRESSION switch are unconditional, that is, the specified warning codes are suppressed by the DataServer irrespective of how the warning condition is produced by Oracle. • **Default**: Disabled. The parameter is disabled to allow for backward copatibility. So, by default, Oracle warnings are treated as errors by the Oracle DataServer.

The following example shows how to use the query-tuning options. The `qt_cache_size` may enhance performance as it increases the default size of the cache (of multiple records used by lookahead cursors) to 32K . In addition, the following example uses a query tuning logging switch that enables the DataServer to write an extended report on the SQL statements it executes:

```
CONNECT oradb -U user -P password
  -Dsrv qt_cache_size,32000,qt_debug,EXTENDED.
```

The following example shows how to use the PRGRS_WARNING_SUPPRESSION parameter to specify Oracle warning codes, ORA-24347 and ORA-24348, from being treated by the DataServer applications as error conditions:

```
CONNECT oradb -U user -P password
  -Dsrv PRGRS_WARNING_SUPPRESSION,24347,24348;
```

Use startup parameters to override three other query-tuning defaults, `INDEX-HINT`, `JOIN-BY-SQLDB` and `USE_MERGE`. The following table describes these startup parameters.

Table 43: Query-tuning startup parameters

Startup Parameter	Description
Index Hint(`-noindexhint`)	• Specifies that the DataServer does not provide index hints to the Oracle DBMS. Generally index hints improve performance, but Oracle's responses to hints vary between releases. • Use `-noindexhint` to test whether performance for a query improves when the DataServer executes it without hints. • Use `-noindexhint` at compile or run time.
Server Join(`-nojoinbysqldb`)	• Specifies that the client evaluates and performs queries that have joins. This might slow performance, but provides results that are consistent with OpenEdge behavior. **Note:** Server joins are never performed for dynamic queries.
USE_MERGE Hint (`-nojoinhint`)	• Specifies that the DataServer does not provide USE_MERGE hint to the Oracle DBMS. Generally the USE_MERGE hint improves performance for outer joins. However, in some cases, it results in slower response time. • Use `-nojoinhint` to test whether performance for OUTER-JOIN query improves when less records are needed. • Use `-nojoinhint` at compile time only. It has no impact on run time.

Setting a session's timestamp

To record a session's time information, you have to set its timestamp. Oracle DataServer enables you to select the timestamp using the `-Dsrv` connection parameter switch, `SESSION:TIME-SOURCE`.

The following table describes the `SESSION:TIME-SOURCE` switch:

Table 44: SESSION:TIME-SOURCE

Switch	Description
`SYS`	Specifies the system timestamp for the session
`DB`	Specifies the database timestamp for the session
`UTC`	Specifies the UTC timestamp for the session
`[TZH:TZM]`	Specifies the timezone timestamp for the session

The following code would set the system timestamp for a session using the `-Dsrv` connection parameter:

```
-Dsrv session-timezone,SYS
```

The following example would set the timezone timestamp for the session using the `-Dsrv` connection parameter to +5:30:

```
-Dsrv session-timezone,+5:30
```

Analyzing application execution with Enhanced Logger

The Enhanced Logger provides a means for either standard reporting of runtime activity or logging diagnostic data for troubleshooting problems based on a set of logging characteristics. The DataServer can be configured to write information to a log file. You control the type and the degree of detail logged using a combination of log entry types and logging levels. Log entry types are categories of information written to the log file. Logging levels control the degree of logging detail for any given log entry type.

The Enhanced Logger for DataServers offers two logging contexts:

- Server context for server-specific log entries
- Client context, which shares logging information with OpenEdge clients and AppServer agents.

Note: Both the server and the client contexts for DataServers include some log entry types that work in concert with one another. You can yield a detailed understanding of the DataServer operations by merging the contexts into one view of DataServer log activity.

LOG-MANAGER system handle attributes and methods

The LOG-MANAGER system handle controls logging settings for the OpenEdge client and DataServer client contexts. The LOG-MANAGER attributes and methods allow you to programmatically control the logging levels and log entry types during the execution of your program.

The following table lists the functions of LOG-MANAGER system handle attributes.

Table 45: LOG-MANAGER system handle attributes

Attribute	Access	Function
ENTRY-TYPES-LIST	Readable	Returns a list of all valid entry types.
LOG-ENTRY-TYPES	Readable/Writable	Returns/Writes a list of one or more types of log entries.
LOGFILE-NAME	Readable/Writable	Returns/Writes the name of the log file.
LOGGING-LEVEL	Readable/Writable	Returns/Writes the level at which log entries are written.
LOG-THRESHOLD	Readable	Returns the file size threshold of log files. When the size of the current log file becomes equal to or greater than the specified size, OpenEdge renames and saves the log file and creates a new log file.
NUM-LOG-FILES	Readable	Returns the number of rolled over log files to keep on disk at any one time, for an OpenEdge session, including the current log file.
HANDLE	Readable	Returns a handle.
INSTANTIATING-PROCEDURE	Readable	Returns a procedure handle for the first, or instantiating, procedure.
TYPE	Readable	Returns the type of handle.

The following table lists the functions of LOG-MANAGER system handle methods.

Table 46: LOG-MANAGER system handle methods

Method	Function
CLEAR-LOG( )	Clears all messages in the current client log file and leaves the file open for writing.
WRITE-MESSAGE( )	Writes user messages to the current log file.
CLOSE-LOG( )	Stops an interactive or batch OpenEdge client from writing messages to the current client log file and closes the log file.

Note: The CLOSE-LOG( ) method is valid only for interactive and batch clients. WebSpeed agents and AppServer servers write a message to the server log file indicating that it is invalid to use the CLOSE-LOG( ) method to close a WebSpeed or AppServer server log file. DataServer connections established by Webspeed or AppServer agents all share the same server log file. So, you cannot execute the CLOSE-LOG( ) method unless there is only one agent with an open DataServer connection. In this case, the method returns FALSE.

For more information on the LOG-MANAGER system handle or its attributes and methods, see the LOG-MANAGER system handle reference entry in *ABL Reference*.

DSLOG-MANAGER system handle attributes and methods

The DSLOG-MANAGER system handle is similar to the LOG-MANAGER system handle. It is used to control the logging settings for the server context of the OpenEdge DataServer logging, while the LOG-MANAGER controls logging settings for OpenEdge Client and DataServer client contexts. Although both provide attributes and methods for enabling, disabling, and changing logging capabilities, the DSLOG-MANAGER applies only to DataServer components.

The DB-CONTEXT attribute must be set before any other DSLOG-MANAGER attribute can be accessed.

The following table lists the DSLOG-MANAGER system handle attributes.

Table 47: DSLOG-MANAGER system handle attributes

Attribute	Access	Function
DB-CONTEXT	Readable/Writable	Specifies the logical database name of a valid and connected data source for a supported db-type. It sets the system handle to a specific server context to enable access to other DSLOG-MANAGER attributes and methods. You can also set it to the alias of a connected data source.
ENTRY-TYPES-LIST	Readable	Returns a list of all valid entry types.
LOG-ENTRY-TYPES	Readable/Writable	Returns/Writes a list of one or more types of log entries. The default value returned by the LOG-ENTRY-TYPES attribute is the logentrytypes specified in the -Dsrv parameter when the foreign data source is connected.
LOGFILE-NAME[14]	Readable	Returns the name of the log file. By default, the DSLOG-MANAGER uses the dataserv.lg log file. You can change the file name by using the -dslog startup parameter from the OpenEdge client or the srvrDSLogFile UBROKER property from the DataServer broker instance. The server context cannot work without a log file open, so it cannot be changed at runtime. If the new log file fails to open, it might result in no log file being available.
LOGGING-LEVEL	Readable/Writable	Returns/Writes the level at which log entries are written. The default value returned by the LOGGING-LEVEL attribute is the logginglevel specified in the -Dsrv parameter when the foreign data source is connected.
LOG-THRESHOLD[15]	Readable	Returns the file size threshold of log files. When the current log file becomes equal to or greater than the specified size, OpenEdge renames and saves the log file and creates a new log file.

Attribute	Access	Function
NUM-LOG-FILES[16]	Readable	Returns the number of rolled over log files to keep on disk at any one time, for an OpenEdge session, including the current log file.
HANDLE	Readable	Returns a handle.
INSTANTIATING- PROCEDURE	Readable	Returns a procedure handle for the first, or instantiating, procedure.
TYPE	Readable	Returns the type of handle.

By default, the DSLOG-MANAGER uses the dataserv.lg log file. You can change the file name by using the -dslog startup parameter from the OpenEdge client or the srvrDSLogFile UBROKER property from the DataServer broker instance.

The following example tests for an active connection before setting the DSLOG-MANAGER to that connected DataServer context and getting a list of log entry types and setting a specific message inside the application at runtime.

```
IF CONNECTED("oradb") THEN DO:
    DSLOG-MANAGER:DB-CONTEXT = 'oradb'.
    MESSAGE DSLOG-MANAGER:LOG-ENTRY-TYPES.
    DSLOG-MANAGER:WRITE-MESSAGE ("Some message after connection").
END.
```

For more information on the DSLOG-MANAGER system handle or its attributes, see the DSLOG-MANAGER system handle reference entry in *ABL Reference*.

The following table lists the functions of DSLOG-MANAGER system handle methods.

[13] Each connection to a DataServer is its own entity. The DB-CONTEXT attribute enables you to provide different logging settings for each data source connection. The DB-CONTEXT attribute must be set to the logical database name or alias which is currently connected, prior to setting any other attributes. The attribute defaults to Unknown value (?). If DB-CONTEXT does not contain the value of a valid logical database name, any attempt to access other attributes of DSLOG-MANAGER fails.

[14] The value returned by the LOGFILE-NAME is the default name dataserv.lg or; the name specified by the -dslog client startup parameter for self-service clients; the name specified by the -dslog startup parameter when ProBroker is started; the name specified by the srvrDSLogFile attribute in the ubroker.properties file when a Unified Broker instance is started for DataServer connections.

[15] The value returned by the LOG-THRESHOLD is one of the following: the name specified by the -logthreshold client startup parameter for self-service clients; the name specified by the -logthreshold startup parameter when ProBroker is started; the name specified by the srvrLogThreshold attribute in the ubroker properties file when a Unified Broker instance is started for DataServer connections.

[16] The value returned by the NUM-LOG-FILES is one of the following: the name specified by the -numlogfiles client startup parameter for self-service clients; the name specified by the -numlogfiles startup parameter when ProBroker is started; the name specified by the srvrNumLogFiles attribute in the ubroker.properties file when a Unified Broker instance is started for DataServer connections.

Table 48: DSLOG-MANAGER system handle methods

Method	Function
`CLEAR-LOG( )`	Clears all messages existing in the current client log file and leaves the file open for writing.
`WRITE-MESSAGE( )`	Writes user messages to the current log file.

Note: The `CLOSE-LOG( )` is not a supported `DSLOG-MANAGER` system handle method. You cannot execute the `CLOSE-LOG( )` method unless there is only one brokered DataServer server spawned by the Unified Broker or by ProBroker.

Log entry types

Log entry types represent categories of information you can write to a log file. You can categorize the information into server and client contexts. The log entry types in the client context are applicable to DataServer clients, ABL clients, AppServer agents and WebSpeed agents. However, the log entry types in the server context are applicable only to DataServer brokers.

The degree of detail written to the log file is determined by the logging level. For more information on logging levels, see Logging levels on page 191.

The following table lists the DataServer log entry types in the client context.

Table 49: DataServer Log entry types (Client context)

Log entry type	Description
`DS.QryInfo`	Logs messages about client processing for SQL queries. By default, setting the `DS.QryInfo` log entry type in the client context will automatically set the `SQL` log entry type in the server context.
`DS.Cursor`	Logs client-side cursor details. By default, setting the `DS.Cursor` log entry type in the client context will automatically set the `Cursor` log entry type in the server context.
`4GLTrans`	Logs client-side transaction details. Level 4 of the entry type writes client context information to the log file specific to DataServers. Output from the `4GLTrans` log entry type can supplement output of the `TRANS` log entry type in the server context.
`DB.Connects`	Logs client-side connection details. Level 3 of this log entry type logs the `-Dsrv` connection switch values in the server context.
`DS.Performance`	Logs client-side performance details. Levels 2,3, and 4 of the performance entry type generate log outputs at the session level, connection level, and individual round trip levels respectively.

For more information on log entry types of the DataServer client context and other log entry types specific to the client context, see *Troubleshoot ABL Applications*.

The following table lists the DataServer log entry types in the server context.

Table 50: DataServer Log entry types (Server context)

Log entry type	Description
SQL	Logs information about SQL queries. Similar to the legacy qt_debug,SQL option. • Level 2 (Basic) — Logs details of SQL query preparation and execution. • Level 3 (Verbose) — Adds bound values when prepared and/or executes SQL utilize bind parameters.
Cursor	Logs information about cursor details. Similar to the legacy qt_debug,Cursor option. • Level 2 (Basic) — Logs basic cursor details such as table number, client and server cursor identifiers.
Trans	Logs information about transaction details. • Level 2 (Basic) — Logs messages when a SQL transaction begins or ends on the server or when a transaction is undone against the MS SQL Server and Oracle database. • Level 3 (Verbose) — Adds messages for the beginning and ending of sub-transactions and when a transaction is undone. • Level 4 (Extended) — Logs details about records locked in a transaction including the type of lock, lock upgrades, etc.
Connects	Logs details of connection attributes and related information. Mapped to the connection information formerly output with legacy qt_debug,SQL and qt_debug,Cursor. • Level 2 (Basic) — Logs details about connection parameters set before and after connection. Logs connection pooling information for firehose cursors. Assigns a connection ID for connection identification in the log file.
Performance	Logs performance details in the server context.

For more information on log entry types of the DataServer server context and other log entry types specific to the server context, see *Troubleshoot ABL Applications*

The following table identifies the logging levels and entry types for client and server contexts.

Table 51: Setting logging levels and log entry types

Operation Mode	Logging Option	OE Client	MSS/Oracle DataServer
Startup / connection	Logging level	`-logginglevel, parameter`	`-Dsrv logginglevel, parameter`
	Log entry types	`-logentrytypes, parameter`	`-Dsrv logentrytypes, parameter`
	ubroker.properties[17][18]	`-logginglevel, parameter -logentrytypes, parameter`	`-Dsrv logginglevel -Dsrv logentrytype`
Runtime setting	Logging level	`LOG-MANAGER: LOGGING-LEVEL attribute`	`DSLOG-MANAGER: LOGGING-LEVEL attribute`
	Logging level	`DSLOG-MANAGER: LOG-ENTRY-TYPES attribute`	`DSLOG-MANAGER: LOG-ENTRY-TYPES attribute`

The following code example shows how logging levels and log entry types are set using the client startup switch:

```
-logentrytypes DS.Cursor:3
-logentrytypes DS.Cursor -logginglevel 3
```

In the following example, logging levels and the log entry type are set using the LOG-MANAGER system handle:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DS.Cursor:3"
```

In the following example, the logging level and log entry types are defined using the `-Dsrv` switch.

```
-Dsrv logentrytypes,SQL:3
-Dsrv logentrytypes,SQL,logginglevel,3
```

[17] logginglevel and logentrytypes specified in the ubroker properties file are for AppServer agents in the client context only.

[18] To specify logginglevel and logentrytypes at startup for the DataServer's server context, continue to use the -Dsrv startup/connection options. logginglevel and logentrytypes specified in the ubroker properties file for a DataServer broker instance are ignored. The -Dsrv options specified within the srvrStartupParam of a DataServer broker instance are also ignored.

The following example demonstrates the use of a DSLOG-MANAGER system handle with a DataServer DB-CONTEXT attribute.

```
DSLOG-MANAGER:LOG-ENTRY-TYPES = "SQL:3"
```

Logging levels

You can choose one of five logging levels, as shown in the following table.

Table 52: Logging levels

Logging level	Description
0 (None)	Logs no entries. This is equivalent to turning logging off.
1 (Errors)	Logs OpenEdge error messages. This includes all error messages and is unrelated to the entry types specified. Errors continue to be logged at all higher levels.
2 (Basic)	Log entry type determines the logged information. Each log entry type generates at least some output. This is the default.
3 (Verbose)	Log entry type determines the logged information.
4 (Extended)	Log entry type determines the logged information.

Setting the DataServer logging levels and log entry types

Client context log settings can be set using:

- Start-up parameters at client startup
- `LOG-MANAGER` system handles in client applications

Server context log settings can be set using:

- The `-Dsrv` switch at client startup or during a database connection
- `DSLOG-MANAGER` system handle in client applications

Setting all other DataServer log settings (`-clientlog`, `-dslog`, `-numlogfiles`, `-logthreshold`, and `-logappend`)

- Start-up parameters at client startup. These settings are shared with the server context when DataServers are connected self-service.

Note: Of these, `-dslog` is not shared but simply passed onto the server.

- `Ubroker.properties` specified for agents at AppServer startup.

Note: AppServer client context does not include the ability to forward a DataServer log file name onto the server context.

- LOG-MANAGER system handles in client applications.

Note: Client context does not include the ability to forward a DataServer log file name onto the server context.

Server context settings can be set using:

- ProBroker startup parameters (-dslog, -numlogfiles, -logthreshold, -logappend)
- Unified-Broker DataServer instance attributes (srvrDSLogFile, srvrNumLogFiles, srvrLogThreshold, srvrLogAppend)

Note: DSLOG-MANAGER system handle can be used in client applications to return the above settings but cannot modify them.

For more information on LOG-MANAGER and DSLOG-MANAGER startup parameters, see the LOG-MANAGER and DSLOG-MANAGER entriesin*ABL Reference* and *Troubleshoot ABL Applications*.

Server Context Log File Naming

When no log file name is specified for the server context, its default name continues to be dataserv.lg in the working directory, just as it was using the legacy qt_debug logging options.

Both OpenEdge client startup and ProBroker startup allow the server context log file to be specified with the -dslog parameter. For a DataServer broker instance specified to the Unified Broker, the srvrDSLogFile attribute in the ubroker.properties file specifies the server context log file name.

If the server context log file name is specified without any path, it is placed in the working directory. If it is specified with a relative path name, it searches for that relative path under the absolute path of the working directory unless the $DSLOGDIR environment variable is set. When $DSLOGDIR is set, it overrides the working directory as the root path for placing the server log file. The relative path name of the specified log name is combined with this absolute path to place the server context file. If you specify a full absolute path for the server context log file name, it is used exclusively to set the log file path.

The values for DSLOGDIR and -dslog can be set as described in the following table:

Table 53: Parameters for creating log file

DSLOGDIR value	-dslog value	Result
An existing directory	dslog name	dslog file is created in the DSLOGDIR
An existing directory	Qualified path and dslog name	dslog file is created in the dslog path
An existing directory	Not set	Log file with default name is created in DSLOGDIR
A new/non-existent directory	dslog name	dslog file is created in working directory
Not set	Qualified path and dslog name	dslog file is created in the dslog path

DSLOGDIR value	-dslog value	Result
Not set	Not set	Log file with default name is created in working directory
Not set	A new/non-existent path and dslog name	Log file with ds_ prefix is created in working directory
Not set	dslog name	dslog file is created in working directory

Creating a unique log file per connected client

By default, the log file started in the server context is shared by all client/server connections started under a specific broker.

You can optionally set another environment variable at startup called `$UNIQUE_DSLOG` if you wish to have a unique log file name for each client session.

When the `$UNIQUE_DSLOG` environment variable is set on, the log file name starts with `ds_` followed by the log file name specified for the server context followed by the server's process ID number. The file extension continues to be `.lg`. The full name syntax is as follows:

```
ds_<-dslog or srvrDSLogFile name>_<pid>.lg.
```

Enabling ABL to SQL Correlation in Logging

The Enhanced Logger allows ABL to SQL correlation so that you can compare ABL database requests with SQL directives that are executed on the server.

In the DataServer log file, ABL to SQL correlation includes information from the client context, the ABL module and line number, along with the server context SQL that was generated based on the ABL code and run against the foreign data source. The correlation is enabled using the `-Dsrv PRGRS_LOG_ABL,n` switch where `n` ranges from `0` to `2`, as demonstrated in the following example:

```
-Dsrv logentrytypes,SQL,PRGRS_LOG_ABL,2
```

For a DataServer broker instance, the `SQL` log entry type can be set by using the `-Dsrv logentrytypes,SQL` switch in the `'srvrStartupParam'` property under the **UBroker.MS** section in **OpenEdge Explorer** or `ubroker.properties` files or using the parameter file .pf.

The following table describes the PRGRS_LOG_ABL connect option values:

Table 54: PRGRS_LOG_ABL connect option values

Option Value	Description
0	Disables the ABL to SQL correlation feature. No correlation information will be logged.

Option Value	Description
1	Enables the ABL to SQL correlation feature for SQL-related log entry types. Every SQL request derived from the log entry type output is preceded by ABL filename and line number which can be correlated to a compiled listings file.
2	Enables a superset of value 1 which, along with correlation information, includes visual delimiters marking the start and end of each ABL file module.

Note: The `SQL` log entry type is specifically designed to output SQL-related logging information and therefore is used most commonly in conjunction with the `PRGRS_LOG_ABL -Dsrv` switch. However, the `PRGRS_LOG_ABL` switch can be enabled to acquire module and line number information with any log entry type that writes SQL request information to the log.

The ABL to SQL correlation is logged in the `dataserv.lg` file where the log entries will contain the ABL module and the line number where the ABL procedure started. It will also contain information about the SQL query executed on the foreign data source and the visual delimiters.

The following ABL constructs log SQL query information:

- `FIND` statements
  - `FIND FIRST`
  - `FIND NEXT`
  - `FIND LAST`
  - `FIND PREV`
  - `FIND CURRENT`
  - `FIND rid`
- `Dynamic FIND` statements
  - `FIND-BY-ROWID`
  - `FIND-CURRENT`
  - `FIND-FIRST`
  - `FIND-LAST`
  - `FIND-UNIQUE`
- FOR statements
  - `FOR EACH`
  - `FOR FIRST`
  - `FOR LAST`
- QUERY statements
  - `OPEN`
  - `GET-LAST`
  - `GET-NEXT`
  - `GET-PREV`
  - `GET-FIRST`
  - `REPOSITION-TO-ROWID`
  - `REPOSITION-BACKWARD`
  - `REPOSITION-FORWARD`
  - `REPOSITION-TO-ROW`
  - `QUERY-OPEN method`
- `CREATE` statement
- `UPDATE` statement
- `DELETE` statement
- `IMPORT` statement
- `EXPORT` statement
- `RUN STORED-PROCEDURE` statement
- `SQL Pass-thru` statement
- `RELEASE` statement
- `VALIDATE` statement
- `ASSIGN` statement
- `END` statement
- `CURRENT-VALUE` statement
- `NEXT-VALUE` statement
- `SET-VALUE` statement
- `DYNAMIC-CURRENT-VALUE` statement
- `DYNAMIC-NEXT-VALUE` statement
- `CAN-FIND` function
- `RECID` function
- `ROWID` function
- `ProDataSet FILL, SAVE-ROW-CHANGES, MERGE-CHANGES` methods
- `DO` transaction

Using PRGRS_LOG_ABL switch to log ABL to SQL correlation

This example shows how the ABL to SQL correlation is logged in the dataserv.lg file, when the FOR EACH statement is compiled.

```
FOR EACH Customer:
    DISP NAME.
END.
```

The log output shown below, includes the ABL module name and the line number where the ABL procedure started in correlation to the SQL queries executed on the foreign data source. Visual delimiters are also logged for clarity in the log files.

```
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL
--------->{START} [C:\OpenEdge\WRK\p18695_DSexample.ped]
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL ABL statement
reference: C:\OpenEdge\WRK\p18695_DSexample.ped line 0
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL SELECT
/*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM HARI.CUST
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL ABL statement
reference: C:\OpenEdge\WRK\p18695_DSexample.ped line 0
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL OMER
T0 WHERE PROGRESS_RECID = :rid
[11/10/04@02:15:46.582+2300] P-007956 T-006300 2 ORA SQL the SQL
statement SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
HARI.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
[11/10/04@02:15:46.584+2300] P-007956 T-006300 2 ORA SQL ABL statement
reference: C:\OpenEdge\WRK\p18695_DSexample.ped line 1
[11/10/04@02:15:46.584+2300] P-007956 T-006300 2 ORA SQL SELECT
/*+ INDEX_ASC(T0 CUSTOMER##COUNTRYPOST) */ PROGRESS_RECID
```

For more information about the PRGRS_LOG_ABL switch, see *Troubleshoot ABL Applications*.

qt_debug logging

The existing qt_debug logging only logs server information. This option is available only for backward compatibility and no further logging enhancements will be done in the current and future releases.

The qt_debug option of the DataServer (-Dsrv) startup parameter (and the QUERY-TUNING DEBUG phrase) instructs the DataServer to print information on the queries it generates to the dataserv.lg log file. The qt_debug option provides extended diagnostic capabilities that you can use to determine which parts of your application might be causing additional network traffic or processing by Oracle. The syntax for the qt_debug options is:

```
CONNECT <dbname> -U <username> -P <password>
-Dsrv qt_debug,<option>[,qt_debug,<option>...]
```

The default setting is for all debugging information to be turned off (-qt_no_debug). Specify qt_debug,*option* to override the NO-DEBUG default.

The following table lists the diagnostic capabilities of qt_debug.

Table 55: Diagnostic options

Option	Description
SQL	Specify qt_debug,SQL to print the SQL that the DataServer executes against the Oracle DBMS in the dataserv.lg file.
EXTENDED	Specify qt_debug,EXTENDED to print the SQL information that the SQL option generates, plus additional information such as cursor statistics in the dataserv.lg file.
CURSOR	Information about the Oracle cursors that the DataServer uses for internal Oracle Call Interface (OCI) calls and for opening queries. It tracks when cursors open, close, and when the DataServer reuses them. It also summarizes each cursor's activity. These diagnostics are especially helpful when determining ABL and Oracle maximum cursor settings or cursor "leaks" that your application might have.
DATA-BIND	Information about the data types, buffer sizes, and addresses that the DataServer and Oracle use when binding variables.
PERFORMANCE	Information on the time certain operations take. These statistics are available only for some platforms. Note that any time differences between what the DataServer reports and what the Oracle tkprof utility reports might be due to network performance issues, rather than to DataServer or Oracle behavior.
SUMMARY	Information on cursors and timing in summary form as an OpenEdge data (.d) file. This option is not available as a QUERY-TUNING phrase option.
VERBOSE	Information on all of the above. This option generates a very large log file. Remember to clear your log file before using this option to test a procedure.

Note: Turning on debugging options decreases DataServer performance. Be sure to turn off debugging options when running DataServer applications in production mode.

This connection statement causes the DataServer to report on the time that certain operations take:

```
CONNECT oradb -U user -P password
  -Dsrv qt_cache_size,32000,QT_BYTE,qt_debug,PERFORMANCE.
```

Index cursors

The OpenEdge Index Cursor (-c) connection parameter sets the maximum number of Oracle cursors that the DataServer client session uses when you connect to an Oracle database. Specify -c after you specify the name of the Oracle database (-db*database-name*) in the list of parameters.

The DataServer uses cursors whenever it executes an SQL statement to access data in a table. Each Oracle cursor uses up to 4K of memory. To minimize memory consumption, the DataServer attempts to free and reuse Oracle cursors as soon as possible. It also reuses cursors that are active (not free) if there are no free cursors available. This might reduce performance, but it allows the application to continue even if there are not enough cursors. ABL uses a least-recently-used algorithm to select which active cursor to reuse.

Oracle allows you to set the maximum number of cursors in your `init.ora` file using the `OPEN_CURSORS` parameter. The valid range for numbers of cursors varies depending on the version of Oracle and system configuration.

ABL also maintains a default maximum number of Oracle open cursors as 50 for the DataServer. When you use the `-c` parameter to set the maximum number of cursors, you cannot exceed the number that your `init.ora` file specifies. For example, if the Oracle `OPEN_CURSORS` parameter is set to 250, then you can set the upper limit for maximum open cursors open to 250 with the `-c` parameter.

Determining the optimal number of cursors for your application involves balancing memory consumption, performance, and possible application failures. Use the `-Dsrv qt_debug,EXTENDED` parameter to log information on how many cursors your application uses. The following excerpt from the `dataserv.lg` file shows the cursor handler identifier within the angle brackets (<*n*>) that the DataServer uses for each OCI call:

```
OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.CUSTOMER
          T0 WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <2>
OCI call omru   <2>
OCI call OCIHandleAlloc <0>
OCI call OCIStmtPrepare <3>     sqlcrc = 60664
     SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ * FROM DOCTEST.ORDER_ T0
          WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <3>
OCI call omru   <3>
OCI call OCIHandleAlloc <0>
OCI call OCIStmtPrepare <4>     sqlcrc = 56980
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##COUNTRY_POST)  */ PROGRESS_RECID
          unique_id_0,CUST_NUM,COUNTRY,NAME,ADDRESS,ADDRESS2,CITY,STATE,
          POSTAL_CODE,CONTACT,PHONE,SALES_REP,CREDIT_LIMIT,BALANCE,TERMS,
          DISCOUNT,COMMENTS,PROGRESS_RECID FROM DOCTEST.CUSTOMER T0
OCI call OCIStmtExecute <4>
OCI call OCIHandleAlloc <0>
OCI call OCIStmtPrepare <5>     sqlcrc = 48586
     SELECT /*+ INDEX_ASC(T0 ORDER_##CUST_ORDER)  */ PROGRESS_RECID un
     ique_id_0,ORDER_NUM,CUST_NUM,ORDER_DATE,SHIP_DATE,PROMISE_DATE,CA
     RRIER,INSTRUCTIONS,PO,TERMS,SALES_REP,PROGRESS_RECID FROM DOCTEST
     .ORDER_ T0 WHERE (CUST_NUM = :1)
OCI call OCIStmtExecute <5>
```

Avoid setting the `-c` parameter too low or too high:

- **Too low** — A low setting can cause unnecessary recompiles of SQL, which hurts performance. Your application could also fail because it opens more queries, nested `FOR EACH`, or `FIND` statements, that reference different indexes, than the `-c` parameter allows.
- **Too high** — A high setting can cause unnecessary consumption of resources such as memory and cursors, which can hurt performance when they are not reused. Your application can also fail when you allocate all available cursors, including a cursor that the Oracle DBMS needs for internal purposes. If this occurs, Oracle returns a recursive SQL error.

In case your application exceeds the maximum value specified for the `-c` parameter, you can use the close inactive cursor operation. For more information on closing inactive cursors, see RUN STORED-PROC statement with special internal option on page 93 and Closing inactive cursors on page 128.

Local schema caching

The ability to store schema definitions in a local file allows you to access them more quickly. Running DataServer applications with a local schema cache in addition to a schema holder results in better performance.

An ABL `SAVE CACHE COMPLETE` statement creates a binary file that contains the entire schema for an OpenEdge database. Use the following syntax to create a cache file for a connected schema holder:

```
SAVE CACHE COMPLETE schema-holder-name TO filename.
```

For example, the following statement creates a cache file named `ocache` for the `oholder` schema holder:

```
SAVE CACHE COMPLETE oholder TO ocache.
```

To use the cache file for a schema holder, specify the `-cache` parameter and the cache filename when you connect to the schema holder. For example, the following `CONNECT` statement connects the Oracle database with the schema holder and tells ABL to use the cache file:

```
CONNECT oholder -RO -db oradb -U scott -P tiger -cache ocache.
```

If you make any changes to the schema holder, create a new cache file for the schema holder.

For more information, see *Develop ABL Applications* and the "SAVE CACHE Statement" reference entry in *ABL Reference*.

Connection failures and OpenEdge responses

The following table lists circumstances under which a connection might fail and describes the responses you might receive.

Table 56: Failure responses

Failure circumstances	Results
During startup	The system displays an error message and returns to the operating system prompt.
During a `CONNECT` statement	The system aborts the remainder of the `CONNECT` statement (as though the remainder never existed) and a run-time error condition occurs. Any connections you made previous to the failed connection remain in effect. You can use the `NO-ERROR` modifier with the `CONNECT` statement to trap run-time errors. If you use the `NO-ERROR` modifier and it fails, you see the same failure behavior as you do with an unsuccessful `CONNECT` statement. However, run-time error conditions do not occur.
During an attempted auto-connect	The system aborts the remainder of the connect (as though the remainder never existed) and a run-time error condition occurs. Any connections you made previous to the failed connection remain in effect. There is no way to trap auto-connect run-time error conditions.

Failure circumstances	Results
During an attempt to connect using the Data Dictionary	The Data Dictionary displays an error message and returns to the main window.
During an attempt to connect a connected database with a different logical name	The system responds with a run-time error condition and you can continue. You can use the `NO-ERROR` modifier to suppress the error.
During an attempt to connect a connected database with the same logical name	The system responds with a warning and you can continue. You can use the `NO-ERROR` option to suppress the warning.
During an attempt to connect an unconnected database, when the logical name is already in use by a connected database	The system responds with a run-time error condition and you cannot connect to the second database.
During an attempt to connect to an Oracle database that uses a double-byte character set	The system responds with an error that the code page of the database and the `-cpinternal` value differ. You also receive messages about missing conversion tables.

Connection troubleshooting

Some reasons that a connection attempt to an Oracle database might fail are:

- The OpenEdge schema holder is not connected.
- The Oracle process is not running, or not running correctly. Try to connect with SQL*PLUS to check the status of the Oracle RDBMS.
- The value of the `ORACLE_SID` environment variable that started up the Oracle process is different from the current value of `ORACLE_SID`.
- The user ID and password combination you provided during connection is invalid for the Oracle database.
- For a remote DataServer connection failure, you didn't include the correct name of the host and server in the network. Be sure to enter the same server name you used when you started the broker.
- Error messages about mismatched code pages are caused when `-cpinternal` and `-cpstream` do not match the code page that the Oracle database uses.

Accessing the DataServer log using qt_debug logging options

OpenEdge supports a log file named `dataserv.lg` that is dedicated to tracking information related to DataServers. Information about the processes for all DataServers is located in this single file. The log file provides a useful record of connection and disconnection processes and error messages that you can use to diagnose problems or failures. In addition, for platforms that dynamically load the Oracle shared library, information about the library is stored in the log file. For each process, the log provides the process identification number, physical database name, database type, user ID, and a message about the status of the process. Here is an excerpt from a sample DataServer log file:

```
                Wed Oct 29 12:28:53 2007
12:28:53 (pid 242) : Remote ORACLE server begin. (1897)
12:28:53 (pid 242) : Executable was compiled on Sep  2 2003 at 20:11:23 (5067)
12:28:53 (pid 242) : Login to dataserver db as user X. (2689)
12:28:54 (pid 242) : ORASRV version 12 (6440)
12:28:54 (pid 242) : Using shared library c:\orant\Oracle901\bin\OCI.DLL (oci 8.1)
12:28:54 (pid 242) : ORACLE_HOME set to 'c:\orant\oracle901'
12:28:55 (pid 242) : -Dsrv qt_debug: 0 (0x0 None) (6489)
12:28:55 (pid 242) :   Cursor limit: 84 (-c setting) (6490)
```

Note: The log output sample shown above is generated using the `qt_debug` logging option and not the Enhanced logger.

The query-tuning `DEBUG` option causes the DataServer to write information about the SQL that it generates to the `dataserv.lg` file as well. You can access the DataServer log file using the host machine.

Naming the log file

The DataServer log file name, the number of log files, the log threshold, and the log append switch settings are derived from their respective client startup switches: `-dslog`, `-numlogfiles`, `-logthreshold` and `-logappend`. For example:

- In **self-service mode**, `-dslog`, `-numlogfiles`, `-logthreshold`, and `-logappend` startup switches apply to both client and DataServer log files
- In **client-server mode**, the `-dslog`, `-numlogfiles`, `-logthreshold`, and `-logappend` settings are derived from the broker startup
- In **ProBroker**, these are the `-dslog`, `-numlogfiles`, `-logthreshold`, and `-logappend` startup parameters from the `_probrkr` command line
- In the **Unified Broker**, these are `srvrDSLogFile`, `srvrNumLogFiles`, `srvrLogThreshold`, and `srvrLogAppend` ubroker properties

All the above attributes are static in nature. They are set at startup and cannot be modified once established. They are read-only attributes when requested from the `DSLOG-MANAGER` runtime system handle.

Use the `-dslog` client startup parameter to specify the DataServer log file name. If `-dslog` is not specified, then the DataServer chooses the default log file name of `dataserv.lg`.

Setting the log file location

You can specify the absolute path of the log file name with -dslog startup switch to set the location of the log file. The absolute path is obtained from the $DSLOGDIR environment variable. So for example, setting $DSLOGDIR="C:\OE\Wrk\logs" and setting -dslog to mydatasrv.lg, yields the target server log file name of C:\OE\Wrk\logs\mydatasrv.lg. If $DSLOGDIR is not set and just the log file name is specified, then the log file is created in the directory where the OpenEdge session (client or agent) is started.

The path for the log file name is derived in the following order of preference.

1. Usage precedence is for -dslog with an absolute path name.
2. On specifying -dslog with relative path than path name, the -dslog specified is relative to the $DSLOGDIR path.
3. If -dslog is a relative path and no value has been specified for $DSLOGDIR, then the log file will located in the working directory where the DataServer server component is started. This can be:
   a) The current directory where the OpenEdge client is started, if using an OpenEdge client with a self-service DataServer.
   b) The working directory of your AppServer broker instance when using a self-service DataServer in the Unified Broker framework.
   c) The working directory of your DataServer broker instance when using a client-server DataServer in the Unified Broker framework.
   d) The current directory where you started your traditional broker using a client-server DataServer with ProBroker.

   If neither -dslog nor $DSLOGDIR are specified, the default log file name dataserv.lg is used with the same directory path in the order of precedence as previously mentioned.

Handling log file creation errors

If creation of the DataServer log file fails for any reason, an error is reported and the DataServer continues to execute. If an error occurs, the DataServer will derive its log file name as ds_<pid>.lg where pid is the process ID of the server process. The DataServer continues to write log messages to ds_<pid>.lg log file. For example, if you don't have write privilege on the $DSLOGDIR directory, the ds_<pid>.lg file is created in the directory where the client or broker agent is started. This behavior is optionally available by default by setting environment variable $UNIQUE_DSLOG at startup.

Restrictions on dynamic changes to log file

When the client and server share the same log file in self-service mode, the LOG-MANAGER system handle is available to allow the log file to be managed. However, when self-service DataServers don't share the same log file with the client, or if the DataServer is run in client-server mode, the server log file is likely to exist among other servers writing to the same log simultaneously. For this reason, dynamic changes to the log file, such as renaming or closing the log file, are restricted for all server log file contexts. The one exception is the CLEAR-LOG(_) function which is allowed only if no other process than the one initiating the CLEAR-LOG(_) is writing to the log file simultaneously.

Using the $UNIQUE_DSLOG environment variable

You can use the $UNIQUE_DSLOG environment variable to isolate the log output of a given server instance and prevent the sharing of log files resources. When the $UNIQUE_DSLOG variable is set, the designated DataServer log file name becomes a root name. Specifically, it becomes the prefix of a log file name that is followed by the process ID `<pid>` of the running server process - which is then suffixed with the familiar `.lg` extension if the server name is `dataserv`. This feature can be disabled by unsetting the `UNIQUELOG` environment variable.

With a unique name, the server executable can manage its own log file copy. However, the `DSLOG-MANAGER` system handle continues to disallow changes to the log file name or closing of the file at runtime. This restriction ensures that the DataServer always has an open log file to which it can write information. The legacy `qt_debug` logging mechanism relies on this being the case.

For information on managing log settings, see Analyzing application execution with Enhanced Logger on page 182.

To access the DataServer log file on the host machine:

1. Before starting up the broker, set the `DSLOGDIR` environment variable or logical to the name of the directory where you want to place the log file.

   Note: If you do not set the environment variable, OpenEdge writes the information to the `dataserv.lg` file in the current directory of the process. If OpenEdge cannot open this file, it writes the information to the `ds_process-id.lg` file in the current directory of the process, and you will have one file per process.

2. Open the `dataserv.lg` file to read the DataServer log.

7

The DataServer Tutorial

This chapter presents step-by-step instructions for tasks associated with the DataServer.

For details, see the following topics:

- Oracle Utilities menu
- Managing server attributes
- Creating a schema image
- Updating a schema image
- Verifying a schema image
- Changing logical name and connection information
- Modifying a schema image
- Changing a code page
- Deleting a schema image
- Migrating an OpenEdge Database to Oracle with PRO/SQL
- OpenEdge DB-to-Oracle utility
- OpenEdge DB-to-Oracle Incremental Schema Migration utility
- Adjust an Oracle schema image to an OpenEdge database

Oracle Utilities menu

The tools you need to create and maintain your Oracle DataServer schema are found on the **DataServer** > **Oracle Utilities** menu. The following table briefly describes the contents of the menu. The sections which follow will describe each item in detail.

Table 57: OpenEdge DataServer for Oracle Utilities menu

Menu item	Function
Create DataServer Schema...	Creates a schema image for an Oracle database
Update/Add Table Definitions...	Updates the schema image to reflect any changes you made to the Oracle data definitions
Verify Table Definition...	Verifies that the data definitions in the schema image match your Oracle data definitions
Edit Connection Information...	Changes connection information or the logical database name in a schema image
Change DataServer Schema Code Page...	Changes the code page in a schema image
Server Attributes	Provides an interface to create, view, and maintain server attributes, and define constraint definitions in your OpenEdge database that can be migrated to the server. Constraint definitions are applied to the columns and indexes of the foreign database during migration and become metadata in your OpenEdge schema holder
Delete DataServer Schema...	Deletes a schema image
Run ORACLE SQL *Plus...	Modifies your Oracle database with SQL*PLUS commands
Schema Migration Tools	Accesses utilities for migrating an OpenEdge database to an Oracle database, incrementally migrating a schema to Oracle, adjusting a schema image, and performing a bulk load of data

When you access an Oracle utility, you might see a dialog box for verifying your user ID and password. Choose `OK`, or enter a new user ID and password with the privileges required for creating and updating a schema image. See The schema holder on page 24 for more information.

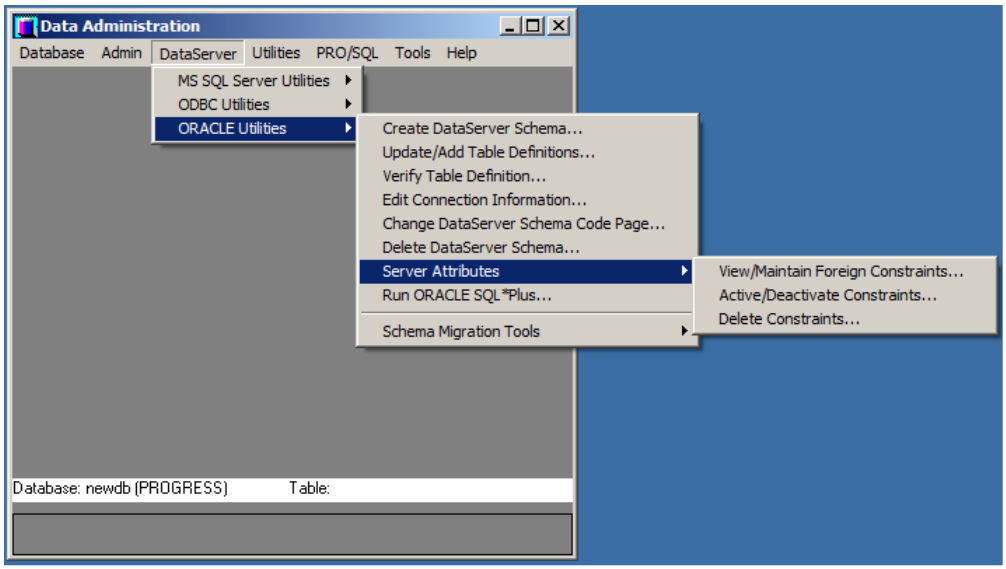
Note: The DataServer makes an additional connection to the schema holder when it creates, updates, or verifies a schema image. If you are connecting to multiple databases or schema images, make sure that you set the Number of Databases (`-h`) parameter to a value that accommodates the number of databases and schema images plus one.

Managing server attributes

In OpenEdge Release 11, the DataServer for Oracle is enhanced to enable you to define foreign RDBMS constraint definitions within the meta schema of your OpenEdge database for use in migrating your database to a foreign data source. Once migrated to the foreign data source, constraint definitions can be pulled back into your OpenEdge schema holder.

To achieve this, OpenEdge, provides a submenu for the Oracle DataServer Utilities for creating constraint definitions. The following figure shows the Server Attributes submenu.

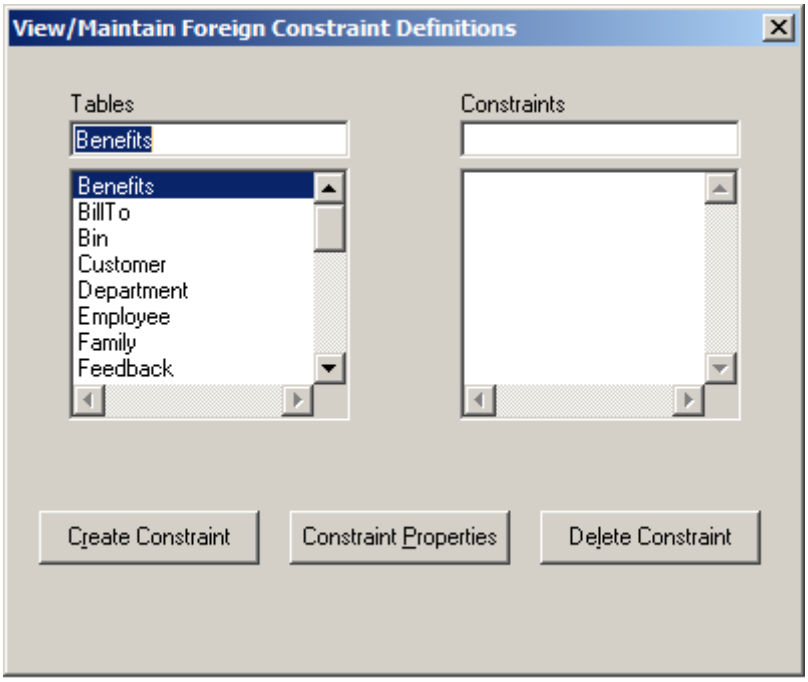
Figure 10: Server Attributes submenu



From the **Server Attributes** submenu, you can access dialog boxes that provide an interface to create, view, and maintain server attributes, and define constraint definitions applied to the data and indexes of the foreign database within your OpenEdge schema holder. The dialog boxes accessed from the **Server Attributes** submenu are as follows:

- **View/Maintain Foreign Constraints** — Use the **View/Maintain Foreign ConstraintDefinitions** dialog box, shown in the following figure , to define and delete constraint definitions as well as view constraint definition properties.

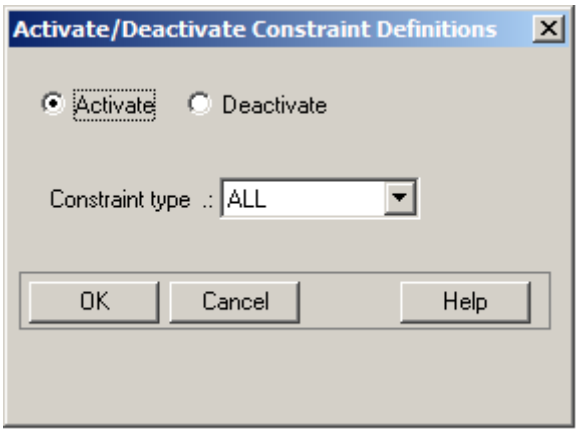
  Figure 11: View/Maintain Foreign Constraint Definitions dialog box



  See Defining constraint definitions in Data Administration on page 210 for details on how to define constraint definitions. See Modifying an existing constraint definition on page 216 for details on how to maintain constraint definitions.

- **Activate/Deactivate Constraints** — Use the **Activate/Deactivate Constraint Definitions** dialog box, shown in the following figure , to activate or deactivate constraint definitions you defined.

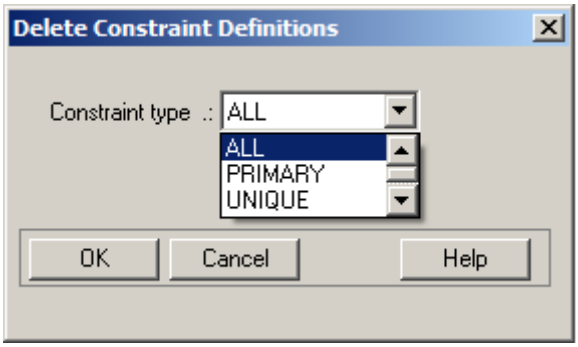
  Figure 12: Activate/Deactivate Constraint Definitions dialog box



  See Activating or deactivating a class of existing constraint definitions on page 216 for details on how to activate and deactivate constraint definitions.

- **Delete Constraints** — Use the **Delete Constraint Definitions** dialog box, shown in the following figure , to delete defined constraint definitions.

  Figure 13: Delete Constraint Definitions dialog box



  See the for details on how to delete constraint definitions.

Overview of foreign database constraint definitions

A constraint definition is a property assigned to one column or a set of columns to guarantee data consistency and integrity. You can define the following constraint definitions in your OpenEdge meta schema which you want to migrate to a foreign data source:

- **Primary key constraint definition** — Ensures that each row in a table is uniquely identifiable. Primary keys require a value that is not NULL or NULL-capable. There can only be one primary key, and one primary key constraint definition per table.
- **Unique key constraint definition** — Enforces uniqueness in a column or set of columns. You can have multiple unique key constraint definitions per table.
- **Foreign key constraint definition** — Prevents record updates that break foreign key relationships between tables.
- **Check constraint definition** — Defines an evaluation expression for data validation during an update to a field or insertion of a new row.

Constraint definition names

Within **Data Administration**, constraint definitions are given default names as defined in the following table.

Table 58: Default constraint definition names

Constraint definition type	Default constraint definition name
Primary key	_PKC_ *table-name-index-name-* idx
Unique key	_UKC _*table-name-index-name-* idx
Foreign key	_FKC _*parent-index-child index-* idx
Check	_CC _*table-name-field-name-* idx

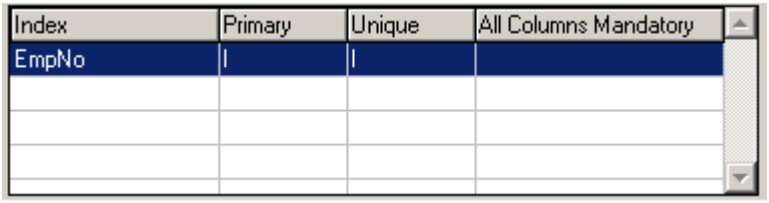
Defining constraint definitions in Data Administration

Use the procedures described in the following sections to define constraint definitions in **Data Administration**:

- Defining a primary key constraint definition on page 210
- Defining a unique key constraint definition on page 212
- Defining a foreign key constraint definition on page 213
- Defining a check constraint definition on page 214

The dialog boxes for defining primary, unique, and clustered index constraint definitions contain a common section that displays current settings for both OpenEdge properties and defined constraint definitions, a sample of which is shown in the following figure .

Figure 14: Index properties



Possible values for each field are:

- **C** — A **C** in a column indicates that a constraint definition of this type is currently defined. The constraint definition can be created in the foreign database if you configure your migration to migrate constraint definitions.
- **I** — An **I** indicates that the property is currently defined your OpenEdge database.

Defining a primary key constraint definition

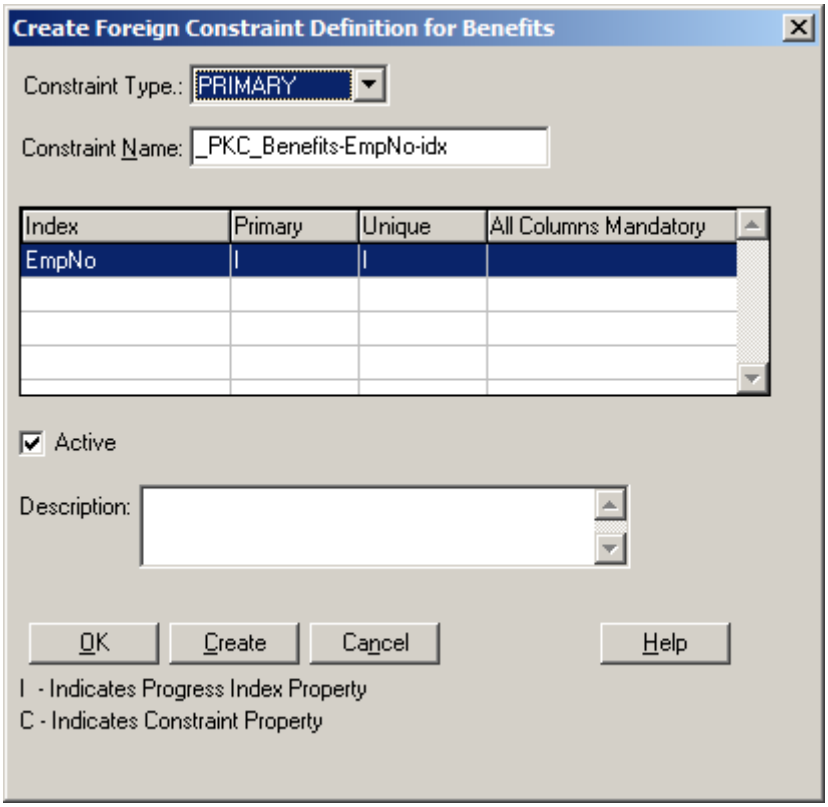
The following procedure describes how to define a primary key constraint definition.

To define a primary key constraint definition:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**. The **View/Maintain Foreign Constraints** dialog box appears.

3. Select the table for which you are defining a primary key constraint definition and click **Create Constraints**. The **Create Foreign Constraint Definition for** *tablename* dialog box appears, as shown in the following figure .

Figure 15: Create Foreign Constraint Definition for tablename dialog box



4. Accept the default constraint definition name, or enter a new one.
5. Select an index for the primary key constraint definition. Only indexes with an **I** in the **All Columns Mandatory** column are recommended for the primary key constraint definition. As long a column is unique, you can select as the primary constraint. Note, however, that the columns designated as the primary constraint are migrated as `NOT NULL` capable. That is, if you store many records in your OpenEdge database with the unknown value, they cannot be migrated to the foreign data source, given the defined primary constraint on that server object.
6. By default, the primary key constraint definition has the **Active** check box checked. Leave this box checked if you want the constraint definition you are defining to be available for migration.
7. Optionally, add a description of the constraint definition in the **Description** box.
8. Click **OK** to save the constraint definition and exit the dialog box. Click **Create** to save the constraint definition and remain in the dialog box to define more constraints for the selected table.

   If the primary constraint definition is incorrectly defined, a message is displayed and the constraint definition is not created.

   The system fails to generate a constraint definition when an index is not defined as unique by the presence of either an OpenEdge unique attribute (indicated by an **I**) or by a unique constraint definition (indicated by a **C**) in the **Unique** column.

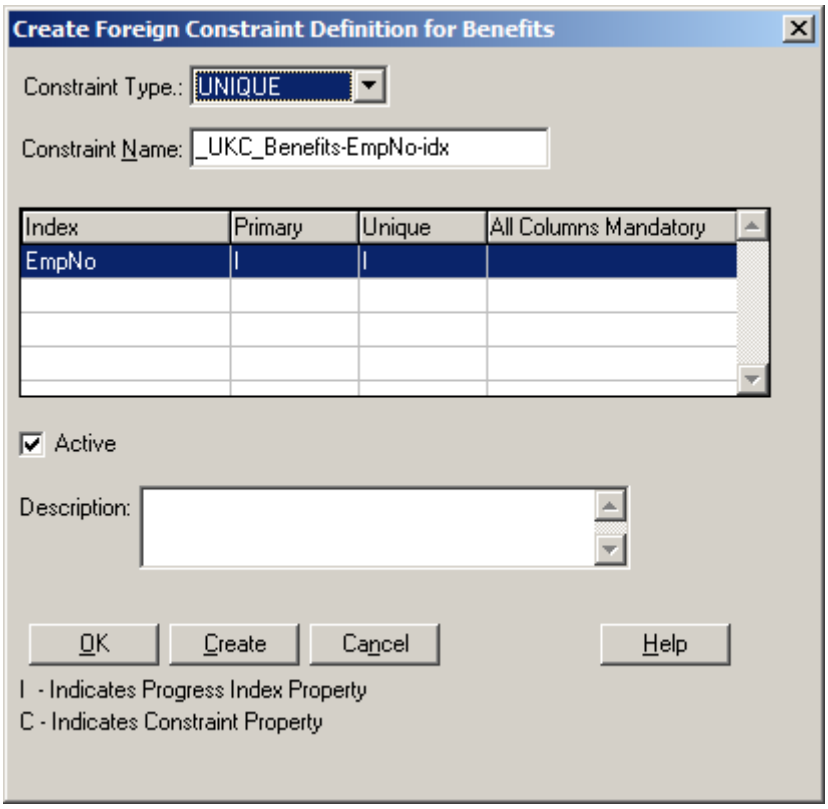
Defining a unique key constraint definition

The following procedure describes how to define a unique key constraint definition. Alternately, you can define unique key constraint definitions during migration.

To define a unique key constraint definition:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select**DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**. The **View/Maintain Foreign Constraints** dialog box appears.
3. Select the table for which you are defining a unique key constraint definition and click **Create Constraints**. The **Create Foreign Constraint Definition for** *tablename* dialog box appears, as shown in figure below.
4. Select **UNIQUE** from **Constraint Type**. The appearance of the **Create Foreign Constraint Definition for** *tablename* dialog box is modified as shown in the following figure .

   Figure 16: Create Foreign Constraint Definition for tablename dialog box for unique constraint definition



5. Accept the generated default constraint definition name, or enter a new one.
6. Select an index for the unique key constraint definition.
7. By default, the unique key constraint definition has the **Active** check box checked. Leave this box checked if you want the constraint definition you are defining to be available for migration.
8. Optionally, add a description of the constraint definition in the **Description** box.
9. Click **OK** to save the constraint definition and exit the dialog box. Click **Create** to save the constraint definition and remain in the dialog box to define more constraint definitions for the selected table.

Defining a foreign key constraint definition

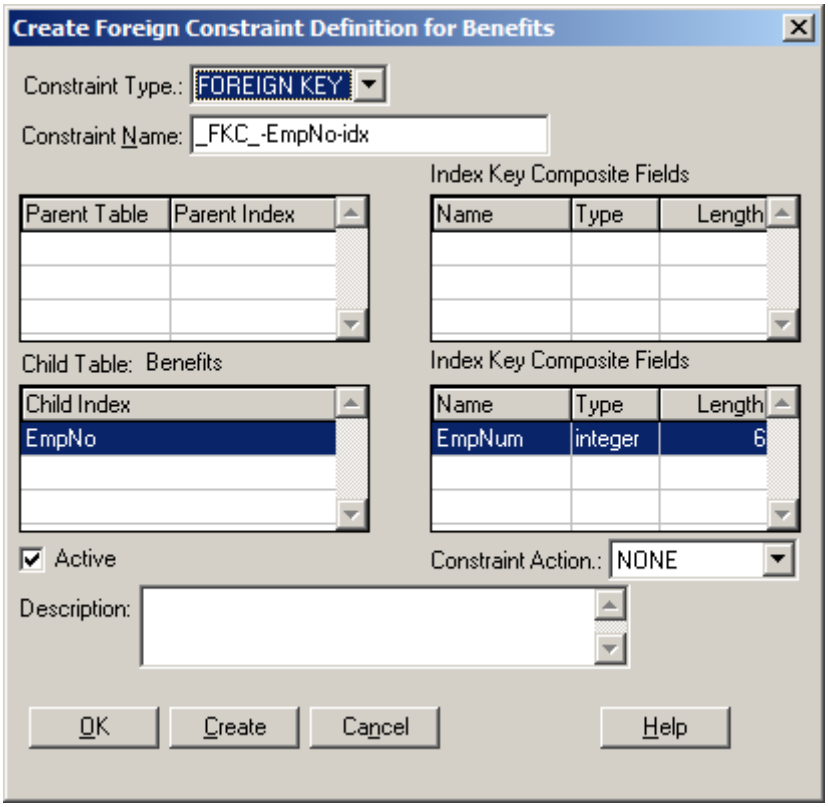
The following steps demonstrate how to define a foreign key constraint.

To define a foreign key constraint definition:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**.

   The **View/Maintain Foreign Constraint Definitions** dialog box appears.
3. Select the table for which you are defining a foreign key constraint definition and click **Create Constraints**. The **Create Foreign Constraint Definition for** *tablename* dialog box appears, as shown in figure below.
4. Select **FOREIGN KEY** from **Constraint Type**. The **Create Foreign Constraint Definition for** *tablename* for a foreign key dialog box appears as shown in the following figure .

   Figure 17: Create Foreign Constraint Definition for tablename dialog box for foreign key constraint definitions



5. Accept the default constraint definition name, or enter a new one.
6. The **Parent Table/Parent Index** lists the indexes from all of the other tables defined in the OpenEdge database, available for selection. For the DataServer for Oracle, the list shows primary indexes. Below the parent index information is the **Child Index** list. The **Child Index** list shows the available indexes for the selected table. For both parent and child indexes, the **Index Key Composite Fields** shows the columns that comprise the indexes. Select an index from the parent and child lists with matching composite fields.
7. By default, the foreign key constraint definition has the **Active** check box checked. Leave this box checked if you want the constraint definition you are defining to be available for migration.
8. Select an appropriate action item from the **Constraint Action**.

- Constraint actions can be **NONE**, **CASCADE**, or **SET NULL**. A typical constraint action ensures that the data integrity, in case you need to delete data in the rows of a referenced table.
- The **NONE** constraint action ensures data is not deleted from a row which exists in a foreign key table that references the value in the referenced table.
- The **CASCADE** constraint action ensures that whenever you delete rows in the master (referenced) table, the respective rows of the child (referencing) table with a matching foreign key column is deleted.
- The **SET NULL** constraint action ensures that the foreign key values in the referencing row are set to `NULL` when the referenced row is deleted.

9. Similarly, a row cannot be deleted as long as there is a reference to it from a foreign key table.
10. Optionally, add a description of the constraint definition in the **Description** box.
11. Click **OK** to save the constraint definition and exit the dialog box. Click **Create** to save the constraint definition and remain in the dialog box to define more constraint definitions for the selected table.

Defining a check constraint definition

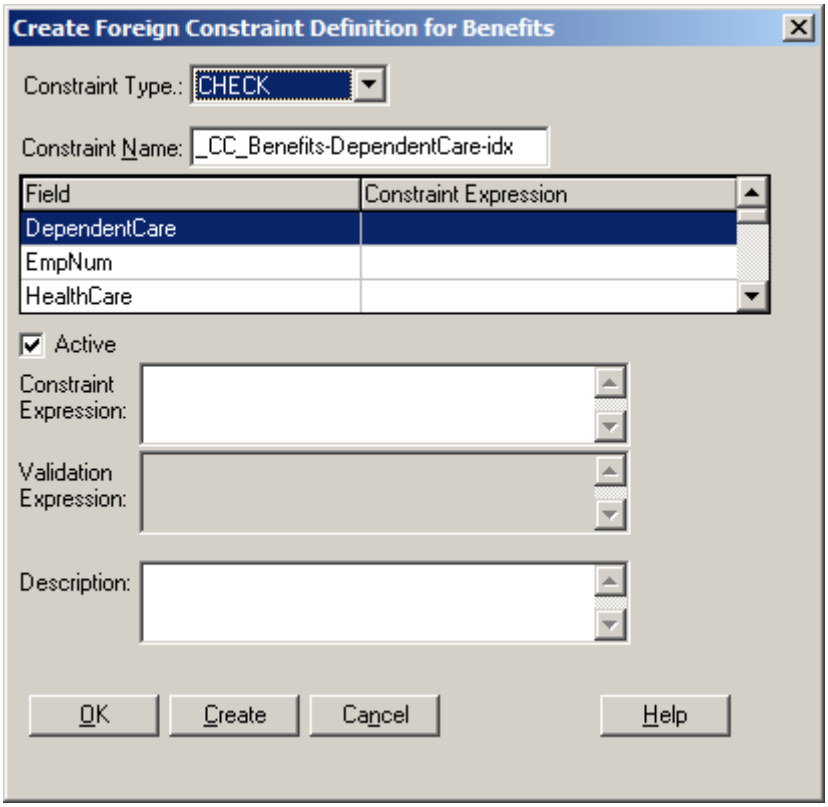
A check constraint definition is a server-side validation. The following procedure describes how to define a check constraint definition.

To define a check constraint definition:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**. The **View/Maintain Foreign Constraint Definition** dialog box appears.
3. Select the table for which you are defining a check constraint definition and click **Create Constraints**. The **Create Foreign Constraint Definition for** *tablename* dialog box appears, as shown in figure below.

4. Select **CHECK** from **Constraint Type.** The appearance of the **Create Foreign Constraint Definition for** *tablename* dialog box appears as shown in the following figure .

   Figure 18: Create Foreign Constraint Definition for Customer dialog box for check constraint definitions



5. Accept the default check constraint definition name, or enter a new one.
6. Select a field from the **Field** list.
7. By default, the check constraint definition has the **Active** check box checked. Leave this box checked if you want the constraint definition you are defining to be available for migration.
8. Enter an expression for server-side validation in the **Constraint Expression** field.

   Caution: The expressions entered for are not checked for correct syntax. If the expression is not valid on the server, the check constraint definition is rejected when building server check constraint definition objects on the server.

9. A validation expression is a client-side validation check that conforms to OpenEdge syntax requirements and only takes place during interactive screen I/O. This is in contrast to a check constraint definition which is tied to the actual database column, not the screen dialog.

   Caution: The validation expression is a read-only value for reference. Use the **Field Properties** dialog box of the **Data Dictionary** to maintain validation expressions.

10. Optionally, add a description of the constraint definition in the **Expression Description** box.
11. Click **OK** to save the constraint definition and exit the dialog box. Click **Create** to save the constraint definition and remain in the dialog box to define more constraint definitions for the selected table.

Modifying an existing constraint definition

Once you have defined a constraint definition, you can change a subset of a constraint definition's properties. The following procedure describes how to modify a constraint definition.

To modify a constraint definition:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**. The **View/Maintain Foreign ConstraintDefinition** dialog box appears.
3. Select the table and constraint definition you want to modify, and click **Constraint Properties**. The **Modify Foreign Constraint Definition for** *tablename* dialog box appears.
4. The **Modify Foreign Constraint Definition for** *tablename* dialog box is very similar to the **Create Foreign Constraint Definition for** *tablename* dialog box, with the exception being that only modifiable fields are active. Modify your constraint definition as required.

   Note: If the constraint definition property you want to change is not modifiable, you can delete the existing constraint definition and define a new one.

   ---

5. Click **OK** to save the changes to the constraint definition and exit the dialog box. Click **Create** to save the changes to the constraint definition and remain in the dialog box to modify other constraint definitions.

Activating or deactivating a class of existing constraint definitions

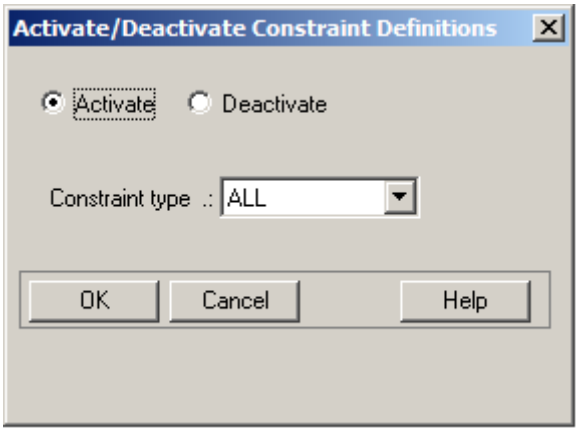
Once you have defined constraint definitions, you can activate or deactivate a whole class of constraint definitions at once. When a constraint definition is deactivated, it is unidentified and disabled during a subsequent migration.

The following steps demonstrate how to activate or deactivate a class of constraint definitions.

To activate a class of constraint definitions:

1. Start **Data Administration** and connect to the OpenEdge database you are migrating.
2. Select**Activate/Deactivate Constraints** > **Server Attributes** > **ORACLE Utilities** > **DataServer**. The **Activate/Deactivate Constraint Definitions** dialog box appears, as shown in the following figure .

   Figure 19: Activate/Deactivate Constraint Definitions dialog box

3. Select the **Activate** or **Deactivate** toggle.

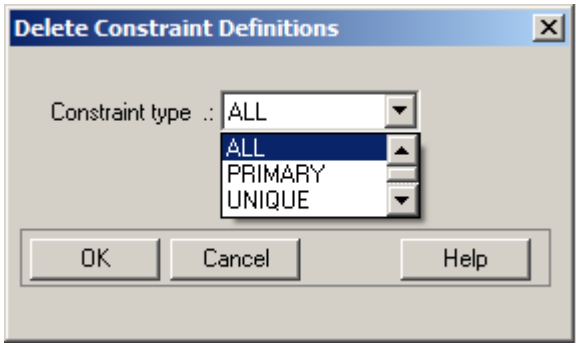
4. Select a constraint type or ALL.
5. Click **OK** to activate or deactivate constraint definitions and exit the dialog box, or click **Cancel** to exit without changing the state of the constraint definitions.

Deleting an existing constraint definition

Once you have defined a constraint definition, you can delete it. The following procedures demonstrate how to delete all constraint definitions or one class of constraint definitions.

1. To delete one constraint definition:
   a) Start **Data Administration** and connect to the OpenEdge database you are migrating.
   b) Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **View/Maintain Foreign Constraints**. The **View/Maintain Foreign Constraint Definitions** dialog box appears.
   c) Select the table and constraint definition you want to delete.
   d) Click **Delete Constraints** to delete the selected constraint definition.
2. To delete all constraint definitions or all of a particular type of constraint definition:
   a) Start **Data Administration** and connect to the OpenEdge database you are migrating.
   b) Select **DataServer** > **ORACLE Utilities** > **Server Attributes** > **Delete Constraints**. The **Delete Constraint Definitions** dialog box appears, as shown in the following figure .

      Figure 20: Delete Constraint Definitions dialog box



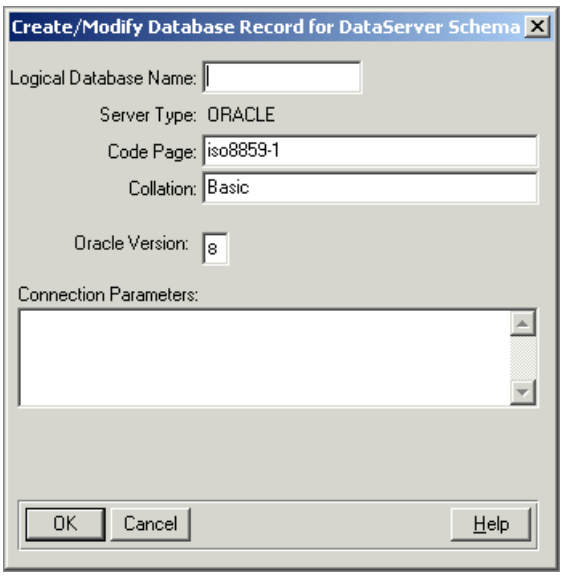
   c) Select the **Constraint type**, either **ALL** or a specific sub-set.
   d) Click **OK** to delete the constraint definitions. You are prompted to confirm your decision. Click **Yes** to delete the constraint definitions or **Cancel** to exit without deleting the constraint definitions.

Creating a schema image

Once you have started Oracle, started the DataServer processes, and created and connected to a local empty OpenEdge database, you can create the schema image. For information on starting Oracle see Running Oracle on page 151. For information on starting the DataServer processes, see Starting the DataServer processes on page 152.

To create a schema image:

1. From the Windows **Data Administration** main menu or the character-based **Data Dictionary** main window, select **DataServer** > **ORACLE Utilities** > **Create DataServer Schema**. The following dialog box appears:



2. Type a logical database name in the **Logical Database Name** field. You use the logical database name to connect to your Oracle database. You also use it to refer to the database in your programming applications. For more information on database names, see *Develop ABL Applications*.

   If you are building a schema holder for a distributed Oracle database, the logical name that you choose must be unique across the distributed database.

   Note: If you place the schema image from a second non-OpenEdge database into a schema holder, you must give the second schema image a different logical database name from the first schema. The schema holder has one physical name, but each schema image it contains must have a different logical name.

   ---

3. In the **Code Page** field, type the name of the code page for the schema image. The name must be the OpenEdge name for the code page.

   Note:

   If you choose UTF-8 as the schema image code page, your schema holder's code page must also be UTF-8. If your Oracle environment uses a code page that OpenEdge does not support, you must supply a conversion table that translates between the OpenEdge client code page and the code page that Oracle uses.

   See Code pages on page 37 in for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see *Internationalize ABL Applications*.

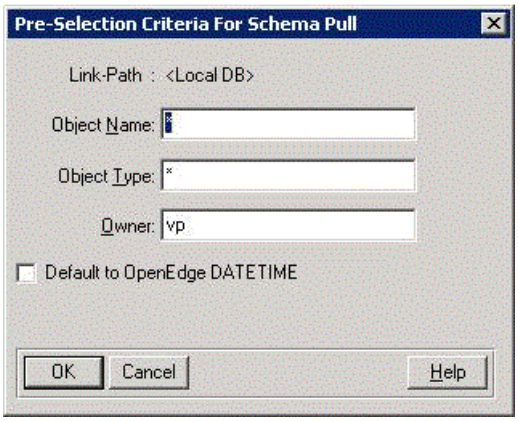
   ---

4. In the **Collation** field, type the name of the collation table.
5. In the **Oracle Version** field, type the version number of Oracle you are using. Possible values are `8` (the default), `9, or 10g`.
6. Type the required connection parameters in the **Connection Parameters** field.

At a minimum, you need the User ID and Password parameters. Remote connections also require networking parameters. See Connecting a schema holder at startup on page 172 for a description of the required and optional connection parameters. This is how you identify the Oracle user in an Oracle Networking connection:

```
-U userid@service-name -P password
```

7. Choose **OK**. The utility prompts you to verify the Oracle user ID and password.
8. If you did not specify the User ID (`-U`) and Password (`-P`) parameters in the previous dialog box, type an Oracle user ID and password combination that has the privileges required for creating a schema holder.
9. Choose **OK**. The following dialog box appears:



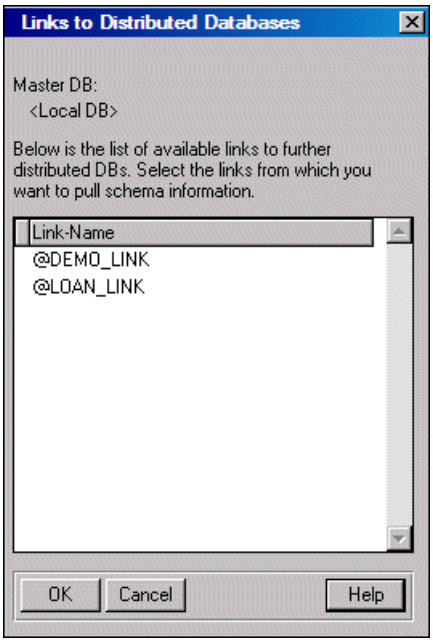
10. Preselect the objects that the schema holder should contain. You can select these objects by object name, object type, or owner/library. By default, the wild card symbol (*) appears and specifies that the utility selects all objects. Typically, you should not include system-owned objects.

Note: Progress Software Corporation recommends that you do not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)

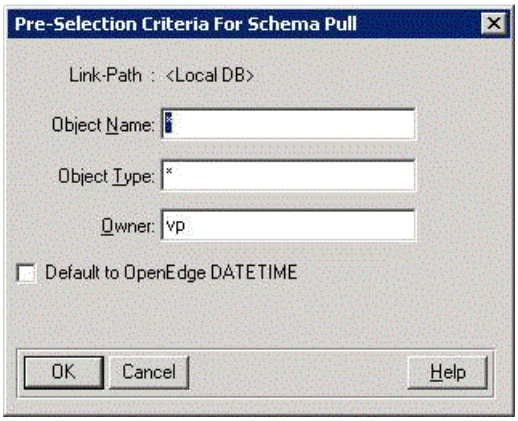
---

11. Choose **OK**.

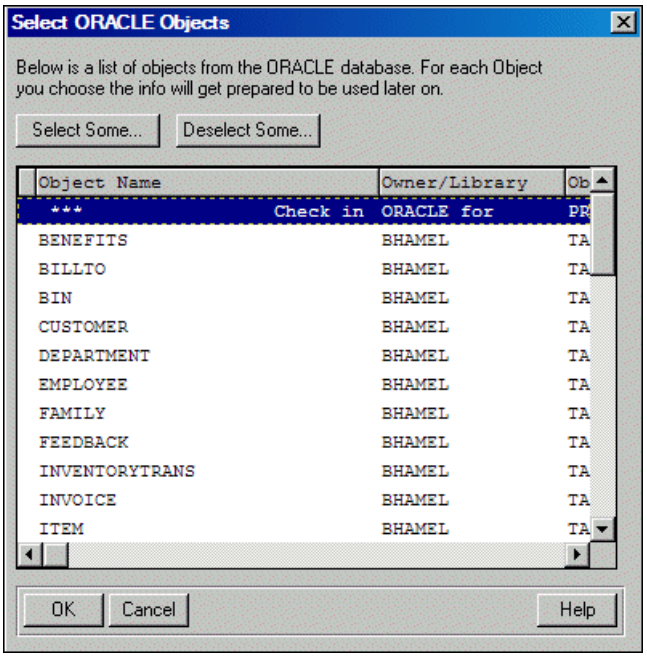
    If your Oracle database is part of a distributed database, the **Links to Distributed Databases** dialog box appears, listing the linked databases that make up the distributed database. Otherwise, go to Step 14.



12. Select the linked databases whose objects you want to include in the schema holder, then choose OK. The **Pre-Selection Criteria For Schema Pull** dialog box appears. You can select objects by name, type, and owner for a linked database. For example:



13. Choose **OK**. The following dialog box appears:



14. Select the objects that you want to include in the schema holder.

Note: If object names repeat across a distributed database, the DataServer qualifies each name by adding *-n*. For example, if your database has two `INVOICE` tables, the schema image will list them as `INVOICE` and `INVOICE-1`.

---

15. Choose **OK**. The DataServer reads information about the database objects and loads the data definitions into the schema holder. The time this process takes depends on the size and number of the Oracle objects.

    For each table, the DataServer selects an index or the native `ROWID` to support the `ROWID`. If an Oracle object (such as some views) does not support the `ROWID` function, the DataServer issues the warning, "Please check warnings and messages in the file ds_upd.e." The `ds_upd.e` file lists the objects that do not support `ROWID`.

    You can change the DataServer's selection by using the Data Dictionary. For example, you can choose to use an alternate integer index or the native `ROWID`. See Defining the ROWID on page 231 for instructions and ROWID function on page 72 for more information on `ROWID`.

    If the DataServer encounters Oracle errors while creating or maintaining schema information or loading data, it displays the Oracle message and message number on screen. See your Oracle documentation for complete descriptions of the error messages.

    The DataServer makes an additional connection to the schema holder when it creates, updates, or verifies a schema image. If you are connecting to multiple databases or schema images, make sure that you set the Number of Databases (`-h`) parameter to a value that accommodates the number of databases and schema images plus one.

Updating a schema image

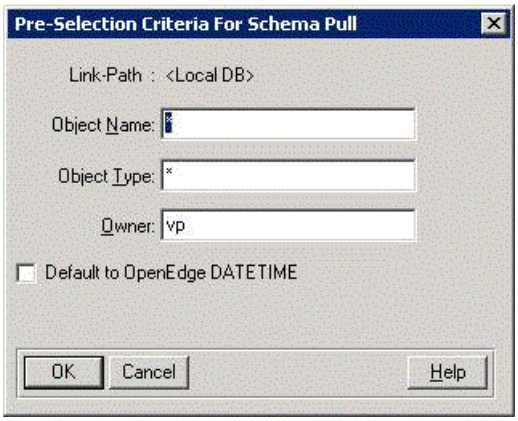
The Update/Add Table Definitions utility allows you to:

In this exercise, you update a schema image to reflect any changes you might have made to the data definitions for your Oracle database.

- Add additional object definitions from the Oracle database to a schema image. Use this option if you add a new table or view to the Oracle data definition and want that change reflected in the schema image.
- Update existing object definitions in a schema image to reflect a change in the supporting Oracle database object definitions.

To update a schema image:

1. Choose **DataServer** > **ORACLE Utilities** > **Update/Add Table Definitions**. The utility will ask you to verify that you are connected with a User ID and Password combination that has sufficient privileges. If the current User ID is correct, click **OK**, otherwise enter a different User ID and Password combination.
2. Once you are properly connected to Oracle, the following dialog box appears:



3. Preselect the Oracle objects that the utility uses to update the schema image. By default, the wild card symbol (*) appears in the fill-in fields. The wild cards specify that the utility will use all objects in the Oracle database, including system catalog information. You can change the criteria by typing new information in the fill-in fields.

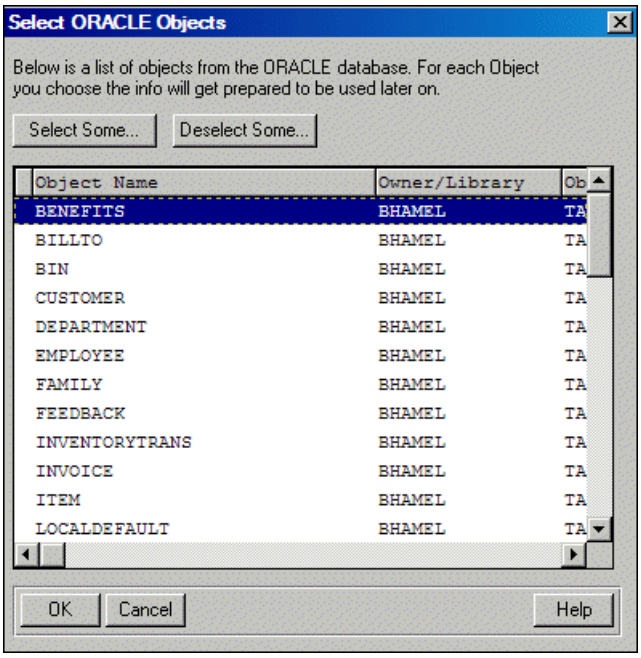
Note:

Progress Software Corporation recommends that you do not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)

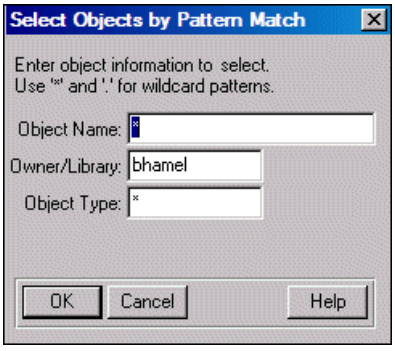
If there are links to Distributed Databases in your Oracle instance, a list of them will be displayed. Either highlight the proper link, or simply click **OK**.

---

4. Choose **OK**. The **Select ORACLE Objects** box lists the objects and table information that you preselected:



5. Select the objects you want to update, then choose **OK**. Typically, you should not include system-owned objects.
6. To select tables by matching a pattern, choose the **Select Some** button. The following dialog box appears:



7. Type the pattern that you want to match, then choose **OK**. The application returns to the main window.

   When you update a definition, the old definition is overwritten with the new one based on the current Oracle object. It also preserves the OpenEdge-specific table information. So, if you want to add a new column to a table in your Oracle database and then update the definition, you do not have to re-enter all of the OpenEdge-specific information for the previously existing columns (fields) in the definition.

Verifying a schema image

You might want to verify that the schema image matches the data definitions in the Oracle database. For example, if you delete the Customer table from the Oracle database, but not from the schema image, the Verify utility reports that there is an orphaned object. You can verify the schema information from a single table or from multiple tables, and then choose to update the table or tables so that the schema information matches the Oracle definitions.

The verify utility reads the definitions in the Oracle database and compares them to the information in the schema image. It reports the differences it finds and the degree of their severity. These are the categories of differences and how they impact your database applications:

- **Minor** — These differences have no impact on the usability of your application.
- **Retained** — The Update utility cannot correct these differences, hence the term "retained." You must determine how severely they impact your application and change the data definitions either in the schema holder using the Data Dictionary or in the Oracle database.
- **Severe** — These differences might cause your application to malfunction. When the Verify utility detects severe differences, it automatically updates the schema image to resolve the discrepancies. It adjusts the information in the schema image to match the Oracle definitions. Severe differences in definitions that the DataServer uses internally also cause the schema image to be updated.

The following table lists the differences that the utility detects.

Table 59: Verify utility report

Database object	Difference	Category
Table	Description	Retained
Table	Foreign type	Severe
Table	Name in OpenEdge	Retained
Table	Package name	Severe
Table	`ROWID` index	Retained
Index	Active	Minor
Index	Description	Retained
Index	Name in OpenEdge	Retained
Index	Unique	Retained[19]
Index field	Abbreviated	Minor
Index field	Ascending	Severe
Index field	Order	Severe
Field	Case-sensitivity	Retained
Field	Decimals	Retained[20]
Field	Description	Retained

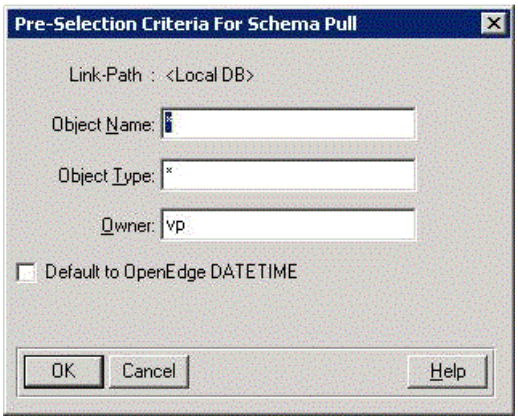
[19] When you update an index, the index is flagged as unique if it was defined as unique in either the Oracle database or the schema image.

[20] If the corresponding information in the Oracle database is incompatible with the information in the schema holder, the affected fields are not updated. For example, if the Oracle data type is NUMBER and the OpenEdge data type is CHARACTER, the data type information is not updated.

Database object	Difference	Category
Field	Extent	Severe
Field	Initial value	Retained[20]
Field	Mandatory	Retained
Field	Name in OpenEdge	Retained
Field	Order	Retained
Field	OpenEdge data type	Retained
Field	OpenEdge format	Retained[20]
Field	Shadow-column Name	Severe

To verify a table:

1. Choose **DataServer** > **ORACLE Utilities** > **Verify Table Definition**. The utility will ask you to verify that you are connected with a User ID and Password combination that has sufficient privileges. If the current User ID is correct, click **OK**. Otherwise enter a different User ID and Password combination.
2. Once you are properly connected to Oracle, the following dialog box appears:

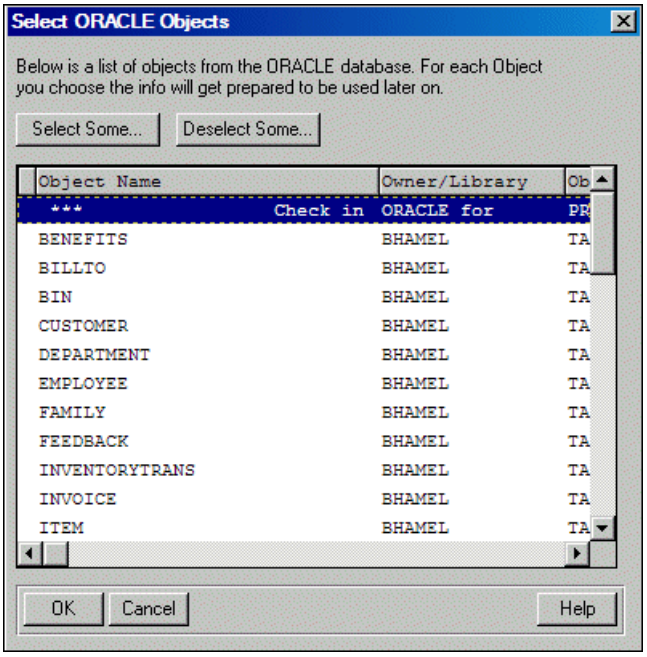


3. Preselect the Oracle objects that the utility uses to update the schema image. By default, the wild card symbol (*) appears in the fill-in fields. The wild cards specify that the utility will use all objects in the Oracle database, including system catalog information. You can change the criteria by typing new information in the fill-in fields.
4. By default, the utility verifies objects in the schema holder that match objects in the Oracle database. To check whether there are objects in Oracle that are not represented in the schema holder, deselect the **Verify only objects that currently exist in the schema holder** toggle box.

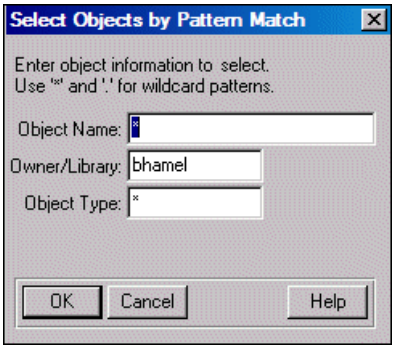
Note: Deselecting the **Verify only objects . . .** toggle box and leaving the **Object_Name** and **Object_Type** as wild cards can cause this function to take a long time to complete if your Oracle database is large.

---

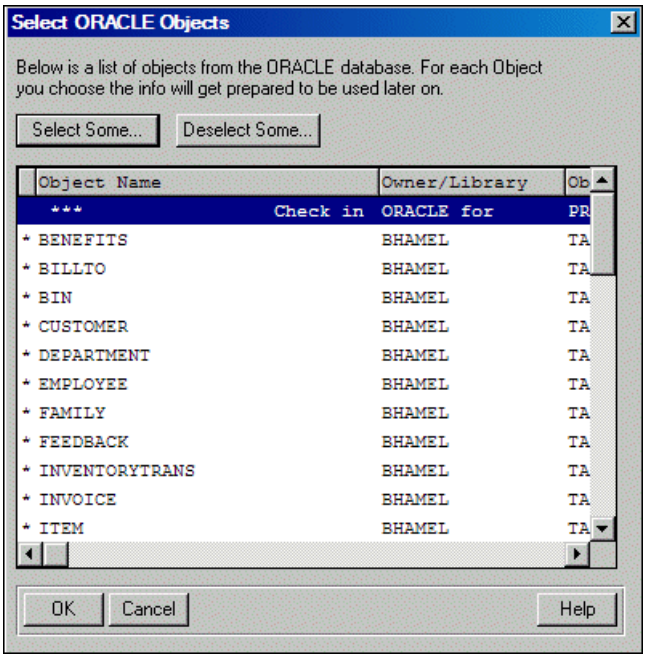
5. There is a 32K limit on the contents of the verify report. Select **Output differences to file** if you anticipate a large volume of differences, or wish to save the report. The file that is produced is named `<ldbname>+.vfy` and is written to the working directory.
6. Choose **OK**. The **Select ORACLE Objects** dialog box lists the objects and table information that you preselected:



7. Select the objects you want to update, then choose **OK**.
8. To select tables by matching a pattern, choose the **Select Some** button. The following dialog box appears:



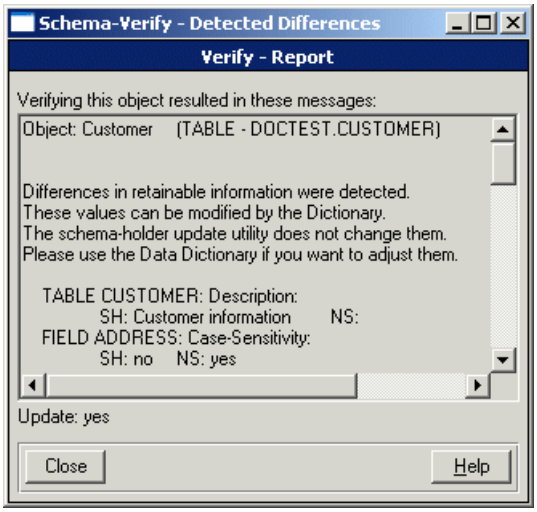
9. Type the pattern that you want to match, then choose **OK** to start the verification. If you did not choose to output the differences to a file, the following dialog box appears, listing the objects and the results of the verification:



10. In Windows, the report window will display automatically. (If there are no differences, to report, a message confirms this situation.) When reading the text of the report, **SH** indicates the value in the schema image; **NS** indicates the value in the Oracle database.

Note: On UNIX, choose the **View Reports** button to view a description of the differences found.

For example:



11. Chose **Close** to return to the **Schema Verify** dialog box.
12. The utility automatically selects objects with severe differences for updating. You can select or deselect all other objects as you wish.

    You must resolve retained differences manually. Retained differences appear in reports until you resolve them.

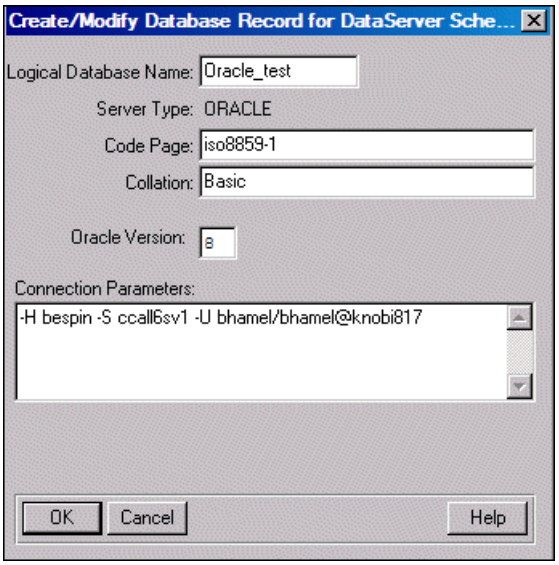
13. Choose **OK** to start the update or **Cancel** to quit the utility without updating the schema image.

Changing logical name and connection information

The **Edit Connection Information** dialog box allows you to modify connection information for your schema image.

To change connection information for a schema image and the Oracle database:

1. Choose **DataServer** > **ORACLE Utilities** > **Edit Connection Information**. The following dialog box appears:



2. Make changes to the **Connection Parameters** fields. When you are done, choose **OK** to return to the main window.

   The changes do not take effect until you disconnect and reconnect the schema holder. When you reconnect, the new connection parameters are used.

   For details on connection parameters, see Connecting the DataServer on page 165 and *Startup Command and Parameter Reference*.

   This utility also allows you to change the logical name of the Oracle database. You cannot be connected when changing the logical name. After the name change, this tool closes. You can restart the tool to continue working with the utilities.

Note: If you rename a schema image, you must recompile the ABL procedures that you compiled under the old schema-image name.

Modifying a schema image

You can begin using the DataServer as soon as you load the Oracle data definitions into the schema holder. However, you might want to use certain OpenEdge features, such as labels, validation expressions, or validation messages. Or, you might want to change the default data type provided for fields in the schema-holder tables.

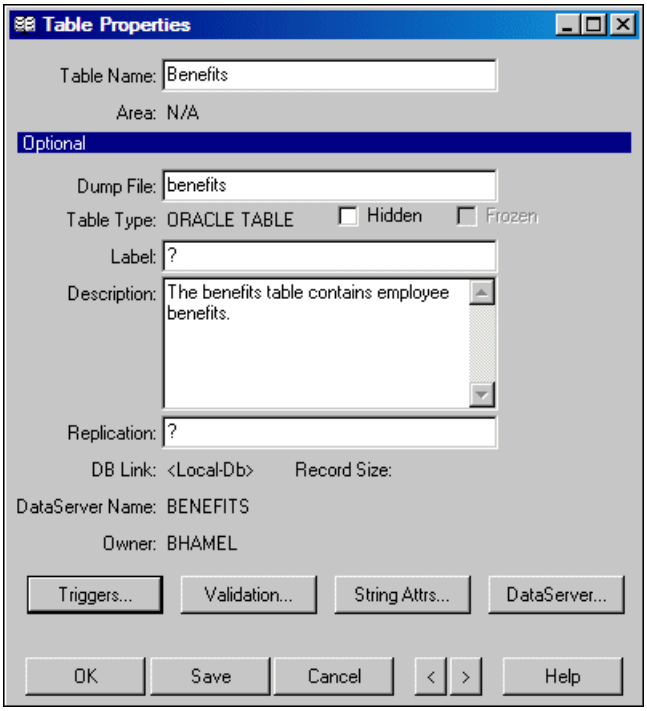
You can define OpenEdge information at both the table and field levels in the schema image. The following sections describe how to enter information at both levels.

Modifying table-level information

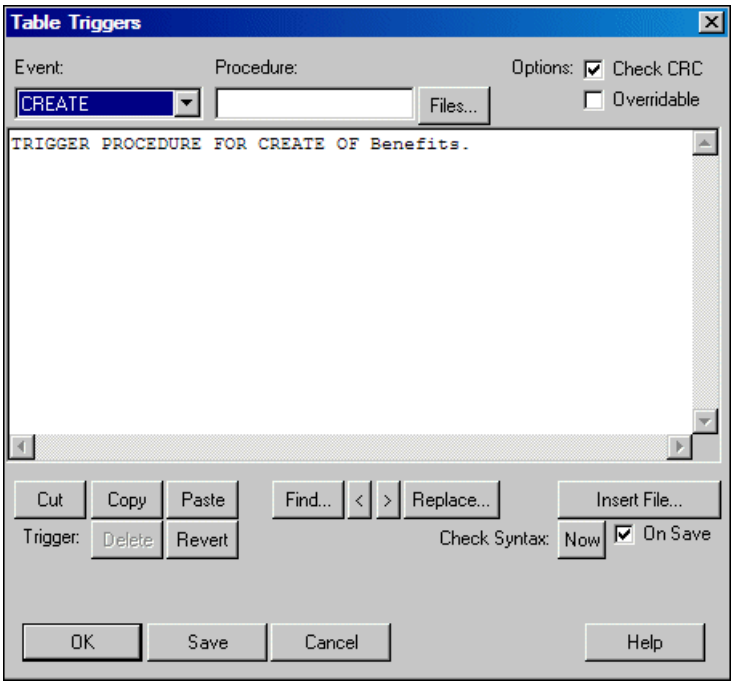
At the table level, labels, descriptions, table triggers, and validation can be modified for each table in the schema image.

To modify information in the schema image at the table level:

1. Access the Data Dictionary.
2. Select a table from the **Tables** list.
3. Choose the **Table Properties** button. The following dialog box appears:



4. Choose the **Triggers** button. The **Table Triggers** dialog box appears:



You can create or include a trigger procedure.

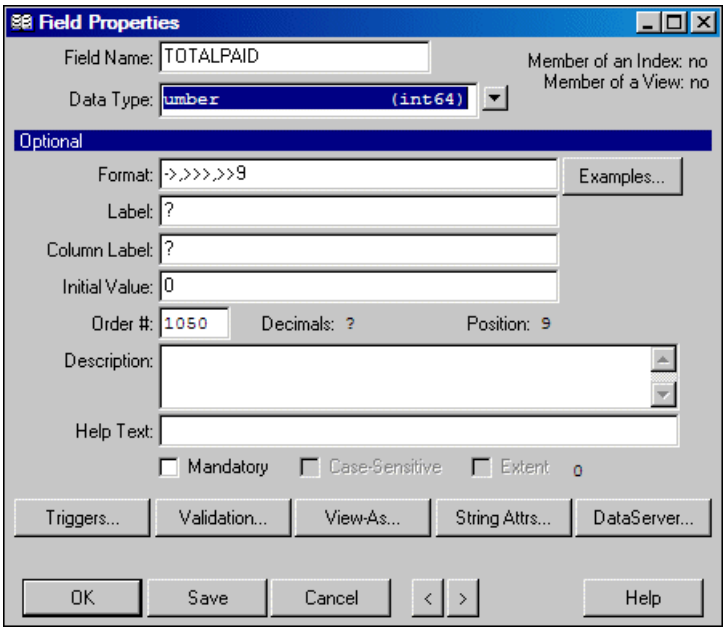
5. Choose **OK** to return to the **Table Properties** dialog box.
6. Choose **OK** to return to the Data Dictionary.

Modifying field-level information

At the field level, labels, descriptions, field triggers, and validation can be modified for each field in a table in the schema image.

To modify information in the schema image at the field level:

1. From the **Data Dictionary** main window, choose the `Fields` mode button.
2. Select a table from the **Tables** list.
3. Select a field from the **Fields** list.
4. Choose the **Field Properties** button. The following dialog box appears:

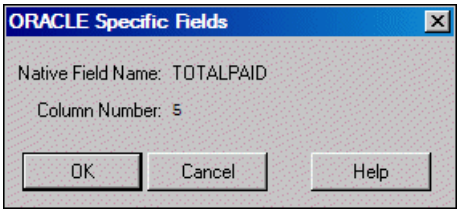


You can enter information at the field level, such as a validation expression or a validation message. You can change the data type or the format in which OpenEdge displays the data.

For example, choose the Oracle DataServer `NUMBER` data type, which in turn defaults to the OpenEdge `decimal` data type mapping. However, you can change the `decimal` mapping to either the OpenEdge `INTEGER` or `INT64` data type instead. (The previous **Field Properties** dialog box shows the `NUMBER` data type with the `INT64` data type option selected.) For more information about optional settings for data types, see Data types on page 50.

For `CHARACTER` fields that are not indexed, you can change the case sensitivity. You cannot change a field definition to be an extent through the Data Dictionary.

5. Choose the **DataServer** button. The following dialog box provides information about the corresponding column in the Oracle database:



6. Choose **OK** to return to the **Field Properties** dialog box.
7. When you are done making changes, choose **OK** to return to the Data Dictionary.

Note: You can override field-level validation expressions in your application by including the appropriate ABL statement.

Defining the ROWID

When you create or update a schema image, the DataServer uses the following guidelines when determining how to support the `ROWID` function:

- If the Oracle table has a PROGRESS_RECID column, the DataServer selects it. This column provides the optimal support for the ROWID function. You cannot select an alternative to it.
- If the Oracle table does not have a PROGRESS_RECID column, the DataServer selects a unique index on a mandatory NUMBER column with precision < 10 or undefined and a scale of 0 or undefined.
- If none of the above is available, the DataServer uses the native ROWID.

The Data Dictionary allows you to select the native ROWID or an alternative index to support the ROWID function. The alternative index should be a stable one.

Note: If you are connected to an Oracle database and you change how the ROWID is supported for a table, you must reconnect to the database to avoid inconsistent row identifiers.

An index that you select to support ROWID must meet the following database- and application-level criteria:

The database criteria are:

- The index consists of a single field.
- The indexed column is a NUMBER data type.

The application criteria are:

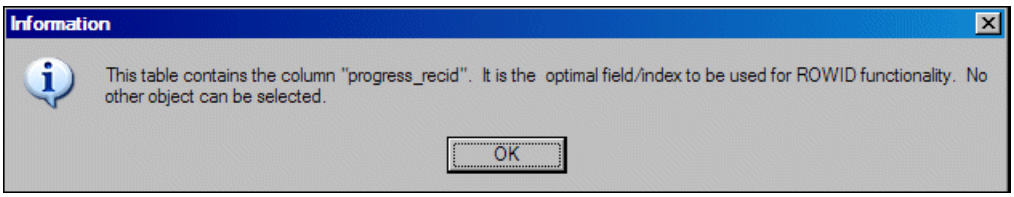
- The index must at least be treated as unique by your application.
- The index must at least be treated as mandatory in your application.
- The indexed column only allows values with fewer than ten digits (for applications associated with an OpenEdge 10.1A Release or earlier).
- The indexed column only allows values with fewer than nineteen digits (for applications associated with an OpenEdge 10.1B Release or earlier).

The Oracle DataServer allows you to select indexes that meet only the first two database-related requirements, but your application must ensure that the index meets the remaining criteria.

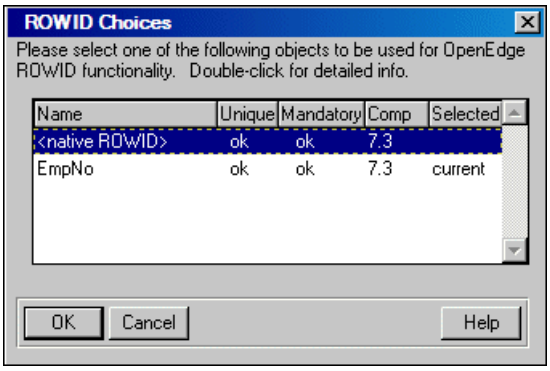
For example, your application might access an indexed column defined as a NUMBER column, but the scale might not be specified. If your application assigns only values between 1 and 214783647 to this column, it meets one of the additional criteria. Your application must ensure that the indexed column meets the other two criteria as well. If you do not meet all three criteria, you risk corrupting your database.

To select an index to support the ROWID function, in the Data Dictionary:

1. Select the table from the **Tables** list.
2. Choose the **Table Properties** button.
3. Choose the **DataServer** button. If the Oracle table contains a PROGRESS_RECID column, the following message appears:



4. If the PROGRESS_RECID message did not appear, the **ROWID Choices** dialog box appears:



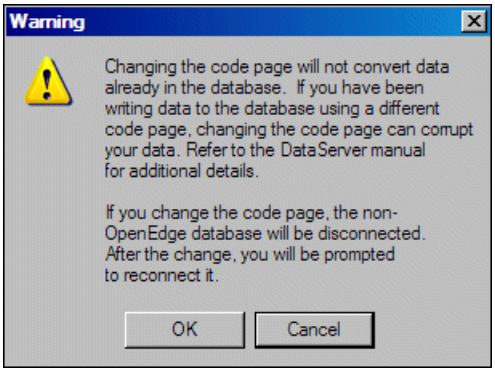
5. Choose **<native ROWID>** or an index to support the `ROWID` function. Choose **OK** to return to the **Table Properties** dialog box.
6. Choose **OK** to return to the Data Dictionary.
7. If you referenced the table during the current session, you must disconnect from Oracle, then reconnect, for the `ROWID` selection to take effect.

Changing a code page

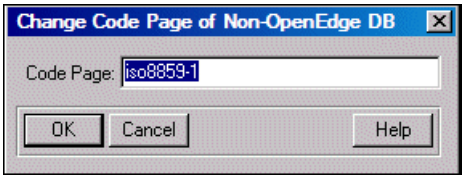
You can change the code page in a schema image at any time. The Create DataServer Schema utility allows you to create a schema image even if you do not have the correct code-page information. You can add or correct the code-page information at a later date. However, if you have been writing 8-bit character data to your Oracle database with the DataServer, that data is unaffected by the change in the code page. Your database might be corrupted if you start writing data with the DataServer and a schema image that uses a new code page.

To change the DataServer code page:

1. Choose **DataServer** > **ORACLE Utilities** > **Change DataServer Schema Code Page**. The following message appears:



2. Choose **OK** to continue. The following dialog box appears:



3. Type the OpenEdge name for the code page that the Oracle Database uses.

   Note:

   See *Internationalize ABL Applications* for a complete list of code pages that OpenEdge supports. If you are using an unsupported code page, OpenEdge allows you to create your own conversion tables.

4. Choose **OK** to change the code page and return to the **Data Administration** main window.

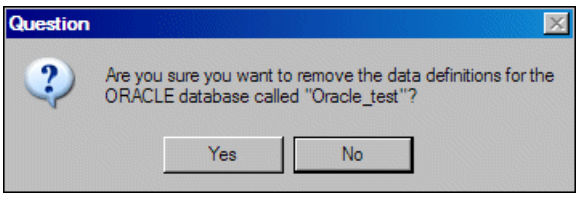
   If you were connected to the schema holder and Oracle database when you chose to change the code page, the utility disconnected you to make the change. The **Connect Database** dialog box appears to allow you to reconnect.

Deleting a schema image

If you no longer need a particular schema image, you can delete it.

To delete a schema image from a schema holder:

1. Choose **DataServer** > **ORACLE Utilities** > **Delete DataServer Schema**. The following dialog box appears:



2. Choose **Yes** to verify your selection. After the schema image is deleted, the Data Admin tool returns to the main window.

Migrating an OpenEdge Database to Oracle with PRO/SQL

The PRO/SQL utility creates an SQL script (`.sql` file) that contains all the data definition statements needed to re-create an OpenEdge database as an Oracle database. The script contains statements to define only those features that are supported by Oracle. For example, if your OpenEdge database has word indexing, the SQL script does not include an equivalent index definition because Oracle does not support this feature.

The script does, however, attempt to mimic some basic OpenEdge functionality that would not otherwise be available in an Oracle database. The script creates a `PROGRESS_RECID` column to support Progress record identifiers, initial values, and a column for each extent of an array.

PRO/SQL is not integrated into the DataServer architecture. Unlike the OpenEdge-to-Oracle migration utility, it does not create a schema holder for you. If you want to access the Oracle database that you create using the PRO/SQL `.sql` file, you must create a schema holder for it.

See *OpenEdge Development: Basic Development Tools* or online Help for instructions for using PRO/SQL.

OpenEdge DB-to-Oracle utility

The OpenEdge DataServer for Oracle supports the OpenEdge DB-to-Oracle utility that allows you to migrate an OpenEdge database to an Oracle database. While the DataServer typically makes an Oracle database conform to an OpenEdge database, this utility provides compatibility in the opposite direction. It copies an existing OpenEdge database schema into a target Oracle database. The utility performs the following tasks:

- Creates objects in the target Oracle database
- Creates the schema holder and schema image
- Optionally populates the Oracle database by dumping and loading the data from the OpenEdge database

The Oracle database you create with this utility is a basis for an application database. Before deploying your new Oracle database, you might want to make manual adjustments to take advantage of additional Oracle features that are not supported by the migration utility, such as clustering and multiple tablespaces.

The OpenEdge-to-Oracle utility has the following requirements:

- A local OpenEdge database
- Oracle permissions, as identified in the following table

Table 60: Oracle permissions

Permission	Object
`CREATE SESSION`	Database
`CREATE TABLE`	Database
`CREATE SEQUENCE`	Database

`SELECT`	System objects:
	`sys.argument$`
	`sys.cclol$`
	`sys.cdef$`
	`sys.col$`
	`sys.com$`
	`sys.con$`
	`sys.dual`
	`sys.icol$`
	`sys.ind$`
	`sys.link$`
	`sys.obj$`
	`sys.procedure$`
	`sys.seq$`
	`sys.syn$`
	`sys.tab$`
	`sys.ts$`
	`sys.user$`
	`sys.view$`

`CREATE SEQUENCE`	Database
Enough quota	Default table space

Preparing a database for the utility

The OpenEdge DB-to-Oracle utility does not literally translate definitions for Progress fields into Oracle columns. It makes some adjustments in order to provide the functionality of the OpenEdge and Oracle systems. The following table describes the changes to a database that result from making an OpenEdge database compatible with Oracle when using the OpenEdge-to-Oracle utility.

Table 61: An OpenEdge DB-to-Oracle database conversion

OpenEdge database	Oracle database
Record ID	An additional `NUMBER` column named `PROGRESS_RECID`
Case-insensitive index	The `UPPER(column-name)` syntax is used by the index SQL
Array	A column for each array element named *`field`*`##1`, *`field`*`##2`, etc.
Sequence	A sequence with the same characteristics (incrementation, beginning value, etc.)
OpenEdge BLOB type	This is set to NOT NULL in Oracle

Note: OpenEdge BLOB must be set to NOT NULL in Oracle, that is, any Oracle application that inserts records into a table also initializes the record with EMPTY_BLOB() function as OpenEdge DataServer cannot access a NULL BLOB column.

The utility makes these changes in the target Oracle database automatically. The following sections explain how to account for these changes when you plan your migration.

Oracle size limitations

The limitations that Oracle and OpenEdge place on the number of columns or fields per table are important issues to consider when you plan a migration:

- **OpenEdge** — 32,000 fields per table
- **Oracle** — Refer to your Oracle documentation for limits

When you use the OpenEdge DB-to-Oracle utility to create an Oracle database, it adds columns to an Oracle table to accommodate OpenEdge functionality (see table An OpenEdge DB-to-Oracle database conversion). Because Oracle limits the maximum number of columns, you must plan for the additional columns generated by the utility when you create the database schema.

When you design the OpenEdge database tables that you want to make compatible with Oracle, leave enough columns free for the migration utility to use. The following table lists how many columns each OpenEdge object requires on an Oracle table.

Table 62: Columns required by OpenEdge objects

Database object	Number of columns
Record ID	One per table
Array	One per array element

Column names

Oracle allows column names to be only 30 characters long. The OpenEdge DB-to-Oracle utility truncates the names of the OpenEdge fields so that they meet this limitation. After running the utility, you can use the Data Dictionary to access the schema holder and modify the field name.

Column width

The OpenEdge-to-Oracle migration utility uses either a field's format or width information (_width value in the _Field schema record) for the column length when it defines the field as an Oracle column. Since OpenEdge allows a field to hold more data than the field's format will display, a field's format might not be the best indicator as to how large a column should be sized on the server. You can select **Width** in the **For Field widths use** migration option to create a column that is wider than the format indicates. If a column generated by the migration utility is not wide enough to hold the data, the utility backs out any data that is loaded and the data in Oracle Server database remains unchanged.

The default size of a column's width is twice the length set for the OpenEdge FORMAT field attribute. When you use the DBTOOL utility option, **SQL Width Scan w/Fix Option**, the utility calculates the column width based on the actual size of data in the columns to ensure that server data length requirements are satisfied during migration. Unless the default column width is greater than the actual data size obtained from the largest column row, the utility adjusts the column width based on its calculation. Otherwise, the default column width is utilized during migration to help prevent under-sized columns on the server from receiving over-populated column data.

When you use the DBTOOL utility to calculate a column size of an extent column, derived from the entire array of extent elements, the extent element with the maximum number of characters is identified, and all the extent elements defined on the server are re-sized to the column length of the extent element with the maximum number of characters. If the extent element with the maximum number of characters exceeds the migration threshold parameter **Maximum Varchar Length**, then each element of the array is re-sized as a LOB server column type. The OpenEdge record buffer may otherwise overflow, whereas a LOB locator takes a very small space in the record.

For more information on the additional considerations for handling character data length, see Handling character length during database migration on page 247 and Adjusting field widths during migration on page 253 section.

Prior to running the utility, you need to determine which method of calculating column size will be used to ensure your data will fit. If necessary, use the **Adjust Field Width** tool in the Data Dictionary to enlarge character, decimal, and array fields or the database administration tool, DBTOOL. For more information on DBTOOL, see *Manage the OpenEdge Database*.

Note: For those fields with a display format of x(8), the utility automatically generates a `VARCHAR(30)` column.

If you have miscalculated the width of a column, you can change it.

Changing the width of the column

To change the width of the column:

1. Use an editor to open the file with the `.sql` extension, which contains information for the entire database.
2. Find the section of the file that describes the table.
3. Assign the column a larger value to accommodate the data.
4. Extract the section of the file that describes the table, including the delete table syntax, and place it in another file named *filename*`.sql`. Be sure to include any index information for the table. Re-create the table by running SQL*Plus. Enter the following command at the system prompt:

```
sqlplus userID/password < filename.sql
```

   This command deletes the data definition for the table and re-creates it.

5. Load the data.

Naming conventions

The OpenEdge-to-Oracle utility generates names for the columns it adds to an Oracle table. These names follow the conventions described in the following table.

Table 63: OpenEdge-to-Oracle naming conventions

OpenEdge field	Oracle column name
Record ID	`PROGRESS_RECID`
Array with three elements(for example, `MONTH`)	*field* `##1`, *field*`##2`, *field*`##3`(`MONTH##1`, `MONTH##2`, `MONTH##3`)

For example, if your source OpenEdge table includes a field extent named `MONTH` with 12 elements, the OpenEdge-to-Oracle utility creates 12 columns of the same data type named `MONTH##1`, `MONTH##2`, `MONTH##3`, etc. ABL reference to `MONTH[3]` translates into a reference to the Oracle column `MONTH##3`.

In addition, the OpenEdge-to-Oracle utility modifies the names of OpenEdge objects to non-Oracle keywords in the Oracle schema. For example, `ORDER` is a reserved word in Oracle, so the utility changes the `order` field of the OpenEdge demo database to the `order_` column in an Oracle database. It also modifies names that might conflict with unrolled field extents and that contain characters unacceptable to Oracle. The OpenEdge field name `order-line` changes to `order_line` to account for the fact that the hyphen (-) is an unacceptable character in Oracle object names. Be aware that if you have objects in your OpenEdge database with names that are very similar, and conflict with Oracle reserved words and restricted characters, they can be modified multiple times to produce a valid name.

Oracle tablespaces for tables and indexes

The OpenEdge-to-Oracle utility allows you to take advantage of Oracle tablespaces for tables and indexes. Oracle tablespaces are logical storage places for data. The physical data is stored in data files. By default, an Oracle database has a single tablespace named `SYSTEM`, which contains the database schema information. Typically, you choose to store data in one or more additional tablespaces, depending on the size of the database.

When running the OpenEdge-to-Oracle utility, you can specify tablespaces that you have previously defined in the empty Oracle database that is the target of the migration. The tablespaces must be online, that is, available for user access. The utility uses the tablespace defined as the default for the user profile you provide when you run the utility (User and Password information). If the tablespace you specify is not online or defined in your Oracle database, an error message is displayed and the initial screen is redisplayed for you to specify a different online table space or to remove the specified table space and use the default user table space. If no tablespace is defined in Oracle, the utility uses the `SYSTEM` tablespace. See your Oracle documentation for more information on tablespaces.

Running the OpenEdge DB-to-Oracle utility

You can run the OpenEdge DB-to-Oracle utility interactively or in batch mode.

To migrate an OpenEdge database to Oracle:

1. Create a target Oracle database if you do not already have one. If you want to use tablespaces, you must define them in your Oracle database before running the OpenEdge-to-Oracle utility. When developing a new DataServer application, start with a new empty database.
2. Start an instance of your target Oracle database.
3. If connecting with Oracle networking, make sure that the `ORACLE_SID` environment variable is set to the Oracle database name.
4. Make sure that the `ORACLE_HOME` environment variable is set to the directory where you installed Oracle.
5. Start the OpenEdge client and connect to the OpenEdge database that you want to migrate to Oracle.

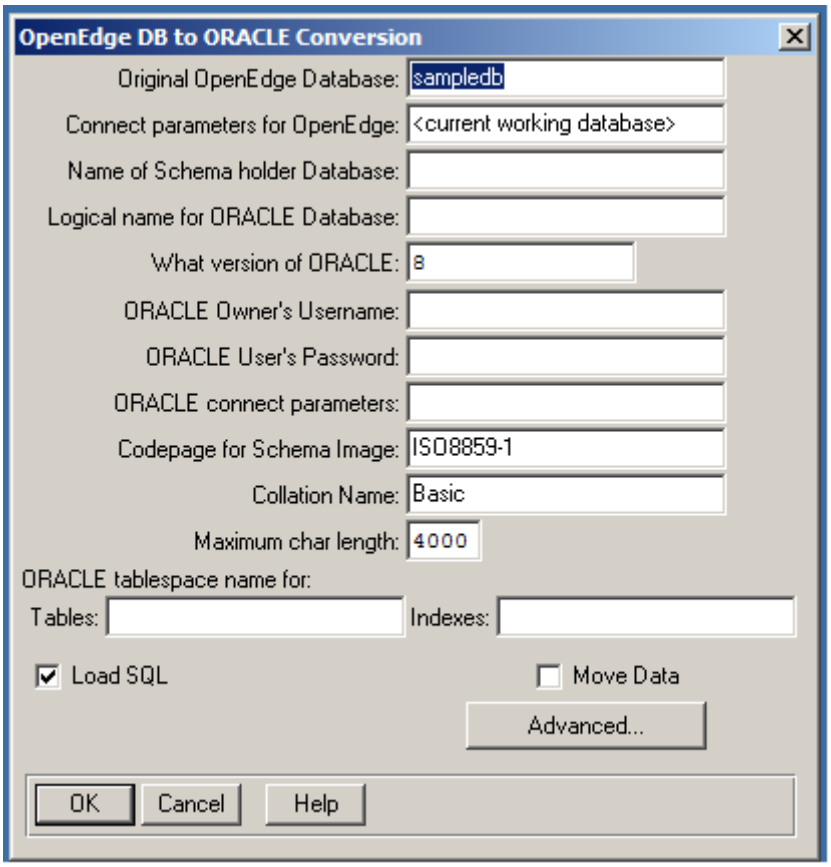
   Note: For a DBE DataServer application, you must specify the Internal Code Page (`-cpinternal`) and Stream Code Page (`-cpstream`) parameters when you start the client. The values that you specify for these parameters much match the code page that the Oracle database uses.

6. In Windows, from the Data Admin tool, choose **DataServer** > **ORACLE Utilities** > **Schema Migration Tool** > **OpenEdge DB to Oracle**.

   On UNIX, access the utility from the DataServer menu in the Data Dictionary. Or, you can run the utility from the command line:

   ```
   pro source_db -p prodict/ora/protoora.p
   ```

7. The following screen appears and prompts you for information:



The following table describes each of the elements on the **OpenEdge to Oracle Conversion** dialog box.

Table 64: OpenEdge to Oracle Conversion

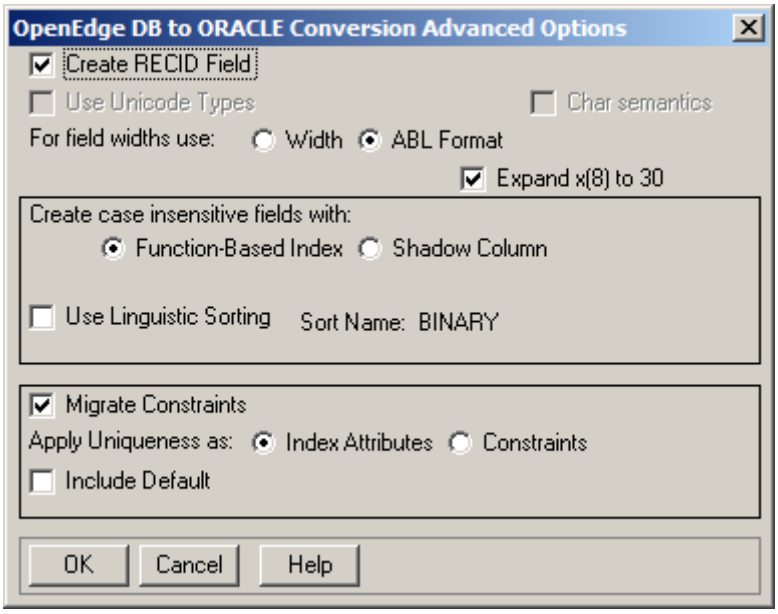
Interface Element	Description
Original OpenEdge Database	The source database name. Accept the default value.
Connect parameters for OpenEdge	The parameters for the connection to the source OpenEdge database, which is the current working database. Accept the default value.
Name of Schema holder Database	Enter the name for the schema holder. The utility creates the schema holder if it does not exist.
Logical name for Oracle Database	Enter the Oracle database logical name. The logical database name is the name of the schema image and the name you will use to refer to the Oracle database in applications. The database's logical name must be different than the name you enter for the schema holder and different than the name of any other schema image existing in that schema holder.

Interface Element	Description
What version of Oracle	Enter the version of Oracle you are using. The default is 8. To enable Unicode, this value must be set to `9` or higher, which represents Version 9i or later.
Oracle Owner's Username	Enter the Oracle database owner's name.
Oracle User's password	Enter the owner's password.
Oracle connect parameters	Enter additional connection parameters as needed.
Code page for Schema Image	Enter the OpenEdge name for the code page that the Oracle Call Interface (OCI) uses. For Unicode support, set the value to UTF-8 when migrating from an OpenEdge database to an Oracle database. **Note:** For a UTF-8 schema image code page, the corresponding OCI code page (that is, the NLS_LANG setting) setting must be AL32UTF8 to ensure full Unicode compatibility.
Collation Name	Enter the collation table you wish to use. By default, the collation table is Basic. The collation table you specify must be defined in the `convmap .dat` file or you will receive an error message.
Maximum char length	Enter a positive value up to and including 4000. This value is defaulted based on the values of other Unicode-specific settings in your migration. See Handling character length during database migration on page 247 for details.
Tablespace tables	Enter the name of the Oracle tablespace where you want to store schema information. The default is the Oracle user's default tablespace.
Tablespace index	Enter the name of the Oracle tablespace where you want to store index information. The default is the Oracle user's default tablespace.
Load SQL	Check this toggle box to load the `.sql` file that contains the data definitions for your OpenEdge database into the Oracle database. By default, this option is checked.

Interface Element	Description
Move Data	Check this toggle box to dump and load data from your OpenEdge database to the Oracle database. Copying data from a large database can take a long time. For example, you might not check this box if you want to dump and load data at a more convenient time.
Advanced	Click **Advanced** to open the **ORACLE Conversion Advanced Options** dialog box The purpose of this dialog box is to reduce clutter on the main dialog box and provide logical grouping of fields on it. See the table below for the details you must enter.

8. To display the **OpenEdge DB to ORACLE Conversion Advanced Options** dialog box, you click the **Advanced** button on the OpenEdge DB to ORACLE Conversion dialog box.

Figure 21: OpenEdge DB to ORACLE Conversion Advanced Options



The following table describes the information for which you will be prompted to enter.

Table 65: OpenEdge DB to Oracle Conversion Advanced Options

Interface Element	Description
Create RECID field	Check this toggle box to create an Oracle database that supports arrays, case-insensitive indexes, backward and forward scrolling, and the OpenEdge record identifier. These objects result in additional columns added to Oracle tables.

Interface Element	Description
Char semantics	Check this toggle box to set the unit of measure for length to character semantics when migrating OpenEdge file fields to Oracle table columns. Length values for the migration are derived from the For field widths use setting. The Char semantics option is only enabled for Unicode migrations. To enable it, the schema image code page must be set to `utf-8` and you must specify an Oracle Version of `9` or later. The default setting is YES if ORACODEPAGE is UTF-8.
Use Unicode Types	Maps character fields to Unicode data types. You must specify `9` or higher (corresponding to an Oracle version of 9i or later) to enable this option.
For field widths use	When pushing fields to a foreign data source, you can select one of two primary field format options: • **Width** — Uses the value of the **_width** field in the _field record. Recommended especially for Unicode implementations. For more information on field widths, see Adjusting field widths during migration on page 253 . • **ABL Format** — Compiles with the current default width specified. (default) If you select the **ABL Format** option, you have an additional setting to define: • **Expand x(8) to 30** — This setting is on by default to indicate that the format for the character fields defined as x(8) will be created as 30 characters. **Note:** You cannot use the **Expand x(8) to 30** setting with the **Width** option.

Interface Element	Description
Create case-insensitive fields with	Provides OpenEdge compatibility to case-insensitive data columns. To support case insensitivity, you must choose between the options, **Function-Based Index** (the default) and **Shadow Column**. When you select the **Function-Based Index option**, you can choose either the `UPPER()` or the `NLS_UPPER()` function; applied to indexes that include case-insensitive columns. The choice of which function to apply to indexes is made internally during migration, which depend upon the selection for **Use Linguistic Sorting** that follows this selection in the Advanced Options sub-section. Select **Shadow Column** to support case-sensitive columns. This option is a legacy option for backward compatibility. By default, this option is not selected, allowing the utility to automatically support ABL case insensitivity via Oracle function-based indexes. **Note:** Function-based indexes and shadow columns can co-exist in a schema holder to support case insensitive indexes.
Using Linguistic Sorting	Check this option to migrate the database with linguistic sorting. By default, the **Linguistic Sorting** option is disabled, and a binary sort is applied to code points. To use linguistic sorting, you must specify a Linguistic Sort Name that corresponds to a valid sort name in Oracle. This sort name is applied to all migrated indexes, which become `NLS_SORT()` function-based indexes. **Note:** If you select Function-Based Index for case-insensitivity of the same indexed columns, then the migrated indexes will be both NLS_UPPER and NLS_SORT function-based indexes Information regarding the linguistic sorting is stored in the schema image. This information is used at the time of session connection to Oracle to set the session settings to NLS_SORT and NLS_COMP. Thus, the database session seamlessly utilizes the proper function-based indexes for sorting and matching record.

Interface Element	Description
Migrate Constraints	Indicates how the constraint definitions are managed. By default, the **Migrate Constraints** box is checked. The **Apply Uniqueness as:** toggle allows you to choose between **Indexes** or **constraint definitions** as the method of enforcing uniqueness. **Indexes** is selected by default to maintain backward compatibility. Select **Constraints** to change the method of enforcing uniqueness.
Include Default	Check this toggle box to include initial values in column definitions.

The utility creates a schema holder, an Oracle database that contains the objects from your OpenEdge database, and a startup procedure that you can use to connect to your schema holder. The startup procedure derives its name from the logical name for your Oracle database. For example, if you specified "orasports" as the logical database name, the utility creates the `corasports.p` startup procedure.

If your OpenEdge database is Unicode enabled, you need to consider the implications of data widening. For more information, see the and Handling character length during database migration on page 247 .

Running the OpenEdge DB-to-Oracle utility in batch mode

To run the OpenEdge DB-to-Oracle utility in batch mode on a UNIX client machine:

1. Create a target Oracle database. You should connect as a user with no pre-existing object.
2. Start an instance of your target Oracle database.
3. If using Oracle Networking, make sure that the `ORACLE_SID` environment variable is set to the Oracle database name.
4. Make sure that the `ORACLE_HOME` environment variable is set to the directory where you installed Oracle.
5. On your client machine, pass parameters to the utility by setting the environment variables as listed in the following table.

Table 66: Environment Variables

Environment Variable	Description
`PRODBNAME`	Specify the source OpenEdge database name.
`PROCONPARMS`	Specify parameters for the connection to the source OpenEdge database.
`SHDBNAME`	Specify the new schema-holder name.
`ORADBNAME`	Specify the Oracle database logical name. The database's logical name can be the same as its physical name, but it must be different from the name you enter for the schema holder.
`ORAVERSION`	Specify the version of Oracle you are using. Possible values are `8`, `9`, or `10`. The default is `8`.

Environment Variable	Description
ORAUSERNAME	Specify the Oracle user's name.
ORAPASSWORD	Specify the user's password.
ORACONPARMS	Specify additional connection parameters for the schema holder.
ORACODEPAGE	Specify the OpenEdge name for the code page that the Oracle Database uses.
TABLEAREA	Enter the name of the Oracle tablespace where you want to store schema information. The default is the SYSTEM tablespace.
INDEXAREA	Enter the name of the Oracle tablespace where you want to store index information. The default is the SYSTEM tablespace.
COMPATIBLE	Specify YES to create an Oracle database that supports FIND PREV/LAST and the OpenEdge ROWID function. The default is YES.
SQLWIDTH	Specify YES to use the _width field to calculate column size instead of using the format field.
ORACOLLNAME	Enter the collation name.
CRTDEFAULT	Specify YES to include initial values in column definitions.
LOADSQL	Specify YES to load the .sql file that contains the data definitions for your OpenEdge database into the Oracle database. Specify NO if you do not want the utility to load the .sql file, for example if you want to edit the file before loading it. The default is YES.
MOVEDATA	Specify YES to dump and load data. Otherwise, if you do not want to populate the Oracle database, specify NO. For example, you might specify NO if your database is large, and you want to dump and load data at a more convenient time. The default is NO.
EXPANDX8	This setting is on by default to indicate that the format for the character fields defined as x(8) will be created as 30 characters.
UNICODETYPES	Maps character fields to Unicode data types. The default is NO.

Environment Variable	Description
CHARSEMANTICS	Sets the unit of measure for length to character semantics when migrating OpenEdge file fields to Oracle data columns. The default setting is YES if ORACODEPAGE is UTF-8.
SHADOWCOL	Determines if shadow columns must be added to the record layout to represent data in upper case form for OpenEdge fields marked case-insensitive.
VARLENGTH	Dependent on other selections. Maximum value is 4000.

6. Enter these commands to set and export environment variables at the system prompt before running the compiled ABL program, protoora.r:

```
PRODBNAME=db-name; export PRODBNAME
PROCONPARMS="-1 -i"
SHDBNAME=schema-holder-name; export SHDBNAME
...pro -b -p prodict/ora/protoora.r
```

Handling character length during database migration

Converting to a Unicode code page or Unicode column character types might cause an expansion of data beyond a data source column's maximum length. When migrating an OpenEdge database to an Oracle data source, you must specify the maximum character length of your character columns. Consider the following when using the OpenEdge DB to Oracle Conversion utility:

When using a non-Unicode code page or when using a Unicode code page with byte-length semantics, the specified length must be a positive value less than or equal to 4000. When using character-length semantics, the Maximum char length default value is automatically modified depending upon other settings, including Code page for Schema Image, Char semantics, and Use Unicode Types. The default value may or may not be precisely accurate for your Unicode configuration.If you select a schema holder code page of utf-8 and the Use Unicode Types toggle box is unselected, the conversion utility sets the default maximum char length to 1000. If the Use Unicode Type toggle box is checked, the utility sets the default Maximum char length to 2000. The default settings accommodate possible character expansion for Unicode. You can override the defaults with your own values based on your specific Unicode configuration and your knowledge of data content for your OpenEdge character fields. The recommended settings for Maximum char length which safely allow for character expansion are summarized in table below.

Handling character type conversions to CLOB

Fields longer than the 4000 byte maximum character length of a VARCHAR2 column in Oracle are migrated as LONG columns by the Oracle DataServer migration tool. You can also set the **Maximum char length** migration option lower than 4000 bytes and field(s) exceeding that limit will be migrated as a LONG column. However, Oracle allows only one LONG column per table. If more than one migrated column is defined with a size greater than 4000 bytes (or the **Maximum char length** option value) or, if data widening from Unicode expansion causes data from more than one column to overflow the 4000-byte limit (or the **Maximum char length** option value), the table definition will become invalid and the table cannot be migrated to Oracle without modification.

Note: N/VARCHAR2 data types cannot exceed 4000 bytes irrespective of what number of characters are defined for the data NVARCHAR2 data type. If you are using an AL16UTF16 NCS data type, set the maximum character size to 2000 bytes. This is because thought the column length is 3000 characters, the single byte characters originating from an ANSI character set will expand to 2 bytes of storage and will occupy more than the maximum 4000 bytes of an N/VARCHAR2 field.

If multiple fields over the **Maximum char length** causes some tables to be excluded from the migration, these OpenEdge database definitions and the application that uses them will need to be modified first before a migration is possible. These database definitions should be changed from CHARACTER to CLOB data types and their associated references in the application should be reworked as LOB-managed data types. Once the number of fields exceeding the **Maximum char length** value has been reduced to one and the application has been changed accordingly, a DataServer table migration can proceed.

The following table provides details on character length settings during migrations:

Table 67: Recommended character length settings

Character semantics				Byte semantics
Use Unicode Types toggle box	**National character set**	**Database character set**	**Max length**	**Max length**
Selected	UTF-8	N/A	1333	4000
Selected	UTF-16	N/A	2000[21]	4000
Unselected	N/A	Non-Unicode	4000	4000
Unselected	N/A	UTF-8	1333	4000
Unselected	N/A	AL32UTF8	1000	4000

[21] Consideration should be given for whether data content might include UTF-16 supplementary characters. These will produce four-byte characters which would favor a maximum character length less than 2000.

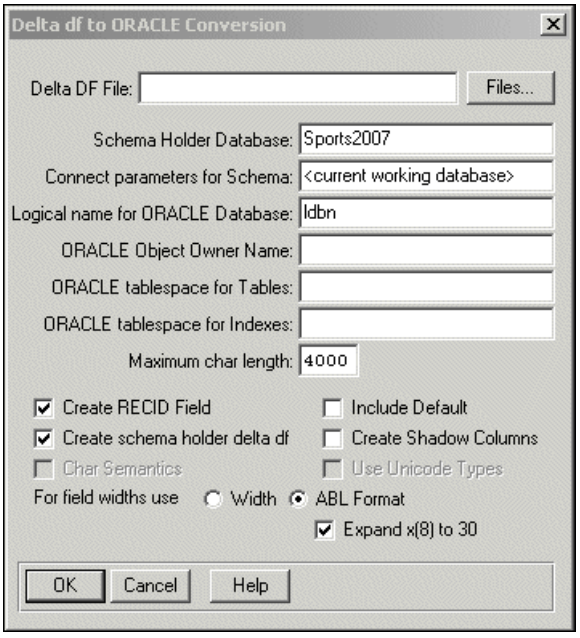
OpenEdge DB-to-Oracle Incremental Schema Migration utility

The Incremental Schema Migration utility allows you to migrate schema changes from an OpenEdge database to an Oracle database. For example, in the process of developing an application in ABL that you will migrate to Oracle, you might want to make and test schema changes in your OpenEdge database that you want reflected in the Oracle database. The utility reads a `delta.df` file that has been created using the standard incremental dump procedure, and creates a delta SQL file, named *<Oracle-logical-db>*`.sql`, that contains the SQL DDL for making the changes and a new delta `.df` file, named *<schema-holder-name>*`.df`. You can then load the `.df` file into the schema holder and apply the SQL file to the Oracle instance to complete the migration process.

Note that you do not make schema changes directly in the schema holder, which must remain synchronized with the Oracle database. The utility uses the schema holder to determine the Oracle definitions.

To run the Incremental Schema Migration utility:

1. From the **Data Admin** main menu, choose **DataServers** > **ORACLE Utilities** > **Schema Migration Tools** > **Generate Delta.sql OpenEdge to Oracle** . The following dialog box appears:



2. Provide the information as described in the following table.

Table 68: Delta df to Oracle Conversion UI elements

Interface element	Description
Delta DF File	The name of the `delta.df` file that was created when you ran the incremental dump routine against two OpenEdge databases. You can browse for the filename by choosing the **Files** button.
Schema Holder Database	The name of the schema holder.

Interface element	Description
Connect parameters for Schema	By default, the current working database is specified. To connect to a different database, specify connection parameters for the Oracle schema holder to be updated.
Logical name for Oracle database	Specify the Oracle database logical name, that is, the name by which you refer to the Oracle database in your application.
Oracle Object Owner Name	Enter the name of the owner.
Oracle tablespace for Tables	Enter the names of any tablespaces to be used here.
Oracle tablespace for Indexes	Enter the names of any tablespaces to be used for indexes here.
Maximum char length	Enter a positive value up to and including 4000. This value is defaulted based on the values of other Unicode-specific settings in your migration. See Handling character length during database migration on page 247 for details.
Create RECID Field	Check this toggle box if your Oracle database currently contains the `PROGRESS_RECID` field. Selecting this option will maintain the use of `PROGRESS_RECID` in any new tables added by this utility.
Include Default	Check this toggle box to include initial values in column definitions.
Create schema holder delta df	Check this toggle box if you want the utility to generate a `.df` file that includes the incremental schema information. You can then load this `.df` file into the schema holder. By default, this toggle box is checked.
Create Shadow Columns	Check this toggle box to support case-sensitive columns with shadow columns. By default, this option is not checked, allowing the utility to automatically support ABL case insensitivity via Oracle function-based indexes. **Note:** Function-based indexes and shadow columns can coexist in a schema holder to support case insensitive indexes.

Interface element	Description
Char semantics	Check this toggle box to set the unit of measure for length to character semantics when migrating OpenEdge file fields to Oracle table columns. Length values for the migration are derived from the For field widths use setting. The Char semantics option is only enabled for Unicode migrations. To enable it, the schema image code page must be set to `utf-8` and you must specify an Oracle Version of `9` or later.
Use Unicode Types	Maps character fields to Unicode data types. Only applies if schema image's code page is UTF-8. You must specify `9` or higher (corresponding to an Oracle version of 9i or later) to enable this option.
For field widths use	When pushing fields to a foreign data source, you can select one of two primary field format options: • **Width** — Uses the value of the **_width** field in the _field record. Recommended especially for Unicode implementations. For more information on field widths, see Adjusting field widths during migration on page 253. • **ABL Format** — Compiles with the current default width specified. (default) If you select the **ABL Format** option, you have an additional setting to define: • **Expand x(8) to 30** — This setting is on by default to indicate that the format for the character fields defined as x(8) will be created as 30 characters. **Note:** You cannot use the **Expand x(8) to 30** setting with the **Width** option.

3. Choose **OK**. The utility generates a `delta.sql` file and, optionally, a `delta.df` file.
4. After running the utility, you must apply the SQL it generates to the Oracle database and load the new `delta.df` file into the original schema holder so that it is synchronized with the modified Oracle database.

   The utility generates SQL that will create objects in the Oracle database. It creates the same objects as the OpenEdgeDB-to-Oracle Migration utility. For example, OpenEdge indexes are case-insensitive. To create this equivalent functionality in the Oracle database, for an index defined in the OpenEdge database on a `CHARACTER` field, the utility generates SQL to use the `UPPER` function for the index. The following table describes the Oracle equivalents of OpenEdge object types.

Table 69: Oracle equivalents of OpenEdge objects

OpenEdge object	Oracle equivalent objects
Case-insensitive Index	The index definition uses the `UPPER` function.

OpenEdge object	Oracle equivalent objects
Array	One column for each extent of the OpenEdge array. The columns are named *field-name*##*extent-number*. For example, an OpenEdge field called `monthly-amount` with an extent of 12 has 12 columns in Oracle with names such as `MONTHLY_AMOUNT##1` through `MONTHLY_AMOUNT##12`.
Table	If **Create RECID Field,** is selected, for any new table, a `PROGRESS_RECID` column is added. This indexed column provides a unique key on the Oracle table.A sequence named `table-name_SEQ` is also added. This sequence populates the `PROGRESS_RECID` column for each row in the Oracle table.
Deleted Field	The column is dropped from Oracle.

Not all OpenEdge objects can be converted to Oracle by this utility. The following table details restrictions on the update.

Table 70: Database modifications not converted to Oracle

Database object	Modification in OpenEdge	Action
Sequence	Starting value altered	None. Oracle does not allow the starting value of a sequence to be altered. You must manually drop and add the sequence to implement this change.
Trigger	Any	Applied to schema holder. ABL triggers are not converted to Oracle SQL.
Character field	Format altered	None. Oracle's restrictions on the alteration of character fields, such as knowing if all fields are `NULL` prior to decreasing the width, cannot be accommodated by this utility, therefore no actions are implemented.

The following table shows how the fields of an OpenEdge table convert to Oracle equivalents.

Table 71: Sample object equivalents

OpenEdge state table	Oracle STATE table
Character field: `State-Name`	`STATE_NAME`

OpenEdge state table	Oracle STATE table
Array with 3 Extents: `State-Fact`	`STATE_FACT##1 STATE_FACT##2 STATE_FACT##3`
Default record identifier object	`STATE##PROGRESS_RECID STATE_SEQ`

The utility ensures that the migrated objects have names that are unique to the Oracle database. If you have given the object a name that is not unique, it drops characters from the end of the name and appends numbers until it creates a unique name. Since Oracle requires that index names be unique to the database, the utility appends the table name to the indexed column name to create a unique name.

Adjusting field widths during migration

Especially when migrating a Unicode UTF-8 OpenEdge database with Unicode data, Progress Software recommends that you select **Width** in the **For fields widths use** options during OpenEdge database migration or Incremental Schema Migration. Prior to migration, the DBTOOL database utility can be used to size your server columns based on the actual size of your CHARACTER, RAW and DECIMAL data rather than using the default format value in the schema to size your server columns.

The width value for each field is based on the maximum number of characters (not bytes) found in the database records of that column. For character data type, the size is calculated based on the code page defined in the database schema. You can also use the Adjust Field Width utility from the Options pulldown menu of the Data Dictionary to manually set a specific character width on an individual column prior to migration. For more information on sizing character data using column width, see Column width on page 237.

While using the Oracle DataServer Migration utility or the Delta df to Oracle Conversion utility, if you select the **Char semantics** toggle box and use the recommended value for **Maximum char length**, you can ensure that the existing data in fields of your OpenEdge database are sized appropriately for Unicode character expansion in the Oracle database.

Updating the Oracle database

You should review, and modify if necessary, the `delta.sql` file that the utility generates before applying it. You can apply the `delta.sql` file to the Oracle database through SQL-based tools, such as Oracle's SQL*Plus.

After applying the `delta.sql` to the Oracle database, you must update the original schema holder by loading the new `delta.df` file into the original schema holder so that the original schema holder reflects the modifications you made to the Oracle database.

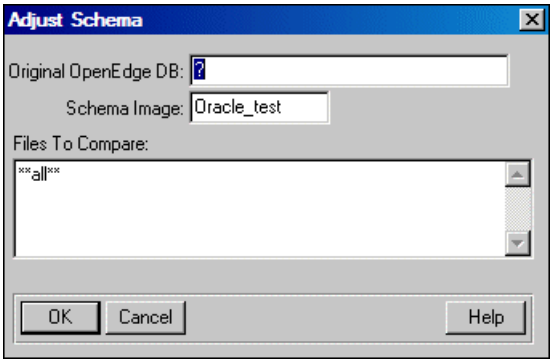
Note: If you select the **Char semantics** toggle box, you should run the Update/Add Table Definitions utility so that the correct byte length is stored in the schema holder.

Adjust an Oracle schema image to an OpenEdge database

The Adjust Schema utility allows you to migrate schema changes from an Oracle database and compare it to the same schema in an OpenEdge database. If you make changes in both your OpenEdge database and the same changes in Oracle using Oracle tools, pull those changes into the schema holder. The Adjust Schema utility will compare the OpenEdge database to the schema image and update the OpenEdge attributes necessary for your ABL code to run against both data sources.

To run the Adjust Schema utility:

1. Connect to the source OpenEdge database.
2. Connect to the Schema Image Schema Holder.
3. From the **DataServer** menu in either the Data Administration tool or the Character Data Dictionary, choose **DataServers** > **ORACLE Utilities** > **Schema Migration Tools** > **Adjust Schema**. The following dialog box appears:



4. Provide the information as described in the following table:

Interface element	Description
Original OpenEdge DB	Enter the name of the source OpenEdge database.
Schema Image	Enter the Oracle database logical name. The logical database name is the name of the schema image.
Files To Compare	Either leave the default **all** for all objects to be compared or enter the names of tables, sequences, or OpenEdge Views to be compared. If listing individual objects, the list must be a comma-separated list within a group of objects and semicolons must separate the groups. The groups are table, sequence, and view, and must be listed in that order.

A

Upgrading DataServer Applications

This appendix addresses two upgrade issues that users may encounter.

For details, see the following topics:

- Upgrading schema holders from Progress Version 9
- Manually adding PROGRESS_RECID to Oracle

Upgrading schema holders from Progress Version 9

If you have a Progress Version 9 schema holder and want to take advantage of OpenEdge Release 10 features, you must make specific preparations before using the Release 10 DataServer.

To upgrade a schema holder to use OpenEdge Release 10:

1. Start Progress Version 9 with the schema holder connected.
2. Dump the data definitions (`.df` file) from the schema holder. Dumping and loading a `.df` file is the only way to preserve any information you might have added to the schema, such as display formats, help strings, and validation expressions.
3. Start OpenEdge Release 10.
4. Create and connect to an empty OpenEdge database.
5. From the Data Admin main menu, choose **Admin** > **Load Data and Definitions** > **Data Definitions (.df)**.
6. Type the name of your `.df` file, then choose **OK**.

   The utility loads the `.df` file into the schema holder.

7. Use the **Verify Table Definitions** utility to verify the data definitions in the OpenEdge schema holder.
8. In Progress Version 8, the DataServer did not store information in the schema holder on maximum sizes of `CHARACTER` and `RAW` columns. If your schema holder was migrated to Version 9 without being dumped and loaded it could be missing this information. To make sure that the new schema holder has information on maximum size, use the **Update/Add Oracle Table Definitions** utility.
9. Recompile your r-code against the new schema holder.

Manually adding PROGRESS_RECID to Oracle

If you have an existing Oracle database, and you wish to take advantage of FIND PREV/LAST statements and cursor repositioning in an OpenEdge application for this database, you must manually add the `PROGRESS_RECID`.

To manually add PROGRESS_RECID to the Oracle database:

1. Using SQL*Plus, log in as the Oracle user who owns the table.
2. Create a sequence generator for the table named *table-name_SEQ*. Start with 1 and increment by 1, as shown:

```
CREATE SEQUENCE
table-name
_SEQ START WITH 1 INCREMENT BY 1;
```

3. Add a column to the table named *progress_recid*. This column holds a number that can be null. For example:

```
ALTER TABLE table-name ADD (progress_recid number null);
```

4. Update the table and set the *progress_recid* using *table-name_SEQ.nextval*, as shown:

```
UPDATE table-name SET progress_recid = table-name_SEQ.nextval;
```

5. Create a unique index name, *table-name##progress_recid*, that consists of just the *progress_recid* column, as shown:

```
CREATE UNIQUE INDEX table-name##progress_recid ON
  table-name (progress_recid);
```

6. Drop every non-unique index from the table and recreate it using the same components. Add *progress_recid* as the last component, as shown:

```
DROP INDEX table-name##index-name;
CREATE INDEX table-name##index-name ON table-name
  (column-name, progress_recid);
```

7. Verify that the sequence was created. For example:

```
SELECT table-name_SEQ FROM sys.dual;
```

8. Connect to Oracle and use the Data Dictionary's Oracle utilities to update the schema holder.

B

Environment Variables

This appendix lists the Oracle, OpenEdge, and system environment variables that affect building and running the OpenEdge DataServer for Oracle. See *Configure OpenEdge* for more information about environment variables.

The following table lists the environment variables and describes how to set them.

Table 72: Environment variables

Variable	How to set it
DLC(Required)	The path name of the directory where you installed OpenEdge.
DLCDB(Optional)	The path name of the directory where your databases reside.
DSLOGDIR(Optional)	The pathname of the log file that OpenEdge uses to keep track of DataServer processes and error messages.
NLS_LANG(Optional)	National language support variable. To ensure full Unicode compatibility, set to NLS_LANG=.AL32UTF8 on the system where the Oracle DataServer and Oracle OCI client libraries reside. When setting this variable, you must use the dot (.) notation.
ORACLE_HOME(Required)	The path name of the directory where you installed Oracle.

Variable	How to set it
ORACLE_SID(Optional)	The identifier for the Oracle instance you are accessing.
ORASOPATHNAME(Optional)	The complete file specification of the Oracle client shared library.
ORASRV(Optional)	The path name to the executable of the Oracle DataServer. Set it to the DataServer executable you want to run: %DLC%\bin_orasrv.exe (in Windows), $DLC/bin/_orasrv (on UNIX) or one you created with the OEBuild files provided in the oebuild directory.
ORAVERSION(Optional)	The version of Oracle you are accessing (8, 9, or 10. Version 8 is the default). Set this environment variable when you are upgrading a DataServer application to Oracle 8.
PROBRKR(Optional)	The path name to the executable of the broker.
PROEXE(Optional)	The path name of the OpenEdge client executable. After setting this variable, you can run a customized executable (by default, _progres on UNIX) by entering pro at the system prompt.
TNS_ADMIN(Optional)	The pathname of the directory where listener.ora and tnsnames.ora configuration files are found for the server. Or for the client, where the tsnames.ora file is found.

Note: If you changed the value of an environment variable during a session, you might have to shut down the DataServer processes and restart them before the new value takes effect.

C

Sample Queries

This appendix contains sample queries and the information that the DataServer provides when you specify the `QUERY-TUNING (DEBUG SQL)` option. In each case, notes explain the DataServer and cursor behavior. The numbers in angle brackets (<*n*>) indicate cursor handles for SQL statements.

For details, see the following topics:

- Query 1
- Query 2
- Query 3
- Query 4
- Query 5
- Query 6
- Query 7
- Query 8
- Query 9
- Query 10
- Query 11
- Query 12
- Query 13

Query 1

In Query 1 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<3,4> The `WHERE` clause generated by the DataServer positions the cursor after the row retrieved by the first use of cursor <2> to retrieve `CUSTOMER 2`.

```
FIND customer 2.
DISPLAY customer.name customer.custnum customer.postalcode.
FIND NEXT customer.
DISPLAY customer.name customer.custnum customer.postalcode.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
   SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 53596
   SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((CUST_NUM = :1))  order by CUST_NUM
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call OCIHandleAlloc <0>
 OCI call omru   <3>
 OCI call OCIStmtPrepare <4>     sqlcrc = 3562
    SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((CUST_NUM > :p1)) order by CUST_NUM
 OCI call OCIStmtExecute <4>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <4>
       Cursor <4> Rows processed 20 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
      Total Rows processed     1 (previous executions)
 OCI call OCIHandleFree <0>
       Cursor <3> Rows processed 1 (last execution)
 OCI call OCIHandleFree <0>
```

Query 2

In Query 2 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<2> The DataServer uses this cursor for customer 2.

<5> The `WHERE` clause generated by the DataServer positions the cursor for `country-post` after `customer 2`. The `ORDER BY` clause uses the `PROGRESS_RECID` column as the final component to guarantee unique ordering.

```
FIND customer 2.
DISPLAY customer.name customer.custnum customer.postalcode.
FIND NEXT customer USE-INDEX Country-Post.
DISPLAY customer.name customer.custnum customer.postalcode WITH FRAME b.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 53596
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((CUST_NUM = :1))  order by CUST_NUM
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <3>
 OCI call OCIHandleAlloc <0>
 OCI call OCIHandleAlloc <0>
 OCI call omru   <4>
 OCI call OCIStmtPrepare <5>     sqlcrc = 49589
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##COUNTRY_POST) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((upper(COUNTRY) = upper(:p1) AND
      upper(POSTAL_CODE) = upper(:p2) AND PROGRESS_RECID > :p0) OR
      (upper(COUNTRY) = upper(:p1) AND upper(POSTAL_CODE) > upper(:p2)) OR
      (upper(COUNTRY) > upper(:p1))) order by upper(COUNTRY),
      upper(POSTAL_CODE), PROGRESS_RECID
 OCI call OCIStmtExecute <5>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <5>
       Cursor <5> Rows processed 20 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
      Total Rows processed     1 (previous executions)
 OCI call OCIHandleFree <0>
       Cursor <4> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <3> Rows processed 1 (last execution)
 OCI call OCIHandleFree <0>
```

Query 3

In Query 3 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<3> This cursor selects the PROGRESS_RECID column for a particular row by CUST_NUM.

```
FIND customer 2.
DISPLAY customer.name customer.custnum customer.postalcode.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ *
      FROM DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 53596
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((CUST_NUM = :1))  order by CUST_NUM
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <3>
       Cursor <3> Rows processed 1 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
 OCI call OCIHandleFree <0>
```

Query 4

In Query 4 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<3> The DataServer executes the FIND FIRST customer.

<4> The DataServer retrieves `customer 2`.

```
FIND FIRST customer.
DISPLAY customer.name customer.custnum customer.postalcode.
FIND customer 2.
DISPLAY customer.name customer.custnum customer.postalcode.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 1858
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 order by CUST_NUM
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <3>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <4>     sqlcrc = 53596
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID
      FROM DOCTEST.CUSTOMER T0 WHERE ((CUST_NUM = :1))  order by CUST_NUM
 OCI call OCIStmtExecute <4>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <4>
       Cursor <4> Rows processed 1 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
      Total Rows processed     1 (previous executions)
 OCI call OCIHandleFree <0>
       Cursor <3> Rows processed 20 (last execution)
 OCI call OCIHandleFree <0>
```

Query 5

In Query 5 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<3> The single lookahead cursor selects columns directly. It ignores the field list because the FOR EACH loop defaults to a SHARE-LOCK. Also, since FOR EACH loops do not guarantee order of retrieval, the DataServer has not added an ORDER BY clause. The DataServer called OCIStmtFetch to fetch an array of rows. The DataServer used the default cache-size of 8192. Since 472 bytes are required for each row, it used 8024 bytes of cache to fetch up to 17 rows each call. Processing the 85 rows in the CUSTOMER table required a total of 5 array fetches.

```
FOR EACH customer FIELDS (name custnum postalcode):
  DISPLAY customer.name customer.custnum customer.postalcode.
END.
```

```
OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
     USTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 60425
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID un
     ique_id_0,CUST_NUM,COUNTRY,NAME,ADDRESS,ADDRESS2,CITY,STATE,POSTA
     L_CODE,CONTACT,PHONE,SALES_REP,CREDIT_LIMIT,BALANCE,TERMS,DISCOUN
     T,COMMENTS,PROGRESS_RECID FROM DOCTEST.CUSTOMER T0
 OCI call OCIStmtExecute <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call omru   <3>
       Cursor <3> Rows processed 85 (last execution)
      Number of array fetches  5
      Number of rows fetched   85
      Number of array rows     17
      Number of array columns  18
      Number of tables         1
      Space for one row        472
      Requested cache size     8192
      Actual cache size used   8024
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
```

Query 6

In Query 6 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<3> The single lookahead cursor selects columns directly. It selects all columns because the query does not contain a field list.

```
FOR EACH customer NO-LOCK:
  DISPLAY customer.name customer.custnum customer.postal-code.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
     USTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 60425
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID un
     ique_id_0,CUST_NUM,COUNTRY,NAME,ADDRESS,ADDRESS2,CITY,STATE,POSTA
     L_CODE,CONTACT,PHONE,SALES_REP,CREDIT_LIMIT,BALANCE,TERMS,DISCOUN
     T,COMMENTS,PROGRESS_RECID FROM DOCTEST.CUSTOMER T0
 OCI call OCIStmtExecute <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call OCIStmtFetch <3>
 OCI call omru   <3>
       Cursor <3> Rows processed 85 (last execution)
      Number of array fetches  5
      Number of rows fetched   85
      Number of array rows     17
      Number of array columns  18
      Number of tables         1
      Space for one row        472
      Requested cache size     8192
      Actual cache size used   8024
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
```

Query 7

In Query 7 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<3> The cursor selects only the fields in the field-list. The default cache-size of 8192 is sufficient to hold 106 rows. A single fetch retrieves the entire CUSTOMER table.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK:
  DISPLAY customer.name customer.custnum customer.postalcode.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
     USTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 40887
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID un
     ique_id_0,CUST_NUM,NAME,POSTAL_CODE FROM DOCTEST.CUSTOMER T0
 OCI call OCIStmtExecute <3>
 OCI call omru   <3>

       Cursor <3> Rows processed 85 (last execution)
      Number of array fetches  1
      Number of rows fetched   85
      Number of array rows     105
      Number of array columns  4
      Number of tables         1
      Space for one row        78
      Requested cache size     8192
      Actual cache size used   8190
 OCI call OCIHandleFree <0>

       Cursor <2> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
```

Query 8

In Query 8 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<3> This is a standard cursor. The default cache size is 1024. Since the DataServer fetches only the PROGRESS_RECID column, it requires only 4 bytes for each row. A single fetch retrieves all 85 PROGRESS_RECID values in the CUSTOMER table.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK
  QUERY-TUNING (NO-LOOKAHEAD):
  DISPLAY customer.name customer.custnum customer.postalcode.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 57261
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID
      unique_id_0 FROM DOCTEST.CUSTOMER T0
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <2>
      < repeat for each row >
 OCI call OCIStmtExecute <2>
 OCI call omru   <3>

       Cursor <3> Rows processed 85 (last execution)
      Number of array fetches  1
      Number of rows fetched   85
      Number of array rows     256
      Number of array columns  1
      Number of tables         1
      Space for one row        4
      Requested cache size     1024
      Actual cache size used   1024
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
      Total Rows processed     84 (previous executions)
 OCI call OCIHandleFree <0>
```

Query 9

In Query 9 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch column values.

<3> This is a standard cursor. Note that the advantage of using a field list is lost by not using a lookahead cursor. The DataServer uses the schema comparison cursor to retrieve column values by the PROGRESS_RECID column. Only those fields mentioned in the field list are available to the client.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK
  QUERY-TUNING (NO-LOOKAHEAD):
  DISPLAY customer.name customer.custnum customer.postalcode.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
     USTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 57261
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID un
     ique_id_0 FROM DOCTEST.CUSTOMER T0
 OCI call OCIStmtExecute <3>
 OCI call omru   <2>
 OCI call OCIStmtExecute <2>
 OCI call omru   <2>
       < repeat for each row >
 OCI call OCIStmtExecute <2>
 OCI call omru   <3>
       Cursor <3> Rows processed 85 (last execution)
      Number of array fetches  1
      Number of rows fetched   85
      Number of array rows     256
      Number of array columns  1
      Number of tables         1
      Space for one row        4
      Requested cache size     1024
      Actual cache size used   1024
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 1 (last execution)
      Total Rows processed     84 (previous executions)
```

Query 10

In Query 10 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<3> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<4> The cursor is used to perform the join by the SQLDB. Since the query specifies NO-LOCK, this cursor selects the fields in the field list in addition to those that the client requires (T0.PROGRESS_RECID, T1.PROGRESS_RECID, T1.CUST_NUM). With the default cache size of 8192, processing the entire join requires 3 array fetches.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK,
  EACH order FIELDS (orderdate salesrep) OF customer NO-LOCK:
  DISPLAY customer.name customer.custnum customer.postalcode
    order.orderdate order.salesrep.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
     USTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 60664
     SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ * FROM DOCTEST.ORD
     ER_ T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <3>
 OCI call omru   <3>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <4>     sqlcrc = 51753
     SELECT T0.PROGRESS_RECID unique_id_0,T0.CUST_NUM,T0.NAME,T0.POSTA
     L_CODE,T1.PROGRESS_RECID unique_id_1,T1.ORDER_DATE,T1.SALES_REP F
     ROM DOCTEST.CUSTOMER T0,DOCTEST.ORDER_ T1 WHERE (T1.CUST_NUM = T0
     .CUST_NUM)
 OCI call OCIStmtExecute <4>
 OCI call OCIStmtFetch <4>
 OCI call OCIStmtFetch <4>
 OCI call omru   <4>
       Cursor <4> Rows processed 207 (last execution)
      Number of array fetches  3
      Number of rows fetched   207
      Number of array rows     87
      Number of array columns  7
      Number of tables         2
      Space for one row        94
      Requested cache size     8192
      Actual cache size used   8178
 OCI call OCIHandleFree <0>
       Cursor <3> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
       Cursor <2> Rows processed 0 (last execution)
 OCI call OCIHandleFree <0>
```

Query 11

In Query 11 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<3> The DataServer uses the cursor to compare schema information. It does not fetch any column values. Note that because the ORDER_ table contains a date, the DataServer does not reuse this cursor to fetch column values.

<4> The cursor is used to perform the join by the SQLDB. The join still requires a lookahead cursor.

<5> Since the query requests the ORDER_ row with a SHARE-LOCK, the DataServer must refetch each ORDER_ row to get all the columns. If the ORDER_ table did not have a record identifier (PROGRESS_RECID in this case), this query would fail. If you must retrieve the ORDER_ row with a SHARE-LOCK, removing the field list eliminates the need to refetch the row.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK,
  EACH order FIELDS (orderdate salesrep) OF customer SHARE-LOCK:
  DISPLAY customer.name customer.custnum customer.postalcode
    order.orderdate order.salesrep.
END.
```

```
 OCI call OCIStmtPrepare <2>     sqlcrc = 41633
     SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <2>
 OCI call omru   <2>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <3>     sqlcrc = 60664
     SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ * FROM
      DOCTEST.ORDER_ T0 WHERE PROGRESS_RECID = :rid
 OCI call OCIStmtExecute-DescribeOnly <3>
 OCI call omru   <3>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <4>     sqlcrc = 51753
     SELECT T0.PROGRESS_RECID unique_id_0,T0.CUST_NUM,T0.NAME,
      T0.POSTAL_CODE,T1.PROGRESS_RECID unique_id_1,T1.ORDER_DATE,
      T1.SALES_REP FROM DOCTEST.CUSTOMER T0,DOCTEST.ORDER_ T1 WHERE
      (T1.CUST_NUM = T0.CUST_NUM)
 OCI call OCIStmtExecute <4>
 OCI call OCIStmtFetch <4>
 OCI call OCIStmtFetch <4>
 OCI call OCIHandleAlloc <0>
 OCI call OCIStmtPrepare <5>     sqlcrc = 18715
     SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ ORDER_NUM,CUST_NUM
     ,TO_CHAR(ORDER_DATE,'YYYYMMDDHH24MISS'),TO_CHAR(SHIP_DATE,'YYYYMM
     DDHH24MISS'),TO_CHAR(PROMISE_DATE,'YYYYMMDDHH24MISS'),CARRIER,INS
     TRUCTIONS,PO,TERMS,SALES_REP,PROGRESS_RECID  FROM DOCTEST.ORDER_
     T0 WHERE PROGRESS_RECID = :rid
 OCI call omru   <5>
 OCI call OCIStmtExecute <5>
 < repeat for each row >
 OCI call OCIStmtExecute <5>
 OCI call omru   <4>
```

Query 12

In Query 12 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch CUSTOMER rows.

<3> The DataServer uses the cursor to compare schema information. It does not fetch any column values. Note that because the ORDER_ table contains a date, the DataServer does not reuse this cursor to fetch column values.

<4> The cursor is used to perform the join by the SQLDB. It uses a standard cursor for the join. Each row of the join requires 8 bytes of the cache because the join cursor fetches only the unique integer record identifiers.

<5> The DataServer uses this cursor to fetch ORDER_ rows by the PROGRESS_RECID column. It cannot use the schema comparison cursor (<39>) because the DataServer must perform a TO_CHAR conversion on the date columns.

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK,
  EACH order FIELDS (orderdate salesrep) OF customer NO-LOCK
    QUERY-TUNING (NO-LOOKAHEAD):
  DISPLAY customer.name customer.custnum customer.postalcode
    order.orderdate order.salesrep.
END.
```

```
OCI call OCIStmtPrepare <2>     sqlcrc = 41633
    SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM
      DOCTEST.CUSTOMER T0 WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <2>
OCI call omru   <2>
OCI call OCIStmtPrepare <3>     sqlcrc = 60664
    SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ * FROM
      DOCTEST.ORDER_ T0 WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <3>
OCI call omru   <3>
OCI call OCIStmtPrepare <4>     sqlcrc = 48763
    SELECT T0.PROGRESS_RECID unique_id_0,T1.PROGRESS_RECID
      unique_id_1 FROM DOCTEST.CUSTOMER T0,DOCTEST.ORDER_ T1
      WHERE (T1.CUST_NUM = T0.CUST_NUM)
OCI call OCIStmtExecute <4>
OCI call omru   <2>
OCI call OCIStmtExecute <2>
OCI call OCIStmtPrepare <5>     sqlcrc = 18715
    SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ ORDER_NUM,CUST_NUM
    ,TO_CHAR(ORDER_DATE,'YYYYMMDDHH24MISS'),TO_CHAR(SHIP_DATE,'YYYYMM
    DDHH24MISS'),TO_CHAR(PROMISE_DATE,'YYYYMMDDHH24MISS'),CARRIER,INS
    TRUCTIONS,PO,TERMS,SALES_REP,PROGRESS_RECID  FROM DOCTEST.ORDER_
    T0 WHERE PROGRESS_RECID = :rid
OCI call omru   <5>
OCI call OCIStmtExecute <5>
OCI call omru   <2>
OCI call OCIStmtExecute <2>
      < repeat, alternating handles 5 and 2 >
     Cursor <4> Rows processed 207 (last execution)
     Number of array fetches  2
     Number of rows fetched   207
     Number of array rows     128
     Number of array columns  2
     Number of tables         2
     Space for one row        8
     Requested cache size     1024
     Actual cache size used   1024
OCI call OCIHandleFree <0>
     Cursor <5> Rows processed 1 (last execution)
     Total Rows processed     206 (previous executions)
OCI call OCIHandleFree <0>
     Cursor <2> Rows processed 1 (last execution)
     Total Rows processed     203 (previous executions)
OCI call OCIHandleFree <0>
     Cursor <3> Rows processed 0 (last execution)
```

Query 13

In Query 13 presented in this section, note the following points related to cursor behavior:

<2> The DataServer uses the cursor to compare schema information and fetch CUSTOMER rows.

<3> The DataServer uses the cursor to compare schema information. It does not fetch any column values. Note that because the ORDER_ table contains a date, the DataServer does not reuse this cursor to fetch column values.

<4> The DataServer uses a lookahead cursor to select fields in the field list in addition to those required by the client.

<5> The lookahead cursor selects fields from the ORDER_ table that correspond to a particular CUSTOMER row (WHERE CUST_NUM = :1).

```
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK,
  EACH order FIELDS (orderdate salesrep) OF customer NO-LOCK
    QUERY-TUNING (NO-JOIN-BY-SQLDB):
  DISPLAY customer.name customer.custnum customer.postalcode
    order.orderdate order.salesrep.
END.
```

```
OCI call OCIStmtPrepare <2>     sqlcrc = 41633
    SELECT /*+ INDEX(T0 CUSTOMER##PROGRESS_RECID) */ * FROM DOCTEST.C
    USTOMER T0 WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <2>
OCI call omru   <2>
OCI call OCIStmtPrepare <3>     sqlcrc = 60664
    SELECT /*+ INDEX(T0 ORDER_##PROGRESS_RECID) */ * FROM DOCTEST.ORD
    ER_ T0 WHERE PROGRESS_RECID = :rid
OCI call OCIStmtExecute-DescribeOnly <3>
OCI call omru   <3>
OCI call OCIStmtPrepare <4>     sqlcrc = 40887
    SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM)  */ PROGRESS_RECID un
    ique_id_0,CUST_NUM,NAME,POSTAL_CODE FROM DOCTEST.CUSTOMER T0
OCI call OCIStmtExecute <4>
OCI call OCIStmtPrepare <5>     sqlcrc = 17552
    SELECT /*+ INDEX_ASC(T0 ORDER_##CUST_ORDER)  */ PROGRESS_RECID un
    ique_id_0,CUST_NUM,ORDER_DATE,SALES_REP FROM DOCTEST.ORDER_ T0 WH
    ERE (CUST_NUM = :1)
OCI call OCIStmtExecute <5>
 < repeat for each row >
     Cursor <4> Rows processed 85 (last execution)
     Number of array fetches  1
     Number of rows fetched   85
     Number of array rows     105
     Number of array columns  4
     Number of tables         1
     Space for one row        78
     Requested cache size     8192
     Actual cache size used   8190
OCI call OCIHandleFree <0>
     Cursor <5> Rows processed 0 (last execution)
     Total Rows processed     207 (previous executions)
     Number of array fetches  0
     Number of rows fetched   0
     Number of array rows     141
     Number of array columns  4
     Number of tables         1
     Space for one row        58
     Requested cache size     8192
     Actual cache size used   8178
```

D

Building DataServer Executables

This appendix describes how to build DataServer executables for all possible configurations.

Note: From 12.0 onwards, if you are an AIX64 customer - building the DataServer executables is no longer needed. OpenEdge provides executables like _progres, _orasrv, _mproapsv which can be used directly. These executables dynamically load Oracle Client libraries pointed by ORACLE_HOME. However, the environment variable ORACLE_HOME needs to be set to the top-level directory of **Oracle Client Installation**. With this change, from 12.0 onwards progress will stop shipping the build scripts needed for building executables.

For details, see the following topics:

- Overview of the building process
- Requirements for Building the DataServer Executables
- Building executables for the UNIX Client
- Building executables for the UNIX Host

Overview of the building process

OpenEdge provides a single set of executables (both client and server) that will access any supported version of Oracle. These executables dynamically locate and load Oracle shared libraries on those UNIX platforms where Oracle supports shared objects. For information on whether of not your version of Oracle supports shared objects, see your Oracle documentation or system administrator.

The single executable dynamically locates and loads the Oracle shared libraries. This means there is no need to set the system dynamic library path environment variable. However, you must set the environment variable ORACLE_HOME by specifying the pathname of the directory on your system where Oracle is installed. On some 64-bit systems, ORASOPATHNAME must also be set to the complete file specification of the Oracle client shared library.

Whether or not you can use the single executable depends on your version of UNIX and your version of Oracle. There are two possible scenarios:

1. Oracle supports shared objects on your version of UNIX and OpenEdge is able to dynamically load the Oracle shared object when the DataServer connects to your Oracle database. In this case you do not need to build any executables. You do not need to set the system dynamic library path environment variable. You can use _progres as the client executable and _orasrv as the server executable.
2. Oracle does not provide a shared object. You must build your server executable. You can also build a self-service client executable (orarx). Refer to the following sections of this appendix for instructions. Once you have built your server executable (_orasrv), you can use _progres to access a remote DataServer only. Because you statically link the necessary Oracle code when you build your server, you do not need to set the LD_LIBRARY_PATH environment variable.

The following table lists possible UNIX client/server configurations and describes the process for setting up the client and server modules and indicating which executables need to be built.

Table 73: Building DataServer processes

Client	Server	Building and configuring
UNIX	Local UNIX	Use a single executable on a UNIX client process that includes the Oracle DataServer. Use this configuration for local access or if you want the DataServer to access the remote Oracle instance through Oracle Networking.
UNIX	Remote UNIX	On the UNIX client machine, use a client executable that includes the remote Oracle DataServer. Building the client is optional. By default, the UNIX client executable supports access to the remote DataServer.On the UNIX host machine, use two executables: • The remote DataServer broker • The remote Oracle DataServer
UNIX	Remote Windows	On the UNIX client machine, use a client executable that includes the remote Oracle DataServer. By default, the UNIX client executable supports access to the remote DataServer.In the Windows host machine, configure the default broker executable using OpenEdge Management or OpenEdge Explorer.
Windows	Remote UNIX	In Windows, use the default OpenEdge client executable (prowin32.exe).On the UNIX host machine, use two executables: • The remote DataServer broker • The remote Oracle DataServer

Requirements for Building the DataServer Executables

OpenEdge provides scripts to build customized executables, including the DataServer for Oracle on those platforms where a dynamically loadable shared library is unavailable. Prior to running these scripts, the environment must be configured.

To set up your environment to run the OEBuild utility scripts:

1. You must run the scripts to create executables on the machine where you plan to run them. For example, if you are building a remote DataServer configuration, to build the broker and the DataServer, run the scripts on the host. To build the client, run the scripts on the client machine. If you are building executables for deployment, be sure to build the executables on the same platform where your users will run them.
2. Make sure that you have a linker installed on your system.
3. Verify settings for OpenEdge environment variables. Set `DLC` to the directory where you have installed OpenEdge.
4. Verify that the `buildenv.sh` script (located in the `$DLC/oebuild/make` directory) includes the appropriate environment variable settings for your system.

   The build script calls the `buildenv.sh` script. The `buildenv.sh` script sets default values for environment variables that point to objects, libraries, and options involved in linking executables. You can edit this script to customize environments for different sites.

   A log file, `buildenv.log`, is also created. It includes information about how your DataServer environment is configured. This information is useful to you and Technical Support when troubleshooting connection problems. Here is part of a sample `buildenv.log` file:

```
Build Environment: Wed Aug  6 19:32:28 EDT 2003
------------------------------------------------------------
User: uid=488(doctest) gid=110(rdl) groups=377(dba),378(oracle),379(oracle6)
Working Directory:              /scratch/doctest/AIX
Machine OS:                     AIX
Machine OS Version:             2
Machine Type:                   IBM,7025-F50
DLC Setting:                    /usr1/docqa/100a/dlc

User Defined ORALIB: No, buildenv script defined

ORALIB Setting:
------------------------------------------------------------
/gateways/aix/oracle/8.1.7/lib/libclntsh.a

ORACLE_HOME Setting:
------------------------------------------------------------
/gateways/aix/oracle/8.1.7

LIBPATH Setting:
------------------------------------------------------------

PATH Setting:
------------------------------------------------------------
:/usr/bin/X11:/usr/dt/bin:/usr/bin:/etc:/usr/sbin:/usr/ucb:/sbin:
/usr/rdl/sc:/usr/rdl/aix/bin:/tools/bin:/tools/aix/bin:
/usr1/docqa/100a/dlc/bin:/usr/local/bin:/users/devp/doctest/bin:.
PROPATH Setting:
------------------------------------------------------------
:/usr1/docqa/100a/dlc/tty:/usr1/docqa/100a/dlc
```

   For more information on environment variables, see *Configure OpenEdge*.

After setting up the build environment, you can run the build scripts that create the executables required by the various DataServer configurations. The following sections describe how to build, set up, and run client and host DataServer modules.

Building executables for the UNIX Client

This section contains instructions for building client executables. The script is located in the `oebuild/make` directory under the root of your OpenEdge install.

To build a UNIX client executable:

1. Open a terminal and invoke a Bourne shell (`/bin/sh`) on the system where OpenEdge is installed.
2. Change to user '`root`'.
3. Set and export the environment variable DLC to the OpenEdge installation directory. For example:

```
DLC=/usr/OpenEdge/dlc
export DLC
```

4. Optionally set and export the environment variable `IMAGE` to the full pathname of the executable to be generated. By default the client executable is named:

```
$DLC/oebuild/orarx
```

5. Set the environment variable `ORACLE_HOME` to the top level directory of your Oracle installation. For example:

```
ORACLE_HOME=/usr/oracle/9i
export ORACLE_HOME
```

6. Verify that the variable `PATH` contains the directory to the linker.
7. Unset all the library path variables for your operating system. The script sets these variables as required. For example:

```
unset LD_LIBRARY_PATH LIBPATH SHLIB_PATH
```

8. Run the script `$DLC/oebuild/make/build_orarx.sh`. If you did not set the `IMAGE` variable, the Oracle DataServer client executable is built as `$DLC/oebuild/orarx`, otherwise the executable is built as specified by `$IMAGE`.

Note: To use your new executable once it is built, you must replace your default executable or set the `PROEXE` environment variable.

Replacing your default client executable:

1. Make a backup copy of the original executable. For example:

```
cp $DLC/bin/_progres $DLC/bin/_progres.orig
```

2. Copy the newly built executable into your installed `bin` directory. For example:

```
cp $DLC/oebuild/orarx $DLC/bin/_progres
```

3. Change permissions on the newly created executable to include the Set UID bit. For example:

```
chmod u+s $DLC/bin/_progres
```

Identifing your new executable with an environment variable and without relocating

:

1. Define the environment variable `PROEXE`. For example:

```
PROEXE=$DLC/oebuild/orarx
export PROEXE
```

2. Change permissions on the newly created executable to include the Set UID bit. For example:

```
chmod u+s $DLC/oebuild/orarx
```

Building executables for the UNIX Host

This section contains instructions for building host executables. The script is located in the `oebuild/make` directory under the root of your OpenEdge install.

Building a UNIX host executable

1. Open a terminal and invoke a Bourne shell (/bin/sh) on the system where OpenEdge is installed.
2. Change to user 'root'.
3. Set and export the environment variable DLC to the OpenEdge installation directory. For example:

```
DLC=/usr/OpenEdge/dlc
export DLC
```

4. Optionally set and export the environment variable IMAGE to the full pathname of the executable being generated. By default the client executable is named:

```
$DLC/oebuild/_orasrv
```

5. Set the environment variable ORACLE_HOME to the top level directory of your Oracle installation. For example:

```
 ORACLE_HOME=/usr/oracle/9i
export ORACLE_HOME
```

6. Verify that the variable PATH contains the directory to the linker.
7. Unset all the library path variables for your operating system. The script sets these variables as required. For example:

```
unset LD_LIBRARY_PATH LIBPATH SHLIB_PATH
```

8. Run the script $DLC/oebuild/make/build_orasrv.sh. If you did not set the IMAGE variable, the Oracle DataServer server executable is built as:

```
$DLC/oebuild/_orasrv, otherwise the executable is built as specified by $IMAGE
```

Note: To use your new executable once it is built, you must set the ORASRV environment variable.

Replacing your default server executable

1. Make a backup copy of the original executable, if one exists. For example:

```
cp $DLC/bin/_orasrv $DLC/bin/_orasrv.orig
```

2. Copy the newly built executable into your installed `bin` directory. For example:

```
cp $DLC/oebuild/_orasrv $DLC/bin/_orasrv
```

3. Change permissions on the newly created executable to include the Set UID bit. For example:

```
chmod u+s $DLC/bin/_orasrv
```

4. Define the environment variable `ORASRV` to your executable. For example:

```
ORASRV=$DLC/bin/_orasrv
export ORASRV
```

Identifying your new executable with an environment variable and without relocating

1. Define the environment variable `ORASRV`. For example:

```
ORASRV=$DLC/oebuild/_orasrv
export ORASRV
```

2. Change permissions on the newly created executable to include the Set UID bit. For example:

```
chmod u+s $DLC/oebuild/_orasrv
```

Note: Once you create your executables, see Configuring the DataServer on page 135 for instructions to configure and use the DataServer executables.

E

DataServer Command Line Utilities and Startup Parameters

This appendix describes the following utilities and parameters that you use to configure, manage, start, and stop the DataServer host and client:

- Command line utilities allow you to start, stop, and configure installed OpenEdge DataServer for Oracle components. See *Configure OpenEdge* for additional information about the utilities and their role and relationship to other system administration facilities.
- Startup parameters on UNIX and in Windows allow you to start and manage DataServer clients. See *Startup Command and Parameter Reference*.
- *Startup Command and Parameter Reference* for additional information about syntax and usage.

For details, see the following topics:

- Command line utilities for the DataServer

Command line utilities for the DataServer

This section describes the utilities you use to configure, manage, start, and stop a DataServer. The utilities are presented in alphabetical order.The purpose, syntax, and primary parameters for each operating system are detailed.

NSCONFIG utility

Use the NSCONFIG utility to help you debug existing NameServer configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with a NameServer configuration, and checks that the syntax and values are valid.

The NSCONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network.

Syntax

Operating System	Syntax

UNIX
Windows
	`nsconfig`
	`[`
	`[`
	`[ -name name-server]`
	`[ -propfile path-to-properties-file]`
	`[ -validate ]`
	`]`
	`| -help`
	`]`

Parameters

`-name` *`name-server`*

Specifies which existing NameServer configuration to examine. The name must match the name of an existing NameServer configuration in the specified properties file. If you do not specify a NameServer, the NSCONFIG utility analyzes all NameServer configurations defined in the properties file specified by the `-propfile` parameter.

`-propfile` *`path-to-properties-file`*

Specifies a filename or pathname to a file that contains the property settings to be validated, for example `test.properties`. If a filename or pathname is not specified, it defaults to the installation version of the `ubroker.properties` file, such as:

- `%DLC%\properties\ubroker.properties` in Windows
- `$DLC/properties/ubroker.properties` on UNIX

`-validate`

Checks the syntax and values of property settings defined in the specified properties file.

`-help`

Displays command-line help.

Notes

- A single NameServer can simultaneously support all of the AppServer, WebSpeed, and DataServer products using OpenEdge Management or OpenEdge Explorer.
- The `ubroker.properties` file stores all the configuration definitions for each instance of the NameServer, AppServer, DataServer, and WebSpeed Transaction Server products. Each configuration definition contains environment variables, registry entries if Windows, and property settings for each product instance. OpenEdge Management or OpenEdge Explorer and certain command-line utilities such as NSCONFIG, use this file to store, validate, and manage the configurations for the products.

  The `ubroker.properties` file is installed in the `properties` subdirectory of the OpenEdge installation directory. For example, `$DLC/properties/ubroker.properties` on UNIX, or `%DLC%\properties\ubroker.properties` in Windows.

  The file consists of a hierarchical structure of configuration entities, where parent entities provide configuration information that you can override or extend in each child entity. Each configuration entity has a name that begins the entity definition, and the definition contains configuration settings for one or more product instances. For example, the NameServer configurations in `ubroker.properties` can include:

Configuration entityname	Configuration entityfunction
`[UBroker]`	Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.
`[NameServer]`	Defines default property settings for all instances of a NameServer.
`[NameServer.`*`product-instance-name`*`]`	Defines property settings for this instance of a NameServer. The `ubroker.properties` file can contain several of these entities, each with a unique *`product-instance-name`*.

Parent entities provide default values for all of their child entities. For example, the parent `[UBroker]` contains a set of definitions that can be inherited by its child `[NameServer]`, and then again by its child `[NameServer.`*`product-instance-name`*`]`. However, at any child level, a redefinition of any value supersedes the default value of its parent. All children from the redefinition level down inherit this new value.

If you do not have access to OpenEdge Management or OpenEdge Explorer and must configure products within a UNIX or Windows environment, you must edit the `ubroker.properties` file using a text editor such as vi or Notepad.

If you want to manually edit this file to create or modify a product configuration, begin by making a backup copy from the installed `ubroker.properties` file (and naming it for example, `test.properties`).

Once you edit the properties file, use the relevant validation utility such as ORACONFIG to validate the changes and make sure there are no syntax errors or conflicts.

NSMAN utility

Use the NSMAN utility to control the operation of a configured NameServer. The utility allows you to start a NameServer, query its status, and shut down a NameServer.

Syntax

Operating System	Syntax

UNIX
Windows
	nsman
	{
	{ -name *name-server*
	{ -kill | -start | -stop | -query }
	[
	-host *host-name* -user *user-name* | -user *user-name*
	]
	[-port *port-number*]
	}
	| -help
	}

Parameters

-name *name-server*

Specifies the name of the NameServer. This parameter is required.

-kill

Stops and removes the NameServer from memory, no matter what it is doing.

-start

Starts the NameServer.

-stop

Tells the NameServer to stop itself.

-query

Queries the NameServer for its status.

-host *host-name*

Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.

-user *user-name*

Specifies a user name and prompts for a password. A user name and password are required only when you use the -host parameter and specify a remote host name. If you specify a remote host name with the -host parameter, but do not specify a user name with the -user parameter, you receive a prompt for a user-name and password.

`-port` *`port-number`*

Specifies the port number of the machine on which the AdminServer is running. If a port number is not specified, it defaults to 20931.

`-help`

Displays command-line help.

Notes

- A single NameServer can simultaneously support all of the AppServer, WebSpeed, and DataServer products.
- When you specify a user name with the `-user` parameter, Windows supports three different formats:
  - A user name as a simple text string, such as "`mary`," implies a local user whose user account is defined on the local Windows server machine, which is the same machine that runs the AdminServer.
  - A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer, except the user name explicitly references the local machine domain, for example "`.\mary`".
  - A user name as a user account on a specific Windows domain. The general format is *`Domain`*`\`*`User`*, in which the *`User`* is a valid user account defined within the domain and the *`Domain`* is any valid Windows Server, including the one where the AdminServer is running.

ORACONFIG utility

Use the ORACONFIG utility to help you debug existing OpenEdge DataServer for Oracle configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with an OpenEdge DataServer for Oracle configuration, and checks that the syntax and values are valid.

The ORACONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network.

Syntax

Operating System	Syntax
UNIX Windows	

Parameters

`-name` *`DataServer-name`*

Specifies which existing OpenEdge DataServer for Oracle configuration to examine. The name must match the name of an existing OpenEdge DataServer for Oracle configuration defined in the specified properties file. If you do not specify a DataServer by name, the ORACONFIG utility analyzes all OpenEdge DataServer for Oracle configurations defined in the properties file specified by the `-propfile` parameter.

`-propfile` *`path-to-properties-file`*

Specifies a filename or pathname to a file that contains the property settings to be validated, for example `test.properties`. If a filename or pathname is not specified, it defaults to the installation version of the `ubroker.properties` file, such as:

- `%DLC%\properties\ubroker.properties` in Windows
- `$DLC/properties/ubroker.properties` on UNIX

`-validate`

Checks the syntax and values of property settings defined in the specified properties file.

`-help`

Displays command-line help.

Notes

- The `ubroker.properties` file stores all the configuration definitions for each instance of the NameServer, AppServer, DataServer, and WebSpeed Transaction Server products. Each configuration definition contains environment variables, registry entries if Windows, and property settings for each product instance. OpenEdge Management or OpenEdge Explorer and certain command-line utilities such as ORACONFIG use this file to store, validate, and manage the configurations for the products.

- The `ubroker.properties` file is installed in the `properties` subdirectory of the OpenEdge installation directory. For example, `$DLC/properties/ubroker.properties` on UNIX, or `%DLC%\properties\ubroker.properties` in Windows.
- The file consists of a hierarchical structure of configuration entities, where parent entities provide configuration information that you can override or extend in each child entity. For more information about the `ubroker.properties` file, see *Configure OpenEdge*.

ORAMAN utility

Use the ORAMAN utility to control the operation of a configured OpenEdge DataServer for Oracle. The utility allows you to start a broker, query its status, start and stop additional DataServer servers, and shut down the broker.

Syntax

Operating System	Syntax
UNIX Windows	`oraman { { -name DataServer-name { -kill | -start | -stop | -query } [ -host host-name -user user-name | -user user-name ] [ -port port-number] } | -help }`

Parameters

`-name DataServer-name`

Specifies the name of a DataServer. This parameter is required.

`-kill`

Stops and removes the DataServer from memory, no matter what it is doing.

`-start`

Starts the DataServer.

`-stop`

Tells the DataServer to stop itself.

`-query`

Queries the DataServer for its status.

`-host` *`host-name`*

Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.

`-user` *`user-name`*

Specifies a user name and prompts for a password. A user name and password are required only when you use the `-host` parameter and specify a remote host name. If you specify a remote host name with the `-host` parameter, but do not specify a user name with the `-user` parameter, you receive a prompt for a user name and password.

`-port` *`port-number`*

Specifies the port number of the machine on which the AdminServer is running. If a port number is not specified, it defaults to 20931.

`-help`

Displays command-line help.

Notes

When you specify a user name with the `-user` parameter, Windows supports three different formats:

- A user name as a simple text string, such as "`mary`," implies a local user whose user account is defined on the local server machine, which is the same machine that runs the AdminServer.
- A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer, except the user name explicitly references the local machine domain, for example "`.\mary`".
- A user name as a user account on a specific Windows domain. The general format is *`Domain`*`\`*`User`*, in which the*`User`* is a valid user account defined within the domain and the*`Domain`* is any valid Windows Server, including the one where the AdminServer is running.

PROBRKR command line syntax

Starts the DataServer broker. To use the DataServer from a remote client, you must first start the broker. Once you start the broker, it can receive the client requests and spawn the appropriate DataServer.

Syntax

Operating System	Syntax
UNIX Windows	`probrkr -S service-name [ -H host-name]`

Parameters

`service-name`

Specifies the name of the broker process on the host machine.

`host-name`

Specifies the name of the machine where the DataServer broker is installed. The default value is the current host.

Notes

- See *Startup Command and Parameter Reference* for more details on the Server Name (`-S`) and Host Name (`-H`) startup parameters.
- You can use any of the startup parameters with the PROBRKR command. See *Startup Command and Parameter Reference* for details.
- The Oracle DataServer server process (`_orasrv`) will inherit its environment from the broker. Prior to starting the broker, ensure that the `ORASRV` environment variable is set to the name of the server executable, including the full path. See Configuring the DataServer on page 135 for more information on setting the `ORASRV` environment variable.

PROSHUT command

PROSHUT provides an interface to shut down an OpenEdge database server, DataServer broker, and individual OpenEdge user processes. Best practices suggest that before you shut down the server, have all application users quit their OpenEdge sessions. If necessary, you can disconnect users by using the PROSHUT command's Disconnect a User or Unconditional Shutdown parameters.

Syntax

Operating System	Syntax
UNIX Windows	`proshut { db-name | -Gw } [ -S service-name]...`

Parameters

db-name

Specify the name of the schema holder to shut down its server, if it was started in multi-user mode.

`-Gw`

For DataServers, specifies that the DataServer broker is to be shut down.

`-S` *`service-name`*

Specifies the database server or broker process. You must specify the service name if you issue the shutdown command from a machine other than the host.

Notes

For a complete listing of PROSHUT parameters and their functions, see *Manage the OpenEdge Database*.

Index

-1 parameter *172*
4GLTrans log entry type *187*

A

Abbreviated index *76*
ABL to SQL Correlation
    PRGRS_LOG_ABL switch *193*
ABL transaction logging
    log entry types *187*
Add additional index entries here (if needed)
    Nest index terms for sub entries *151*
Aggregates
    views *45*, *47*
Analyzing application execution with Enhanced Logger *182*
ARRAY-MESSAGE option *128*
Arrays
    joins *123*
Auto-connect
    failure *199*

B

BEGINS operator *76*
BFILE *50*
Bind variables
    queries *128*
BIND-WHERE option *128*
Blanks
    leading and trailing *43*
Broker
    running multiple *137–138*, *166*
    starting in Windows *169*
Buffers
    defining for stored procedures *99*
BY option
    joins *123*

C

-c startup parameter *177*
-cache startup parameter *177*
CACHE-SIZE option *128*
Caching
    schema *24*
Case-insensitive indexes *44*, *236*
CHARACTER data type *50*
Character sets
    changing in schema holders *233*
Clear dynamic temp-tables
    defined *106*
CLOB data type *62*
CLOSE QUERY statement *71*
CLOSE STORED-PROCEDURE statement
    complete syntax reference *87*
    retrieving parameter values *97*
CLOSED STORED-PROCEDURE statement
    complete syntax reference *89*
    description *82*
Code pages
    changing in schema holders *233*
    collation table *39*
    conversion table *39*
Column name *237*
Column width *237*
COMPILE statement *75*
CONNECT statement *171*, *199*
Connecting schema holders *171*
Connection failures
    with ORACLE *199*
Connection parameters
    network options *172*
Connects log entry type *187*
CONTAINS operator *76*
COUNT-OF function *76*
-cpinternal startup parameter *39*, *140*, *177*, *239*
-cpstream startup parameter *140*, *177*, *239*
CREATE statement *64*, *76*
Creating a temp-table layout plan
    options *104*
Creating a temp-table plan layout *104*
Creating at temp-table plan layout
    dynamic temp-table in a prepared state *106*
    dynamic temp-table in an unprepared state *106*
Creating schema holders *149*
CURRENT-VALUE function *76*
CURRENT-VALUE statement *76*
Cursor log entry type *187*
Cursor repositioning *71*, *75*
Cursors
    defined *70*
    stored procedures *97*

D

Data definitions *24*
Data Dictionary *229*
Data output and retrieval options
    results retrieved from a database *94*
    return codes *94*

Data output and retrieval options *(continued)*
values of output parameters you define when creating a procedure *94*
Data Service *177*
Data types
changing *230*
CHARACTER *50*
CLOB *62*
DATE *53*, *63*
DECIMAL *50*
display format *230*
INT64 *50*
INTEGER *52*
LOB *62*
LOGICAL *52*
LONG *50*, *74*
NUMBER *50*
RAW *53*
stored procedures *82*
Database conversion
OpenEdge-to-ORACLE *236*, *239*
Database objects
defined *36*
Database Type (-dt) startup parameter *177*
Databases
naming conventions *238*
objects *155*, *217*
OpenEdge objects *36*
schema *24*
triggers *47*
DataServer (-Dsrv) startup parameter
qt_cache_size *74*
stored procedures *82*
DataServer logging level and log entry types setting *191*
DataServer logging using qt_debug logging
$DSLOGDIR environment variable *202*
$UNIQUE_DSLOG environment variable *203*
Handling log file creation errors *202*
Naming the log file *201*
Restrictions *202*
DataServers
architecture *22*
broker *138*, *200*
building executables *277*
compiling applications *75*
cursor repositioning *71*
internal logic *23*
log file *139*, *165*, *169*, *259*
networks *27*, *29*
record creation *64*
starting a broker *290*
starting UNIX client *170*
starting Windows client *171*
transactions *68*, *79*
DATE data type *53*, *63*
-db startup parameter *172*
DB.Connects log entry type *187*
DBA privileges *149–150*
DBE DataServer
startup parameters *140*, *177*, *239*
DBRESTRICTIONS function *76*
Deadlock
stored procedures *82*
DEBUG EXTENDED option *128*
DEBUG SQL option *128*
DECIMAL data type *50*
DEFINE BROWSE statement *74*
DEFINE BUFFER statement *98*
Defining a view to use as a buffer *99*
Defining constraints
activating or deactivating constraints *216*
check constraint definition *214*
deleting constraint *217*
foreign key constraint definition *213*
modifying constraint *216*
primary key constraint definition *210*
unique key constraint definition *212*
Deleting schema holders *234*
Deploying schema holders *159*
Direct Connect *177*
Display formats *230*
Distributed databases
connecting *175*
synonyms *49*
DLC environment variable *259*
DS.Cursor log entry type *187*
DS.Performance log entry type *187*
DS.QryInfo log entry type *187*
DSLOG-MANAGER system handle attributes and methods *184*
DSLOGDIR environment variable *139*, *165*, *169*, *259*
-Dsrv startup parameter *74*, *178*
-dt startup parameter *177*
Dynamic temp-table *88*

E

Environment variables
DLC *259*
DSLOGDIR *139*, *165*, *169*, *259*
ORACLE_HOME *139*, *151*, *165*, *169*, *259*
ORACLE_SID *139*, *151*, *165*, *169*, *200*, *259*
ORASRV *139*, *169*, *259*
ORAVERSION *259*
PROBRKR *139*, *169*, *259*
PROEXE *259*
TNS_ADMIN *259*
UNIX *277*
Error handling *69*
Error messages
incomplete *155*, *217*
ORACLE *155*, *217*

Error messages *(continued)*
related to the LOAD-RESULT-INTO phrase *112*
troubleshooting *199*
Errors
NO-ERROR option *110*
EXCLUSIVE-LOCK option *66*, *123*, *128*
External data types
defined *50*

F

Field extents. *62*
See also Arrays
Field lists
updating records *122*
Field properties
modifying *230*
FIELDS option
SHARE-LOCK *76*
FIND LAST statement *75*
FIND PREV statement *75*
FIND statement
QUERY option *76*
FOR EACH statement *41*
Formats *230*, *237*
Functions
CURRENT-VALUE *76*
DBRESTRICTIONS *76*
MATCHES
double-strings *76*
NEXT-VALUE *76*
PROC-HANDLE *100*
PROC-STATUS *96*
RECID *64*, *72*
ROWID *72*, *155*, *217*
ROWID data type *50*
SETUSERID *76*
USERID *76*, *175*

H

-H startup parameter *172*
handles
SESSION:TIME-SOURCE *76*
Handling errors *110*
HINT option *128*
Hints *41*, *121*, *128*
Host Name (-H) startup parameter *172*

I

Index Cursors (-c) startup parameter *177*
Index Hint (-noindexhint) startup parameter *41*, *128*, *178*
Index hints *41*
INDEX-HINT option *128*
Indexes
abbreviated *76*
case-insensitive *44*, *236*
INDEX-INFORMATION *41*
repositioning *44*
INPUT option
stored procedures *89*
INPUT-OUTPUT option
retrieving parameter values *97*
stored procedures *89*
Instances
defined *23*
INTEGER data type *52*
Internal Code Page (-cpinternal) startup parameter *39*

J

JOIN-BY-SQLDB option *123*, *128*
Joins *45–46*

L

-ld startup parameter *177*
Literals
queries *128*
LOAD-RESULT-INTO phrase
description *82*, *88*
Loading result sets into temp-tables *102*
LOB data type *62*
Local machines
defined *135*
Locks
stored procedures *82*
Log entry types
4GLTrans *187*
Connects *187*
Cursor *187*
DB.Connects *187*
DS.Cursor *187*
DS.Performance *187*
DS.QryInfo *187*
Performance *187*
SQL *187*
Trans *187*
Log files *139*, *165*, *169*, *259*
LOG-MANAGER system handle attributes and methods *183*
Logging levels *191*
LOGICAL data type *52*
Logical Database Name (-ld) startup parameter *177*
Logical database names
schema holders *228*
LONG column *74*
LONG data type *50*
Lookahead cursors *128*
LOOKAHEAD option *128*

M

Managing server attributes
  foreign database constraints *209*
MATCHES function
  double-byte strings *76*
mergeprop utility *31*
Messages
  ORACLE *155*, *217*
Mixed arrays
  defined *104*
Modifying schema holders *228*
Multi-table views *46*

N

Naming conventions *37*, *238*
National Language Support (NLS)
  DataServer environment *140*
  stored procedure *78*
Net 8 *175*
NEXT-VALUE function *76*
NO-ARRAY-MESSAGE option *128*
NO-BIND-WHERE option *128*
NO-DEBUG option *128*
NO-ERROR option
  description *89*
NO-INDEX-HINT option *41*, *128*
NO-JOIN-BY-SQLDB option *123*, *128*
NO-LOOKAHEAD option *128*
-noindexhint startup parameter *41*, *128*, *178*
-nojoinbysqldb startup parameter *123*, *128*, *178*
Non-unique table names *37*
NOT NULL *62*
NSMAN *285*
NUMBER data type *50*
Number of Databases (-h) startup parameter *155*, *206*, *217*

O

OE Build utility *277*
OF option *41*
OPEN QUERY statement *64*, *76*
OpenEdge
  CHARACTER data type *50*
  Data Dictionary *229*
  data types *50*
  database objects *36*
  DECIMAL data type *50*
  INTEGER data type *52*
  LOGICAL data type *52*
  naming conventions *37*, *238*
  ORACLE utilities *155*
  size limitations *236*
  triggers *47*
OpenEdge DB-to-ORACLE utility
  batch mode *239*
OpenEdge Management or OpenEdge Explorer
  Data Service *177*
  Direct Connect *177*
OpenEdge Management or OpenEdge Explorer tool
  definition *31*, *147*
OpenEdge-to-MS SQL Server utility
  parameters *239*
ORACLE
  data types
    external *50*
    internal *50*
  DATE data type *53*
  DBA privileges *149*
  distributed database *155*, *175*, *217*
  LONG data type *50*
  naming conventions *37*, *238*
  Net 8 *175*
  non-unique table names *37*
  NOT NULL *62*
  NUMBER data type *50*
  password *76*, *150*, *172*, *200*
  permissions *149*
  RAW data type *53*
  record locking *66*
  ROWID *50*
  ROWID data type *50*
  running *151*
  schema objects *36*
  size limitations *236*
  SQL*Net *175*
  triggers *47*
  user ID *76*, *150*, *172*, *200*
  views *45*
  WHERE clause *76*
ORACLE 8
  creating schema holder *155*, *217*
ORACLE Call Interface *23*
ORACLE utilities *155*
ORACLE_HOME environment variable *139*, *151*, *165*, *169*, *259*
ORACLE_SID environment variable *139*, *151*, *165*, *169*, *200*, *259*
ORACONFIG *287*
ORAMAN *289*
ORASRV
  setting in ProControl *137*
ORASRV environment variable *139*, *169*, *259*
ORAVERSION environment variable *259*
ORDERED-JOIN option *128*
Ordering results
  USE-INDEX *41*
OUTPUT option
  retrieving parameter values *97*
  stored procedures *89*

P

-P startup parameter *172*
package
  definition *82*
PARAM option
  retrieving parameter values *97*
  stored procedures *89*
Param phrase
  defined *82*
PARAM phrase
  description *82*, *89*
Parameter files *172*
Password *172*
Password (-P) startup parameter *172*
Passwords *76*, *150*, *200*
Performance
  analyzing *196*
Performance log entry type *187*
Permissions *149*
Physical Database Name (-db) startup parameter *172*
Physical database names *172*
PRO*C *23*
PRO/SQL utility *234*
PROBRKR command *290*
PROBRKR environment variable *139*, *169*, *259*
PROC-HANDLE function *100*
PROC-HANDLE phrase
  description *82*, *88*
PROC-STATUS function *96*
PROC-STATUS phrase
  description *82*, *88*
Procedures
  compiling *75*
PROEXE environment variable *259*
Progress_RECID column *74*
PROLOAD environment variable *277*
PROSHUT command *291*
protoora utility *239*

Q

qt_debug logging *196*
qt_bind_where option *178*
qt_cache_size option *178*
qt_lookahead option *178*
qt_no_bind_where option *178*
qt_no_lookahead option *178*
Queries
  records not visible *64*, *76*
  USING option *76*
Query tuning
  startup parameters *178*
QUERY-TUNING phrase *41*, *128*

R

r-code
  size *75*
R-code *75*
RAW data type *53*
Read-only (-RO) startup parameter
  Read-only (-RO) startup parameter *172*
RECID function
  joins *123*
  replacing with ROWID *72*
  transactions *64*
Record creation *64*
Record ID *236*
Record locking *66*
RELEASE statement *64*
Remote databases
  starting a DataServer broker *290*
Remote DataServer
  starting *166*
Remote machines
  defined *135*
Repositioning indexes *44*
Required: add at least one index entry *151*
Result set
  defined *82*
Results retrieved from a database
  defining a special view on ORACLE data source to use as a buffer for row results *95*
  loading results into a temp-table *95*
  using OpenEdge-supplied proc-text-buffer from row results *94*
Retrieving output parameter values *97*
Retrieving result sets using proc-text-buffer *98*
Retrieving results with cursor arguments *97*
Return codes *94*, *96*
Return status
  defined *88*
REVERSE-FROM option *128*
-RO startup parameter *172*
ROWID data type *50*
ROWID function
  joins *123*
  record creation *72*
  selecting support *231*
ROWID support
  additional ProDataSet support *117*
  characteristics *113*
  understanding ROWID implementation *113*
  using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO *114*
Run Stored-Procedure statement
  details *87*
RUN STORED-PROCEDURE statement
  complete syntax reference *87*
  description *82*

RUN STORED-PROCEDURE statement *(continued)*
retrieving parameter values *97*
send-sql-statement option
with LOAD-RESULT-INTO option *92*
without LOAD-RESULT-INTO option *92*
syntax *91*
with LOAD-RESULT-INTO phrase *91*
with LOAD-RESULT-INTO phrase syntax *91*, *93*
with send-sql-statement option *92*
without LOAD-RESULT-INTO phrase *90*
without LOAD-RESULT-INTO phrase syntax *90*

S

-S startup parameter
PROSHUT command *291*
SAVE CACHE statement *199*
Schema *24*
Schema Cache File (-cache) startup parameter *177*
Schema caching *24*, *199*
Schema cachings
defined *24*
Schema holders
connecting *171*
creating *149*
defined *23–24*
deleting *234*
deploying *159*
logical names *228*
modifying *228*
updating *221*
verifying *223*
version 7 to 9 *255*
version 8 to 9 *255*
Schema images
defined *24*
Schema objects
defined *36*
ORACLE *36*
Schemas
defined *24*
SELECT . . . FOR UPDATE (SQL) *66*
send-sql-statement option
accessing views *47*
RUN-STORED-PROCEDURE statement *92*
Sequence generators
defined *47*
Sequences *236*
Server Context Log File Naming *192*
Server Join (-nojoinbysqldb) startup parameter *123*, *128*, *178*
Service Name (-S) startup parameter
PROSHUT *291*
SESSION
TIME-SOURCE Connection parameter *182*
SESSION:TIME-SOURCE handle *76*
SETUSERID function *76*
SHARE-LOCK option
FIELDS *76*
Shutdown commands *291*
Single-user Mode (-1) startup parameter *172*
Size limitations
column name *237*
column width *237*
Spaces
leading and trailing *43*
SQL log entry type *187*
.sql script
generating for ORACLE *234*
SQL statements
SELECT *66*
SQL-92
data types *62*
SQL*DBA *150*
SQL*Net *135*, *175*, *275*
sqlnet connection string *175*
Stale cursors
defined *71*
Starting a DataServer broker *290*
Startup parameters
Database Type (-dt) *177*
DataServer (-Dsrv)
qt_cache_size *74*
Index Cursors (-c) *177*
Index Hint (-noindexhint) *41*, *128*, *178*
Internal Code Page (-cpinternal) *39*, *140*, *177*, *239*
Logical Database Name (-ld) *177*
network options
Host Name (-H) *172*
Service Name (-S) *172*
optional *177*
Password (-P) *172*
Physical Database Name (-db) *172*
query tuning *178*
Read-only (-RO) *172*
Schema Cache File (-cache) *177*
Server Join (-nojoinbysqldb) *123*, *128*, *178*
Single-user Mode (-1) *172*
Stream Code Page (-cpstream) *140*, *177*, *239*
User ID (-U) *172*
Statements
COMPILE *75*
CONNECT *171*, *199*
CREATE *64*, *76*
CURRENT-VALUE *76*
DEFINE BROWSE *74*
FIND *41*, *64*, *76*
FIND LAST *75*
FIND PREV *75*
FOR EACH *41*
INDEX-INFORMATION. *41*
OPEN QUERY *64*, *76*

Statements *(continued)*
RUN STORED-PROCEDURE *47*, *86*
SAVE CACHE *199*
UPDATE *66*
VALIDATE *64*
Static temp-table *88*
Stored functions
definition *82*
Stored procedures
and dataserver performance improvement *82*
basics *82*
cursors *97*
data types *82*
defining buffers *99*
definition *82*
implementing and running points *82*
INPUT option *89*
INPUT-OUTPUT option *89*
NLS support *78*
OUTPUT option *89*
PARAM option *89*
returning results *97*
returning status values *89*
Stream Code Page (-cpstream) startup parameter *140*, *177*, *239*
Synonyms
distributed databases *49*
System-owned objects *155*, *217*

T

Table properties
modifying *229*
Table validation
adding to schema holder *229*
Temp-tables
and ProDataSets functionality *104*
loading result sets into *102*
planning a layout for stored procedure result sets *103*
types that support result sets *88*
Terminology *36*
TNS_ADMIN environment variable *259*
Trans log entry type *187*
Transactions *64*, *68*, *72*, *79*
Triggers
adding to schema holder *229*

U

-U startup parameter *172*
ubroker.properties file
hierarchy *284*
Unified Broker framework
and the mergeprop utility *31*
and the OpenEdge Management or OpenEdge Explorer tool *30*
configuring DataServer applications using *30*
Unique log file creation *193*
UNIX environment variable *277*
Unknown value (?) *62*
UPDATE statement *66*
Updating schema holders *221*
Upgrading to the DataServer *33*
USE-INDEX option *41*, *72*
User ID (-U) startup parameter *172*
User IDs *76*, *150*, *172*, *200*
USERID function *76*, *175*
USING option
joins *123*

V

VALIDATE statement *64*
Validation
expression *230*
Values of output parameters you define when creating a procedure *94*
Verifying schema holders *223*
Views
defined *45*
multi-table *46*

W

WHERE clause
literals *128*
ORACLE time option *76*
Word indexing
CONTAINS *76*

Z

-znotrim startup parameter *43*