# Contents

1. **Preface**
   - Oracle DataServer overview
   - DataServer components
   - Security
   - DataServer utilities
   - DataServer demonstration database
   - DataServer configurations
   - Distributed DataServer applications
   - Configuring distributed DataServer applications using the Unified Broker Framework
   - A closer look at the OpenEdge Explorer and Progress Explorer tool
   - Introducing the mergeprop utility
   - Software requirements
   - Guidelines for using the DataServer
   - Documentation resources
   - Guide to related topics

2. **Initial Programming Considerations**
   - Database design issues
     - Oracle and OpenEdge objects and terminology
     - Naming conventions
     - Code pages
     - Support for Unicode
     - Indexes
     - Case-insensitive index components
     - Index repositioning
     - Initial values
     - Oracle views
     - Database triggers
     - Sequence generator
     - Oracle synonyms
## Contents

Data types .......................... 2–16
- Oracle CHAR and VARCHAR2 data types .......... 2–18
- Oracle NUMBER data type ............... 2–18
- Oracle Number data .................. 2–18
- Logical data type and Oracle equivalents .......... 2–18
- ABL supports RAW data type ............. 2–19
- Oracle BLOB and BFILE data types .......... 2–19
- Oracle CLOB and CFILE data types .......... 2–19
- Oracle DATE data type ............... 2–19
- Working with Datetime data types .......... 2–20
- Unsupported data types ............. 2–27

Arrays .......................... 2–28
- Unknown Value (?) .................. 2–29
- Zero-length character strings .......... 2–30
- Record creation .................. 2–31
- Record locking .................. 2–34
- Monitoring locks .................. 2–35
- Transactions .................. 2–36
- Error handling .................. 2–37
- Overflow checking .................. 2–38
- Cursors .......................... 2–39
- Static FIND cursor repositioning .......... 2–39
- Stale cursors .................. 2–39
- Allocating cursors ............. 2–40

ABL issues .......................... 2–41
- RECID function .................. 2–41
- ROWID function .................. 2–41
- Progress_Recid change .................. 2–42
- DEFINE BROWSE statement .......... 2–43
- Field lists .................. 2–43
- FIND PREV/LAST statements .......... 2–44
- Compiling ABL procedures .......... 2–45
- Unsupported ABL statements .......... 2–45
- National Language Support .......... 2–48

RDBMS Stored Procedures .......................... 2–49

3. RDBMS Stored Procedure Details .......................... 3–1
- Overview .................. 3–2
- Defining native stored procedures to ABL .......... 3–2
- RDBMS stored procedure basics .......... 3–3
- RUN STORED-PROCEDURE details .......... 3–7
  - Syntax reference for RUN STORED-PROCEDURE .......... 3–7
  - RUN STORED-PROCEDURE statement .......... 3–8
  - CLOSED STORED-PROCEDURE statement .......... 3–10
  - RUN STORED-PROC statement execution without the LOAD-RESULT-INTO phrase .......... 3–11
  - RUN STORED-PROC statement execution with the LOAD-RESULT-INTO phrase .......... 3–12
  - RUN STORED-PROC statement with send-sql-statement option .......... 3–12
- Data output and retrieval options .......... 3–14
- Return codes .................. 3–14
- Values of output parameters you define when you create a procedure .......... 3–14
- Results retrieved from a database .......... 3–14
## Contents

Interfacing with RDBMS stored procedures .................................................. 3–16
- Retrieving return codes ............................................................................. 3–17
- Retrieving output parameter values .......................................................... 3–17
- Retrieving results with cursor arguments ............................................... 3–18
- Retrieving results sets using proc-text-buffer ...................................... 3–19
- Defining a view to use as a buffer .............................................................. 3–20
- Loading a result set into a temp-table ....................................................... 3–23

Handling errors .............................................................................................. 3–32
- Error messages related to using the LOAD-RESULT-INTO phrase .......... 3–32

ROWID support ................................................................................................. 3–34
- Understanding the ROWID Implementation ............................................. 3–34
- ROWID characteristics .............................................................................. 3–34
- Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO 3–35
- Additional ProDataSet support ................................................................. 3–39

4. Additional Features to Enhance DataServer Performance ..................... 4–1
- Enhancements overview .............................................................................. 4–2
- SQL and ABL queries ............................................................................... 4–3
- Bind variables ........................................................................................... 4–4
- Query tuning .............................................................................................. 4–5
- Caching records ........................................................................................ 4–10
- Oracle hints .............................................................................................. 4–11
- Using field lists when updating records ................................................. 4–12
- Join by SQL DB ......................................................................................... 4–13
- Skipping schema verification .................................................................... 4–15
- Writing queries for performance ............................................................. 4–16

5. Configuring the DataServer ................................................................. 5–1
- DataServer modules .................................................................................. 5–2
- Configuring Windows modules ............................................................... 5–4
  - Configuring the local DataServer in Windows ..................................... 5–4
  - Configuring the DataServer in the Windows host ............................... 5–5
- Configuring UNIX modules .................................................................... 5–6
  - Configuring the local DataServer on UNIX ....................................... 5–6
  - Configuring the DataServer on the UNIX host .................................. 5–7
- Configuring an international environment ............................................. 5–8
- Configuring the DataServer in the Unified Broker administration framework . 5–10
  - Configuring the Explorer administration framework in Windows .... 5–10
  - Configuring the Progress Explorer administration framework on UNIX 5–11
- Creating, Maintaining, and Deploying a schema holder ......................... 5–15
  - Schema holder security ....................................................................... 5–15
  - Running Oracle ...................................................................................... 5–17
  - Starting the DataServer processes ...................................................... 5–17
  - Creating an empty OpenEdge database ............................................. 5–18
  - Using the DataServer utility to create a schema image .................... 5–19
  - Maintaining a schema holder ............................................................... 5–22
  - Deploying a schema holder ................................................................. 5–22
  - Setting up a schema holder as a Windows service ............................... 5–23
## 6. Connecting the DataServer

- Starting the local DataServer ........................................ 6–2
- Starting the remote DataServer ...................................... 6–3
  - Starting the DataServer in the Explorer administration framework .... 6–3
  - Starting and stopping the broker process via the command line ........ 6–6
  - Starting the UNIX client process .................................. 6–7
  - Starting the Windows client process ............................... 6–7
- Connecting a schema holder ........................................... 6–8
  - Connecting a schema holder at startup .......................... 6–9
  - Connecting through Oracle Networking ........................... 6–12
  - Optional connection and startup parameters ..................... 6–13
  - Query tuning with connection and startup parameters .......... 6–14
  - Analyzing performance ............................................. 6–17
  - Index cursors ......................................................... 6–18
  - Local schema caching ................................................ 6–19
- Connection failures and OpenEdge responses ....................... 6–21
  - Connection troubleshooting ......................................... 6–22
  - Accessing the DataServer log ...................................... 6–22

## 7. The DataServer Tutorial ............................................. 7–1

- Oracle Utilities menu .................................................. 7–2
- Creating a schema image ............................................. 7–3
- Updating a schema image ............................................. 7–7
- Verifying a schema image ............................................ 7–9
- Changing logical name and connection information ............... 7–13
- Modifying a schema image ........................................... 7–14
  - Modifying table-level information ................................ 7–14
  - Modifying field-level information ................................ 7–15
  - Defining the ROWID .................................................. 7–17
- Changing a code page .................................................. 7–19
- Deleting a schema image .............................................. 7–20
- Migrating an OpenEdge Database to Oracle with PRO/SQL .......... 7–21
- OpenEdge DB-to-Oracle utility ....................................... 7–22
  - Preparing a database for the utility ............................. 7–23
  - Oracle size limitations ............................................. 7–23
  - Naming conventions .................................................. 7–25
  - Oracle tablespaces for tables and indexes ....................... 7–25
  - Running the OpenEdge DB-to-Oracle utility ..................... 7–26
  - Handling character length during database migration ........... 7–32
- OpenEdge DB-to-Oracle Incremental Schema Migration utility ...... 7–34
  - Adjusting field widths during migration ......................... 7–39
  - Updating the Oracle database ..................................... 7–39
- Adjust an Oracle schema image to an OpenEdge database .......... 7–40

## A. Upgrading DataServer Applications ................................ A–1

- Upgrading schema holders from Progress Version 9 ................ A–2
- Manually adding PROGRESS_RECID to Oracle ....................... A–3

## B. Environment Variables ............................................. B–1
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1–1:</td>
<td>Supported configurations</td>
<td>1–10</td>
</tr>
<tr>
<td>Table 1–2:</td>
<td>How to use this manual</td>
<td>1–17</td>
</tr>
<tr>
<td>Table 1–3:</td>
<td>DataServer-related topics</td>
<td>1–18</td>
</tr>
<tr>
<td>Table 2–1:</td>
<td>Oracle and OpenEdge objects</td>
<td>2–2</td>
</tr>
<tr>
<td>Table 2–2:</td>
<td>Supported Oracle views</td>
<td>2–11</td>
</tr>
<tr>
<td>Table 2–3:</td>
<td>OpenEdge and Oracle sequence generators</td>
<td>2–13</td>
</tr>
<tr>
<td>Table 2–4:</td>
<td>Supported synonyms</td>
<td>2–15</td>
</tr>
<tr>
<td>Table 2–5:</td>
<td>Oracle and OpenEdge data types</td>
<td>2–16</td>
</tr>
<tr>
<td>Table 2–6:</td>
<td>LOGICAL data type and Oracle equivalents</td>
<td>2–18</td>
</tr>
<tr>
<td>Table 2–7:</td>
<td>ABL datatime data types</td>
<td>2–20</td>
</tr>
<tr>
<td>Table 2–8:</td>
<td>Oracle datatime data types</td>
<td>2–20</td>
</tr>
<tr>
<td>Table 2–9:</td>
<td>ABL DATETIME data type conversions</td>
<td>2–21</td>
</tr>
<tr>
<td>Table 2–10:</td>
<td>Converting ABL DATETIME types to ORACLE TIMESTAMP types</td>
<td>2–23</td>
</tr>
<tr>
<td>Table 2–11:</td>
<td>Converting Oracle TIMESTAMP types to ABL DATETIME types</td>
<td>2–25</td>
</tr>
<tr>
<td>Table 2–12:</td>
<td>Unknown value (?) and NULL values</td>
<td>2–29</td>
</tr>
<tr>
<td>Table 2–13:</td>
<td>ABL and Oracle locks</td>
<td>2–34</td>
</tr>
<tr>
<td>Table 2–14:</td>
<td>Oracle locking</td>
<td>2–35</td>
</tr>
<tr>
<td>Table 2–15:</td>
<td>ABL and Oracle differences</td>
<td>2–46</td>
</tr>
<tr>
<td>Table 3–1:</td>
<td>Stored procedure language elements</td>
<td>3–3</td>
</tr>
<tr>
<td>Table 3–2:</td>
<td>Argument data types for stored procedures</td>
<td>3–4</td>
</tr>
<tr>
<td>Table 3–3:</td>
<td>Returning result sets and loading the data into temp-tables</td>
<td>3–25</td>
</tr>
<tr>
<td>Table 3–4:</td>
<td>Options to plan the temp-table layout for result sets</td>
<td>3–26</td>
</tr>
<tr>
<td>Table 4–1:</td>
<td>Query-tuning options</td>
<td>4–6</td>
</tr>
<tr>
<td>Table 4–2:</td>
<td>Controlling join by SQLDB</td>
<td>4–14</td>
</tr>
<tr>
<td>Table 5–1:</td>
<td>DataServer Configurations</td>
<td>5–2</td>
</tr>
<tr>
<td>Table 5–2:</td>
<td>DataServer for Oracle environment variables</td>
<td>5–5</td>
</tr>
<tr>
<td>Table 5–3:</td>
<td>Environment variables for the local DataServer</td>
<td>5–6</td>
</tr>
<tr>
<td>Table 5–4:</td>
<td>Environment variables for the remote DataServer</td>
<td>5–7</td>
</tr>
<tr>
<td>Table 5–5:</td>
<td>OpenEdge DataServer for Oracle sections</td>
<td>5–12</td>
</tr>
<tr>
<td>Table 5–6:</td>
<td>Required Oracle permissions</td>
<td>5–16</td>
</tr>
<tr>
<td>Table 5–7:</td>
<td>Database operations and the schema holder</td>
<td>5–22</td>
</tr>
<tr>
<td>Table 6–1:</td>
<td>Environment variables for the remote DataServer</td>
<td>6–6</td>
</tr>
<tr>
<td>Table 6–2:</td>
<td>Remote DataServer connection parameters</td>
<td>6–9</td>
</tr>
<tr>
<td>Table 6–3:</td>
<td>Required connection parameters</td>
<td>6–10</td>
</tr>
<tr>
<td>Table 6–4:</td>
<td>Optional connection and startup parameters</td>
<td>6–13</td>
</tr>
<tr>
<td>Table 6–5:</td>
<td>Connection query-tuning options</td>
<td>6–15</td>
</tr>
<tr>
<td>Table 6–6:</td>
<td>Query-tuning startup parameters</td>
<td>6–16</td>
</tr>
<tr>
<td>Table 6–7:</td>
<td>Diagnostic options</td>
<td>6–17</td>
</tr>
<tr>
<td>Table 6–8:</td>
<td>Failure responses</td>
<td>6–21</td>
</tr>
<tr>
<td>Table 7–1:</td>
<td>OpenEdge DataServer for Oracle Utilities menu</td>
<td>7–2</td>
</tr>
<tr>
<td>Table 7–2:</td>
<td>Verify utility report</td>
<td>7–9</td>
</tr>
<tr>
<td>Table 7–3:</td>
<td>Oracle permissions</td>
<td>7–22</td>
</tr>
<tr>
<td>Table 7–4:</td>
<td>An OpenEdge DB-to-Oracle database conversion</td>
<td>7–23</td>
</tr>
<tr>
<td>Table 7–5:</td>
<td>Columns required by OpenEdge objects</td>
<td>7–23</td>
</tr>
<tr>
<td>Table 7–6:</td>
<td>OpenEdge-to-Oracle naming conventions</td>
<td>7–25</td>
</tr>
<tr>
<td>Table 7–7:</td>
<td>OpenEdge DB to Oracle UI settings</td>
<td>7–27</td>
</tr>
<tr>
<td>Table 7–8:</td>
<td>Environment Variables</td>
<td>7–31</td>
</tr>
<tr>
<td>Table 7–9:</td>
<td>Recommended character length settings</td>
<td>7–33</td>
</tr>
<tr>
<td>Table 7–10:</td>
<td>Delta df to Oracle Conversion UI elements</td>
<td>7–35</td>
</tr>
<tr>
<td>Table 7–11:</td>
<td>Oracle equivalents of OpenEdge objects</td>
<td>7–37</td>
</tr>
<tr>
<td>Table 7–12:</td>
<td>Database modifications not converted to Oracle</td>
<td>7–38</td>
</tr>
<tr>
<td>Table 7–13:</td>
<td>Sample object equivalents</td>
<td>7–38</td>
</tr>
<tr>
<td>Table B–1:</td>
<td>Environment variables</td>
<td>B–2</td>
</tr>
<tr>
<td>Table D–1:</td>
<td>Building DataServer processes</td>
<td>D–3</td>
</tr>
</tbody>
</table>
Figures

Figure 1–1: DataServer architecture ....................................................... 1–4
Figure 1–2: The OpenEdge DataServer for Oracle logic .......................... 1–5
Figure 1–3: The schema-loading process ............................................... 1–6
Figure 1–4: The local DataServer ......................................................... 1–9
Figure 1–5: A local DataServer accessing remote Oracle database through
Oracle networking .............................................................................. 1–9
Figure 1–6: The remote DataServer for Oracle ...................................... 1–10
Figure 2–1: DataServer processes and code pages ................................. 2–4
Examples

Example 3–1: Passing a SELECT statement as a parameter .......................... 3–13
Example 3–2: Using the send-sql-statement option with LOAD-RESULT-INTO option .......................... 3–13
Example 3–3: Oracle stored procedure pcust .................................................. 3–16
Example 3–4: ABL code running the stored procedure pcust .......................... 3–17
Example 3–5: Passing parameters by name ..................................................... 3–17
Example 3–6: Returning database results into the proc-text-buffer and converting the results to the INTEGER data type ................................. 3–19
Example 3–7: Running the send-sql-statement option twice ......................... 3–21
Example 3–8: A stored procedure returning multiple result sets and how to access the result sets using cursor arguments and buffers ..................... 3–22
Example 3–9: Using the RUN STORED-PROC statement with a LOAD-RESULT-INTO phrase ................................................................. 3–24
Example 3–10: Dynamic temp-tables shown in a prepared state .................. 3–28
Example 3–11: Executing a stored procedure call using the LOAD-RESULT-INTO phrase ................................................................. 3–28
Example 3–12: Using an existing temp-table without the TEMP-TABLE-PREPARE ( ) method ................................................................. 3–29
Example 3–13: Using an existing temp-table with the TEMP-TABLE-PREPARE ( ) method ................................................................. 3–29
Example 3–14: A stored procedure that uses the LOAD-RESULT-INTO phrase with a single temp table and the send-sql-statement option ............... 3–30
Example 3–15: Using the PROC-STATUS phrase ........................................... 3–30
Example 3–16: Executing a call to a stored function that returns a result set using the LOAD-RESULT-INTO phrase ........................................ 3–31
Example 3–17: How to trap errors within a stored procedure ......................... 3–32
Example 3–18: Trapping errors when using LOAD-RESULT-INTO .................. 3–32
Example 3–19: Expected ABL behavior—ROWID value of a Temp-Table buffer .... 3–35
Example 3–20: Simple ProDataSet code ......................................................... 3–36
Example 3–21: Using the LOAD-RESULT-INTO technique to populate the underlying temp-table of a ProDataSet ................................. 3–37
Example 3–22: Using the LOAD-RESULT-INTO technique with BEFORE-FILL method to fill the temp-table(s) of a ProDataSet ............................. 3–39
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
- Third party acknowledgements
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.

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

Chapter 1, “Introduction”

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

Chapter 2, “Initial Programming Considerations”

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

Chapter 3, “RDBMS Stored Procedure Details”

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.

Chapter 4, “Additional Features to Enhance DataServer Performance”

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

Chapter 5, “Configuring the DataServer”

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

Chapter 6, “Connecting the DataServer”

Presents instructions for connecting the DataServer and a schema holder.

Chapter 7, “The DataServer Tutorial”

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.
Appendix A, “Upgrading DataServer Applications”

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

Appendix B, “Environment Variables”

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

Appendix C, “Sample Queries”

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

Appendix D, “Building DataServer Executables”

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

Appendix E, “DataServer Command Line Utilities and Startup Parameters”

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

Using this manual

Chapter 1, “Introduction,” 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 PSDN: http://communities.progress.com/pcom/docs/DOC-16074.

Before you begin

Before you attempt to use the Oracle DataServer, read OpenEdge Release Notes, OpenEdge Getting Started: Installation and Configuration, and Chapter 2, “Initial Programming Considerations.” Also, be sure to follow the step-by-step instructions in Chapter 5, “Configuring the DataServer,” and Chapter 6, “Connecting the DataServer,” for installing, configuring, and connecting to Oracle.

See the “Documentation resources” section on page 1–17 for additional information.
Getting started with ABL (Advanced Business Language)

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.

References to ABL compiler and run-time features

ABL is both a compiled and interpreted language that executes in a run-time engine that the documentation refers to as the ABL Virtual Machine (AVM). When 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 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 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 defined for a given application example.
# Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td><strong>SMALL, BOLD CAPITAL LETTERS</strong></td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, <strong>GET</strong> and <strong>CTRL</strong>.</td>
</tr>
<tr>
<td><strong>KEY1+KEY2</strong></td>
<td>A plus sign between key names indicates a <em>simultaneous</em> key sequence: you press and hold down the first key while pressing the second key. For example, <strong>CTRL+X</strong>.</td>
</tr>
<tr>
<td><strong>KEY1 KEY2</strong></td>
<td>A space between key names indicates a <em>sequential</em> key sequence: you press and release the first key, then press another key. For example, <strong>ESCAPE H</strong>.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Fixed width</strong></td>
<td>A fixed-width font is used in syntax statements, code examples, system output, and filenames.</td>
</tr>
<tr>
<td><strong>Fixed-width italics</strong></td>
<td>Fixed-width italics indicate variables in syntax statements.</td>
</tr>
<tr>
<td><strong>Fixed-width bold</strong></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td><strong>UPPERCASE fixed width</strong></td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
<tr>
<td>[ ]</td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td>[ ]</td>
<td>This icon (one arrow) introduces a single-step procedure.</td>
</tr>
<tr>
<td><strong>Period (.) or colon (:)</strong></td>
<td>All statements except <strong>DO</strong>, <strong>FOR</strong>, <strong>FUNCTION</strong>, <strong>PROCEDURE</strong>, and <strong>REPEAT</strong> end with a period. <strong>DO</strong>, <strong>FOR</strong>, <strong>FUNCTION</strong>, <strong>PROCEDURE</strong>, and <strong>REPEAT</strong> statements can end with either a period or a colon.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Small brackets are part of the ABL.</td>
</tr>
<tr>
<td>{ }</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>{ }</td>
<td>Small braces are part of the ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
<tr>
<td></td>
<td>A vertical bar indicates a choice.</td>
</tr>
</tbody>
</table>
### 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:
    DISPLAY 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 OpenEdge 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/dlc/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**.

Third party acknowledgements

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc. [http://www.adventnet.com](http://www.adventnet.com). All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation [http://www.apache.org/](http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without
prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. - Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and
Copyright © 1991 by Lucent Technologies (rnd_prod.s). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (deccstrtod.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (deccstrtof.c; strtord.c); Copyright © 1998 by Lucent Technologies All Rights Reserved (dmisc.c; gdtoa.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

OpenEdge includes http package software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.


OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.
OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names “OpenSSL Toolkit” and “OpenSSL Project” must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called “OpenSSL” nor may “OpenSSL” appear in their names without prior written permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.
OpenEdge includes Sonic software, which includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and
distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery Processor from Saxonica Limited (http://www.saxonica.com/) which are available from SourceForge (http://sourceforge.net/projects/saxon/). The Original Code of Saxon comprises all those components which are not explicitly attributed to other parties. The Initial Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an employee of International Computers Limited (now part of Fujitsu Limited), and original code developed during that time was released under this license by permission from International Computers Limited. From February 2001 until February 2004 Michael Kay was an employee of Software AG, and code developed during that time was released under this license by permission from Software AG, acting as a “Contributor”. Subsequent code has been developed by Saxonica Limited, of which Michael Kay is a Director, again acting as a “Contributor”. A small number of modules, or enhancements to modules, have been developed by other individuals (either written especially for Saxon, or incorporated into Saxon having initially been released as part of another open source product). Such contributions are acknowledged individually in comments attached to the relevant code modules. All Rights Reserved. The contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/ and a copy of the license can also be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

OpenEdge includes XML Tools, which includes Xs3P v1.1.3. The contents of this file are subject to the DSTC Public License (DPL) Version 1.1 (the “License”); you may not use this file except in compliance with the License. A copy of the license can be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR “AS IS” AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
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. This chapter includes the following topics:

- Oracle DataServer overview
- DataServer components
- Configuring distributed DataServer applications using the Unified Broker Framework
- 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 the “DataServer configurations” section on page 1–8 for descriptions of your configuration options.
Figure 1–1 illustrates the DataServer modules as they appear in a networked, or remote, configuration.

**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.
Figure 1–2 shows the internal logic of the OpenEdge DataServer for Oracle.

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.

Figure 1–3 illustrates the schema-loading process.

---

**Figure 1–3: 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 *OpenEdge Deployment: Managing ABL Applications*. For general information about OpenEdge security, see *OpenEdge Getting Started: Core Business Services*.

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 the “Schema holder security” section on page 5–15 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 Chapter 7, “The DataServer Tutorial,” 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 Chapter 7, “The DataServer Tutorial,” 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).
Figure 1–4 shows a local DataServer configuration where all the modules run on one machine. In this case, the Oracle database is also local.

![Diagram of local DataServer configuration](image)

**Figure 1–4: The local DataServer**

Figure 1–5 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.

![Diagram of DataServer accessing remote Oracle database through Oracle networking](image)

**Figure 1–5: A local DataServer accessing remote Oracle database through Oracle networking**
Figure 1–6 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.

![Diagram of the remote DataServer for Oracle]

**Figure 1–6: 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. Table 1–1 lists all supported configurations. It considers possible client/server combinations and networking options.

**Table 1–1: Supported configurations**

<table>
<thead>
<tr>
<th>Client</th>
<th>Networking</th>
<th>DataServer</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenEdge on UNIX</td>
<td>None</td>
<td>DataServer, client, and Oracle on the same machine</td>
</tr>
<tr>
<td>OpenEdge on UNIX</td>
<td>Oracle</td>
<td>DataServer, client, and Oracle Networking on the client machine, and Oracle on a server</td>
</tr>
<tr>
<td>OpenEdge on UNIX</td>
<td>OpenEdge</td>
<td>Client on the client machine, and the DataServer and Oracle on a server</td>
</tr>
<tr>
<td>OpenEdge in Windows</td>
<td>Oracle</td>
<td>DataServer, client, and Oracle Networking on the client machine, and Oracle on a server</td>
</tr>
<tr>
<td>OpenEdge in Windows</td>
<td>OpenEdge</td>
<td>Client on the client machine, and the DataServer and Oracle on a server</td>
</tr>
</tbody>
</table>
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 Progress Explorer administration 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 Progress Explorer administration framework.

In the Progress Explorer administration 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 Progress Explorer, which provides a graphical interface to configuration and administrative tasks. On UNIX, you work with the Explorer administration framework through a properties file and command-line utilities. See the “Configuring distributed DataServer applications using the Unified Broker Framework” section on page 1–12 and OpenEdge Getting Started: Installation and Configuration for an overview of the Unified Broker framework.
Configuring distributed DataServer applications 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 Explorer and Progress Explorer configuration tool, mergeprop utility, and command-line utilities.

The framework also facilitates 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 OpenEdge Getting Started: Installation and Configuration.

A closer look at the OpenEdge Explorer and Progress Explorer tool

The Progress Explorer tool is a graphical user interface that provides an easy way for you to manage OpenEdge servers. The Progress 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 the Progress 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 the OpenEdge Explorer and Progress Explorer tool, see the OpenEdge Explorer and Progress Explorer online help.
Introducing the mergeprop utility

The `mergeprop` utility is a command-line utility that supports functionality similar to that supported by the OpenEdge Explorer and Progress 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:

- The “Configuring the DataServer in the Unified Broker administration framework” section on page 5–10.
- The “Starting the DataServer in the Explorer administration framework” section on page 6–3 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 Chapter 2, “Initial Programming Considerations.”

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.

See Chapter 5, “Configuring the DataServer,” 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.

See the “OpenEdge DB-to-Oracle utility” section on page 7–22 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.

See Chapter 5, “Configuring the DataServer,” for information about where to install DataServer modules and creating a schema holder.
Table 1–2 suggests paths through this manual that accommodate different approaches to using the OpenEdge DataServer for Oracle.

**Table 1–2: How to use this manual**

<table>
<thead>
<tr>
<th>If you are . . .</th>
<th>Read the contents of . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>New to the OpenEdge DataServer for Oracle</td>
<td>Chapter 2, “Initial Programming Considerations”</td>
</tr>
<tr>
<td></td>
<td>Chapter 5, “Configuring the DataServer”</td>
</tr>
<tr>
<td></td>
<td>Chapter 6, “Connecting the DataServer”</td>
</tr>
<tr>
<td></td>
<td>Chapter 7, “The DataServer Tutorial”</td>
</tr>
<tr>
<td>Migrating an OpenEdge database to Oracle</td>
<td>The “OpenEdge DB-to-Oracle utility” section on page 7–22</td>
</tr>
<tr>
<td></td>
<td>Chapter 2, “Initial Programming Considerations”</td>
</tr>
<tr>
<td></td>
<td>Chapter 5, “Configuring the DataServer”</td>
</tr>
<tr>
<td></td>
<td>Chapter 6, “Connecting the DataServer”</td>
</tr>
<tr>
<td></td>
<td>Chapter 7, “The DataServer Tutorial”</td>
</tr>
<tr>
<td>Upgrading the DataServer to OpenEdge Release 10</td>
<td>Chapter 2, “Initial Programming Considerations”</td>
</tr>
<tr>
<td></td>
<td>Chapter 5, “Configuring the DataServer”</td>
</tr>
<tr>
<td></td>
<td>Chapter 7, “The DataServer Tutorial”</td>
</tr>
</tbody>
</table>

**Guide to related topics**

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

- *OpenEdge Getting Started: Database Essentials* explains how to design an OpenEdge database and provides good relational database design guidelines.

- *OpenEdge Getting Started: Progress OpenEdge Studio* and *OpenEdge Getting Started: ABL Essentials* provide information on how to develop applications in ABL.
Table 1–3 lists topics related to building and using the DataServer and indicates where to find the information.

**Table 1–3: DataServer-related topics**

<table>
<thead>
<tr>
<th>Topic</th>
<th>OpenEdge manuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting up OpenEdge</td>
<td><em>OpenEdge Getting Started: Installation and Configuration</em></td>
</tr>
<tr>
<td></td>
<td><em>OpenEdge Data Management: Database Administration</em></td>
</tr>
<tr>
<td>Using the Data Dictionary</td>
<td><em>OpenEdge Getting Started: Database Essentials</em></td>
</tr>
<tr>
<td></td>
<td><em>OpenEdge Data Management: Database Administration</em></td>
</tr>
<tr>
<td>Defining security for an OpenEdge database</td>
<td><em>OpenEdge Data Management: Database Administration</em></td>
</tr>
<tr>
<td>Writing applications in ABL</td>
<td><em>OpenEdge Getting Started: Progress</em></td>
</tr>
<tr>
<td></td>
<td><em>OpenEdge Studio</em></td>
</tr>
<tr>
<td></td>
<td><em>OpenEdge Getting Started: ABL Essentials</em></td>
</tr>
</tbody>
</table>
Initial Programming Considerations

OpenEdge Studio 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 have to consider when you plan your applications and design 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, as detailed in the following sections:

- Database design issues
- Data types
- Arrays
- Unknown Value (?)
- Record creation
- Record locking
- Transactions
- Error handling
- Cursors
- ABL issues
- 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 Table 2–1.

Table 2–1: Oracle and OpenEdge objects

<table>
<thead>
<tr>
<th>Oracle schema object</th>
<th>OpenEdge equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Row</td>
<td>Record</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Sequences</td>
<td>Sequences</td>
</tr>
<tr>
<td>Integrity constraint¹</td>
<td>Validation expression</td>
</tr>
<tr>
<td>No equivalent</td>
<td>Validation message</td>
</tr>
<tr>
<td>Trigger</td>
<td>Trigger</td>
</tr>
<tr>
<td>Stored procedures</td>
<td>Stored procedures (SQL Engine)</td>
</tr>
<tr>
<td>View</td>
<td>View (SQL Engine)</td>
</tr>
<tr>
<td>Clusters</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Oracle Object methods</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Synonyms (SQL Engine)</td>
</tr>
</tbody>
</table>

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.
Figure 2–1 shows a possible configuration of code pages for the DataServer components and processes.

![Diagram of code pages and processes]

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 the “Configuring an international environment” section on page 5–8 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 *OpenEdge Development: Internationalizing 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 NCL0B. 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 NCL0B, 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 `$DLC/prolang/utf/empty`. If you use the Create Server Schema ORACLE DataServer utility to create a DataServer schema and you plan to set the schema holder’s code page to `UTF-8`, the schema holder must also be derived from a copy of the empty `UTF-8` database located at `$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/prolong/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 might also consider several other available client code page settings for Unicode. For more information, see *OpenEdge Development: Internationalizing Applications*. 


Initial Programming Considerations

- **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 *OpenEdge Development: Internationalizing 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 the “OpenEdge DB-to-Oracle utility” section on page 7–22. Specific considerations for handling character length are discussed in the “Handling character length during database migration” section on page 7–32.

**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, 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:
```

**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.

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 the “Query tuning” section on page 4–5 and the “Oracle hints” section on page 4–11 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-insensitive index components

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.

### 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.
```

```
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 \texttt{-Dsrv \textit{srv}-get-prev} switch. This indicates to the DataServer, that it should generate new SQL to satisfy a \texttt{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 \textit{view} is a presentation of data in one or more tables. Table 2–2 lists the level of support available for various types of views.

### Table 2–2: Supported Oracle views

<table>
<thead>
<tr>
<th>View</th>
<th>Supported ABL</th>
</tr>
</thead>
</table>
| All columns from a single table:  
\texttt{SELECT custnum, name, postalcode ...  
FROM customer} | Full support for this view. Use the \texttt{USE-INDEX} option with \texttt{FIND NEXT} statements to get the same results you would expect from ABL. |
| Some columns from a single table:  
\texttt{SELECT custnum, name FROM customer} | Full support for this view. Use the \texttt{USE-INDEX} option with \texttt{FIND NEXT} statements to get the same results you would expect from ABL. |
| All columns from a single table with an expression:  
\texttt{SELECT custnum, \texttt{UPPER(name)}, postalcode...  
WHERE postalcode < 20000  
FROM customer} | Full support for this view, except that you cannot update the column with the expression (\texttt{name}, in this example). |
| All columns from a single table with an aggregate or function:  
\texttt{SELECT custnum, name, postalcode ...  
WHERE postalcode < 20000  
FROM customer  
GROUP BY postalcode} | The only supported statements are:  
\texttt{FOR EACH ... NO-LOCK|SHARE-LOCK:  
FIND ... NO-LOCK|SHARE-LOCK:} |
| Columns from multiple tables:  
\texttt{SELECT custnum, name, postalcode, ordernum  
FROM customer, order  
WHERE salesrep = 'HXM'} | The only supported statements are:  
\texttt{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 OpenEdge Getting Started: ABL Essentials 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:

```sql
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:

**Syntax**

```
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 Chapter 2, “Initial Programming Considerations,” 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:

```sql
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

Table 2–3 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 OpenEdge Getting Started: ABL Essentials for information on defining and using OpenEdge sequences in your database applications.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>OpenEdge</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum/minimum range</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ascending/descending</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Adjustable incrementation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Database object</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Loses a number when a record creation fails</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sequence is disrupted during dumping or loading</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
If you add a sequence to a table in your supporting Oracle database, you must update the schema image to reflect this addition. See Chapter 7, “The DataServer Tutorial,” 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-bit native NUMBER data type. For details about NUMBER data type support, see the “Data types” section on page 2–16.

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 \texttt{NY\_CUSTOMER} with a synonym \texttt{CUSTOMER}. Applications can refer to the table as \texttt{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.

\textbf{Note:} If you use a synonym for a table that has a \texttt{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 \texttt{customer} table includes \texttt{customer\_SEQ} and you define a synonym, \texttt{CUST}, you must also define and include in the schema image a synonym for the sequence generator, such as \texttt{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. \textbf{Table 2–4} describes the support for synonyms in local and distributed (remote) databases.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{Synonym} & \textbf{Points to . . .} & \textbf{Supported} \\
\hline
\texttt{CUSTOMER} in local DB & \texttt{NY\_CUSTOMER} in local DB & Yes \\
\hline
\texttt{CUSTOMER} in local DB & \texttt{NY\_CUSTOMER} in remote DB I & Yes \\
\hline
\texttt{CUSTOMER} in remote DB I & \texttt{NY\_CUSTOMER} in remote DB I & Yes \\
\hline
\texttt{CUSTOMER} in remote DB I & \texttt{NY\_CUSTOMER} in remote DB II & No \\
\hline
\end{tabular}
\end{table}
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 the “Modifying a schema image” section on page 7–14 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 Table 2–5.

Table 2–5 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 Table 2–5 provide additional details about several of the Oracle data types and each data type’s OpenEdge equivalent.

Table 2–5: Oracle and OpenEdge data types

<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>OpenEdge equivalent data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CFILE</td>
<td>CLOB</td>
</tr>
<tr>
<td>CLOB</td>
<td>CLOB</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>NCHAR</td>
<td>^1</td>
</tr>
<tr>
<td>NVARCHAR2</td>
<td>^1</td>
</tr>
<tr>
<td>NCLOB</td>
<td>^1</td>
</tr>
</tbody>
</table>

^1 Indicates that the table is part of a series, with more information provided in subsequent pages.
## Table 2–5: Oracle and OpenEdge data types (2 of 2)

<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>OpenEdge equivalent data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>DECIMAL</td>
</tr>
<tr>
<td></td>
<td>INTEGER</td>
</tr>
<tr>
<td></td>
<td>INT64 (^2)</td>
</tr>
<tr>
<td></td>
<td>LOGICAL</td>
</tr>
<tr>
<td>NUMBER (10)(^3)</td>
<td>INT64</td>
</tr>
<tr>
<td>NUMBER (11)(^3)</td>
<td>INT64</td>
</tr>
<tr>
<td>NUMBER (7,2)(^4)</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>NUMBER (13,2)(^5)</td>
<td></td>
</tr>
<tr>
<td>NUMBER (20(^\ast))(^6)</td>
<td></td>
</tr>
<tr>
<td>CURSOR</td>
<td>INTEGER</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>INTEGER</td>
</tr>
<tr>
<td>LONG</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>RAW (limited support)</td>
<td>RAW</td>
</tr>
<tr>
<td>LONG RAW</td>
<td>RAW</td>
</tr>
<tr>
<td>BLOB</td>
<td>BLOB</td>
</tr>
<tr>
<td>BFILE</td>
<td>BLOB</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>DATETIME (CHAR)(^7)</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>LONG CHARACTER</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>UNDEFINED</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>ROWID</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>LOGICAL</td>
</tr>
</tbody>
</table>

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

**Note:** The only Oracle internal data type that the DataServer does not support is \texttt{ROWID}. The Oracle \texttt{ROWID}, however, has its programming equivalent in the OpenEdge \texttt{ROWID}. See the \texttt{"ROWID function"} section on page 2–41 for more information.
**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 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.

Table 2–5 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 the “Modifying a schema image” section on page 7–14 for instructions.

**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 Table 2–6 shows.

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Any nonzero value</td>
</tr>
<tr>
<td>False</td>
<td>0</td>
</tr>
</tbody>
</table>

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 *OpenEdge Getting Started: ABL Essentials*. For information about the specific statements and functions, see *OpenEdge Development: 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 statements and functions, see *OpenEdge Development: ABL Reference*.

Oracle CLOB and CFILE data types

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.

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 *OpenEdge Development: 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. Table 2–7 provides definitions of the ABL datetime data types:

Table 2–7: ABL datetime data types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATETIME</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>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.</td>
</tr>
</tbody>
</table>

Oracle datetime data types

Oracle datetime data types provide functionality similar to the ABL datetime data types. Table 2–8 provides their definitions:

Table 2–8: Oracle datetime data types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP WITH LOCAL TIME ZONE</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

Table 2–9 describes the effects of converting from one ABL DATETIME data type to another:

Table 2–9: ABL DATETIME data type conversions

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

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.

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. Table 2–10 provides details on converting ABL DATE and DATETIME, and DATETIME-TZ data types to similar Oracle data types:

<table>
<thead>
<tr>
<th>Source Data Type (OpenEdge)</th>
<th>Destination Data Type (Oracle)</th>
<th>Default Conversion</th>
<th>Non-default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Straight copy of date portion. Time set to midnight up to second precision (00:00:00).</td>
<td>Straight copy of date portion. Time set to midnight up to second precision (00:00:00).</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP</td>
<td>Straight copy of date portion. Time set to midnight (00:00:00.000) up to millisecond precision.</td>
<td>Straight copy of date portion. Time set to midnight (00:00:00.000) up to millisecond precision.</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP-LTZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Source Data Type (OpenEdge)</td>
<td>Destination Data Type (Oracle)</td>
<td>Default Conversion</td>
<td>Non-default Conversion</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP-TZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATE</td>
<td>Straight copy of datetime portion up to second precision. Millisecond portion is dropped.</td>
<td>Straight copy of datetime portion up to second precision. Millisecond portion is dropped.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td>Straight copy of the datetime up to millisecond precision.</td>
<td>Straight copy of the datetime up to millisecond precision.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP-LTZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP-TZ</td>
<td>Straight copy of the datetime portion. Time zone set to the OpenEdge client session time zone context.</td>
<td>Straight copy of the datetime portion. Time zone set to the Oracle session time zone context (defined by the application or through -Dsrrv switch).</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATE</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>TIMESTAMP</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>TIMESTAMP-LTZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>TIMESTAMP-TZ</td>
<td>Straight copy with millisecond precision.</td>
<td>Straight copy with millisecond precision.</td>
</tr>
</tbody>
</table>
Converting Oracle TIMESTAMP types to ABL DATETIME types

Table 2–11 provides details on converting Oracle TIMESTAMP data types to similar OpenEdge data types.

Table 2–11: Converting Oracle TIMESTAMP types to ABL DATETIME types

<table>
<thead>
<tr>
<th>Source Data Type (Oracle)</th>
<th>Destination Data Type (OpenEdge)</th>
<th>Default Conversion</th>
<th>Non-default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Straight copy of date portion. Time portion is dropped.</td>
<td>Straight copy of date portion. Time portion is dropped.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATE</td>
<td>Straight copy of date portion. Time portion is dropped.</td>
<td>Straight copy of date portion. Time portion is dropped.</td>
</tr>
<tr>
<td>TIMESTAMP-LTZ</td>
<td>DATE</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP-TZ</td>
<td>DATE</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
<td>Straight copy of datetime portion up to second level accuracy. Millisecond portion will be set to ‘000’.</td>
<td>Straight copy of datetime portion up to second level accuracy. Millisecond portion will be set to ‘000’.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME</td>
<td>Straight copy with millisecond precision.</td>
<td>Straight copy with millisecond precision.</td>
</tr>
<tr>
<td>TIMESTAMP-LTZ</td>
<td>DATETIME</td>
<td>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.</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 2–11: Converting Oracle TIMESTAMP types to ABL DATETIME types

<table>
<thead>
<tr>
<th>Source Data Type (Oracle)</th>
<th>Destination Data Type (OpenEdge)</th>
<th>Default Conversion</th>
<th>Non-default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP-TZ</td>
<td>DATETIME</td>
<td>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.</td>
<td>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 is dropped.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME-TZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME-TZ</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP-LTZ</td>
<td>DATETIME-TZ</td>
<td>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.</td>
<td>Oracle converts TIMESTAMP-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.</td>
</tr>
<tr>
<td>TIMESTAMP-TZ</td>
<td>DATETIME-TZ</td>
<td>Straight copy with millisecond accuracy.</td>
<td>Straight copy with millisecond accuracy.</td>
</tr>
</tbody>
</table>
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 the “Pre-selection criteria for the schema pull” section on page 2–21 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 datetime attributes, 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 (?). Table 2–12 summarizes how the OpenEdge Unknown value (?), the Oracle unknown, and NULL values map to each other.

Table 2–12: Unknown value (?) and NULL values

<table>
<thead>
<tr>
<th>ABL value</th>
<th>Oracle equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown value (?)</td>
<td>NULL</td>
</tr>
<tr>
<td>&quot;&quot; (Zero-length string)</td>
<td>&quot; &quot; (One space)</td>
</tr>
</tbody>
</table>

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 each of the following topics:

- Comparing record creation behavior
- Ensuring record are created in a predictable order

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:

```ABL
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:
```
```apl
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:

```apl
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.

CREATE master.
ASSIGN master.name = “transaction”.
VALIDATE master.

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:
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. Table 2–13 compares OpenEdge locks to their Oracle equivalents.

Table 2–13: ABL and Oracle locks

<table>
<thead>
<tr>
<th>OpenEdge lock</th>
<th>Oracle lock</th>
<th>Transaction processing option lock¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-LOCK</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>SHARE-LOCK</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>EXCLUSIVE-LOCK</td>
<td>SHARE UPDATE (row-level)</td>
<td>SHARE UPDATE (row-level)</td>
</tr>
<tr>
<td>EXCLUSIVE-LOCK UPDATE</td>
<td>SHARE UPDATE (row-level) ... EXCLUSIVE (table-level)</td>
<td>SHARE UPDATE(row-level) ... ROW EXCLUSIVE (row-level)</td>
</tr>
</tbody>
</table>

¹ The Oracle Transaction Processing option provides a low-level locking manager.

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.
Table 2–14 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.

### Table 2–14: Oracle locking

<table>
<thead>
<tr>
<th>ABL statement</th>
<th>SQL statements generated</th>
<th>Oracle locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND customer.</td>
<td>SELECT ... FROM customer;</td>
<td>None</td>
</tr>
<tr>
<td>FIND customer EXCLUSIVE-LOCK.</td>
<td>SELECT ... FROM customer FOR UPDATE;</td>
<td>Share Update</td>
</tr>
<tr>
<td>FIND customer.</td>
<td>SELECT ... FROM customer FOR UPDATE;</td>
<td>None</td>
</tr>
<tr>
<td>UPDATE customer.</td>
<td>SELECT ... FROM customer FOR UPDATE;</td>
<td>{Row} Exclusive²</td>
</tr>
<tr>
<td>UPDATE customer.¹</td>
<td>Compares records UPDATE customer ...;¹</td>
<td></td>
</tr>
</tbody>
</table>

1. 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.

2. 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.

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 *OpenEdge Getting Started: ABL Essentials* 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.

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 *OpenEdge Getting Started: ABL Essentials* 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 OpenEdge Getting Started: ABL Essentials 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:

```plaintext
/* DataServer returns control here when a duplicate state key is encountered. */
rep-bhk:
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-bhk:
      DO ON ERROR UNDO do-bhk, RETRY do-bhk:
         FIND state WHERE state.state = customer.state.
         DISPLAY state.
      /* OpenEdge DB prompts here on duplicate state key. */
      SET state.
      END.
   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:

<table>
<thead>
<tr>
<th>Cust num</th>
<th>Name</th>
<th>st abbr</th>
<th>State</th>
<th>Sls reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Second Skin Scuba</td>
<td>AZ</td>
<td>Arizona</td>
<td>West</td>
</tr>
</tbody>
</table>

OpenEdge database reprompts for the state abbreviation.
With the OpenEdge DataServer for Oracle, if a duplicate key entry occurs in the 00 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:

<table>
<thead>
<tr>
<th>Cust num</th>
<th>Name</th>
<th>st abbr</th>
<th>State</th>
<th>Sls reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Second Scuba</td>
<td>AZ</td>
<td>Arizona</td>
<td>Nest</td>
</tr>
</tbody>
</table>

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 the “Query tuning” section on page 4–5 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 the “Index cursors” section on page 6–18 for more information.

Issuing the CLOSE QUERY statement after opening a query is another technique that you can use to help minimize the number of cursors your application requires.
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 the “Manually adding PROGRESS_RECID to Oracle” section on page A–3 for instructions.

You can make an existing application that includes RECID behave more consistently across data sources by replacing RECID with ROWID. See the “ROWID function” section on page 2–41 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 NUMBER 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 the “Defining the ROWID” section on page 7–17 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 *OpenEdge Development: 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:

```abl
/* 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:

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

OPEN QUERY myquery NO-LOCK.

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

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:

```abl
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:

```abl
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 Appendix C, “Sample Queries,” 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 *OpenEdge Development: 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” section on page 2–41 for 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 OpenEdge Development: ABL Reference.

R-code

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

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.

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 OpenEdge Development: ABL Reference for information on syntax.
Table 2–15 lists some ABL statements and functions that behave differently between the OpenEdge DataServer for Oracle and a native OpenEdge database.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| BEGINS function Abbreviated Index USING 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 operator MATCHES 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 the “Query tuning” section on page 4–5 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.                                                                                                          |
The OpenEdge DataServer for Oracle does not support this function.

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.

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.

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.

Index reposition might cause new SQL to be executed thereby causing newly created records to appear.

This system handle returns the Oracle DataServer’s time information.

You cannot use this function to change the user ID and password of an Oracle login.

You cannot use this option for a query with the FIELDS option. SHARE-LOCK is NO-LOCK for Oracle.

ABL supports this option only if you have mapped Oracle DATE columns to the OpenEdge CHARACTER fields in the schema image.

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.
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.
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 Chapter 3, “RDBMS Stored Procedure Details,” for a complete discussion of various RDBMS stored procedure and Send SQL statement techniques you can run against the Oracle DataServer.
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.

This chapter includes 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.

Table 3–1 identifies and briefly introduces the elements that comprise a stored procedure definition. These elements are discussed later in this chapter.

### Table 3–1: Stored procedure language elements

<table>
<thead>
<tr>
<th>ABL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RUN STORED-PROCEDURE</code> statement</td>
<td>Executes a stored procedure</td>
</tr>
<tr>
<td><code>PROC-HANDLE</code> phrase</td>
<td>Allows you to specify a handle to identify a stored procedure</td>
</tr>
<tr>
<td><code>PROC-STATUS</code> phrase</td>
<td>Reads the return value</td>
</tr>
<tr>
<td><code>LOAD-RESULT-INTO</code> phrase</td>
<td>Allows data from a result set that is returned for a foreign data source to be put into one or more temp-tables</td>
</tr>
<tr>
<td><code>PARAM</code> phrase</td>
<td>Identifies run-time parameters to be passed to and/or from the stored procedure</td>
</tr>
<tr>
<td><code>CLOSE STORED-PROCEDURE</code> statement</td>
<td>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</td>
</tr>
</tbody>
</table>

**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 the “`RUN STORED-PROCEDURE` details” section on page 3–7 for more details about the reference entries presented in Table 3–1.
As noted in Table 3–1, you can pass data types in the RUN STORED-PROCEDURE statement using the PARAM phrase. Table 3–2 lists issues that occur when you pass certain data types as parameters.

Table 3–2: Argument data types for stored procedures

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>Oracle data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL</td>
<td>The DataServer converts each if these data types in the schema image to the equivalent OpenEdge data type as follows:</td>
</tr>
<tr>
<td>INTEGER</td>
<td>• DECIMAL = NUMBER with precision and scale</td>
</tr>
<tr>
<td>INT64</td>
<td>• INTEGER = CURSOR</td>
</tr>
<tr>
<td></td>
<td>• INT64 = NUMBER with precision</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

| 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     | 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. |
| DATETIME | |
| DATETIME-TZ | |

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:

```plaintext
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.

```sql
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), 'yyyymmdhh24miss'))*3600 + EXTRACT (minute from to_timestamp(to_char(dt_date), 'yyyymmdhh24miss'))*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 “RUN STORED-PROCEDURE details” section on page 3–7 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` statement with 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.

Syntax

```
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 the “`LOAD-RESULT-INTO phrase`” section on page 3–8.

Syntax

```
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 “CLOSED STORED-PROCEDURE statement” section on page 3–10.

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

- The “RUN STORED-PROC statement execution without the LOAD-RESULT-INTO phrase” section on page 3–11
- The “RUN STORED-PROC statement execution with the LOAD-RESULT-INTO phrase” section on page 3–12
- The “RUN STORED-PROC statement with send-sql-statement option” section on page 3–12

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-QL 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 the “Loading a result set into a temp-table” section on page 3–23.

**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.

**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:

**Syntax**

```plaintext
([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.

### CLOSED 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` closes the SQL cursor used by the procedure, as shown:

**Syntax**

```
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 CLOSED 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 the “Data output and retrieval options” section on page 3–14.

**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 the “Data output and retrieval options” section on page 3–14.

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. Example 3–1 shows how the code presented passes a SELECT statement as a parameter.

Example 3–1: Passing a SELECT statement as a parameter

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

Example 3–2 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.

Example 3–2: Using the send-sql-statement option with LOAD-RESULT-INTO option

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.
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 the “Interfacing with RDBMS stored procedures” section on page 3–16 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

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.

Example 3–3 is the basis for the following examples of how to run stored procedures in ABL using this stored procedure created in Oracle.

```sql
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;
```

**Example 3–3: Oracle stored procedure pcust**

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 the “Index cursors” section on page 6–18 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.

Example 3–4 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
/* 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.
```

Example 3–4: ABL code running the stored procedure pcust

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.

Example 3–5 shows how the following ABL procedure uses the second option for passing parameters; it passes them by name with the PARAM option.

```ABL
/* 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.
```

Example 3–5: Passing parameters by name

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

\[
\text{RUN STORED-PROC procedure-name variable = PROC-HANDLE ( parameter-list ).}
\]

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

\[
\text{RUN STORED-PROC open_cust h1 = PROC-HANDLE (CUST_CURS = ?, WITCH_V =1).}
\]

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

\[
\text{FOR EACH buffer-name WHERE PROC-HANDLE = variable}
\quad \text{AND CURSOR = [ [db-name.]procedure-name.]} parameter-name :
\quad \text{DISPLAY buffer-name.}
\quad \text{END.}
\]

\[
\text{RUN STORED-PROC open_cust h1 = PROC-HANDLE (CUST_CURS = ?, WITCH_V1).}
\text{FOR EACH proc-text-buffer WHERE PROC-HANDLE = h1}
\quad \text{AND CURSOR = open_cust.CUST_CURS:}
\quad \text{DISPLAY proc-text-buffer.}
\quad \text{END.}
\text{CLOSE STORED-PROC open_cust.}
\]

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:

Syntax

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

For a complete description, see the “DEFINE BUFFER statement” reference entry in OpenEdge Development: ABL Reference.

Technique to use proc-text-buffer

Example 3–6 illustrates returning database results into the proc-text-buffer and converting the results to the INTEGER data type.

```plaintext
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.
```

Example 3–6: Returning database results into the proc-text-buffer and converting the results to the INTEGER data type

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:

   ```sql
   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 the “Updating a schema image” section on page 7–7 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
```

**Example 3–7** runs the send-sql-statement option twice; procedure handles (through the PROC-HANDLE function) identify the different results from the Oracle database.

**Note:** Example 3–7 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.
```

**Example 3–7: Running the send-sql-statement option twice**

If you use more than one send-sql-statement at a time to send SELECT statements, you must explicitly define procedure handles for each.
Example 3–8 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.

```sql
/* 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 Example 3–8 assumes that the pcust_buffer and porder_buffer exist in the schema and that they match the result-sets schema.

```sql
/* 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.
```

Example 3–8: A stored procedure returning multiple result sets and how to access the result sets using cursor arguments and buffers

As Example 3–8 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 `OpenEdge Getting Started: ABL Essentials`.

Example 3–9 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” section on page 3–12 and introduces the dynamic temp-table topic further discussed in this section.
Example 3–9: Using the RUN STORED-PROC statement with a LOAD-RESULT-INTO phrase

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” section on page 3–25.

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 the “ROWID support” section on page 3–34. For in-depth discussions of temp-tables and more information about ProDataSets, see *OpenEdge Development: ProDataSets*.

Table 3–3 highlights additional language elements you can use with the stored procedure and the send-sql statement language to use ROWID.

**Table 3–3: Returning result sets and loading the data into temp-tables**

<table>
<thead>
<tr>
<th>ABL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RUN STORED–PROCEDURE statement</strong></td>
<td>Executes the stored procedure or send-sql-statement option and tells OpenEdge that the stored procedure has ended.</td>
</tr>
<tr>
<td><strong>LOAD–RESULT–INTO phrase</strong></td>
<td>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 using the send-sql-statement option. <strong>Note:</strong> When using SQL statement(s) through a send-sql-statement option or stored procedure to load result sets into temp-tables, <strong>RUN STORED–PROC</strong> carries an implicit Run <strong>CLOSE–PROC</strong> statement. (The stored procedure’s output parameters are available after the <strong>RUN STORED–PROC</strong> executes and closes the procedure.)</td>
</tr>
<tr>
<td><strong>PROC–STATUS phrase</strong></td>
<td>Reads the return value (optional).</td>
</tr>
</tbody>
</table>

**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.

Table 3–4 identifies the temp-table options for which you can plan and the requirements you must fulfill for each option.

### Table 3–4: Options to plan the temp-table layout for result sets

<table>
<thead>
<tr>
<th>To return a result set to this type of temp-table . . .</th>
<th>Then the layout definition is . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Dynamic-prepared state</td>
<td>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.</td>
</tr>
<tr>
<td>Dynamic - unprepared state</td>
<td>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 the “Details about a dynamic temp-table in an unprepared state” section on page 3–27.</td>
</tr>
</tbody>
</table>

**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:

```sql
/* */
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 Chapter 2, “Initial Programming Considerations.”

**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

Example 3–10 shows multiple dynamic temp-tables in a prepared state. It is based on the stored procedure code presented in Example 3–8 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.

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

Example 3–10: Dynamic temp-tables shown in a prepared state

Additional temp-table examples

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

Example 3–11 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 3–8 comparable to the way you can see it works with ABL code.

```abl
/* 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).
```

Example 3–11: Executing a stored procedure call using the LOAD-RESULT-INTO phrase
Example 3–12 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").
```

Example 3–12: Using an existing temp-table without the TEMP-TABLE-PREPARE() method

Example 3–13 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 ttHndl.
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
("select custNum, name from myDB.customer").
```

Example 3–13: Using an existing temp-table with the TEMP-TABLE-PREPARE() method

Note these points as they relate to Example 3–13:

- 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.
Example 3–14 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.

```sql
/* 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").
```

**Example 3–14: A stored procedure that uses the LOAD–RESULT–INTO phrase with a single temp table and the send-sql-statement option**

Example 3–15 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.

```sql
/* 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.
```

**Example 3–15: Using the PROC–STATUS phrase**

In Example 3–15, note that the PROC–STATUS phrase 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.

Example 3–16 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.

```sql
/* 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;
/*
```
Example 3–16: Executing a call to a stored function that returns a result set using the LOAD-RESULT-INTO phrase.
Handling errors

The RUN STORED-PROC statement shows how the RUN STORED-PROC statement supports the NO-ERROR option. Example 3–17 shows how to trap errors within a stored procedure.

```abl
DEFINE VAR ix AS INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement NO-ERROR
   ("select count(*) from xxx.customer where name between 'A' and 'Z'").
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.
END.
CLOSE STORED-PROC send-sql-statement.
```

Example 3–17: How to trap errors within a stored procedure

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. Example 3–18 shows how errors are trapped after LOAD-RESULT-INTO stored procedure execution.

```abl
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.
```

Example 3–18: Trapping errors when using LOAD-RESULT-INTO

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.
- 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, focusing on:

- Understanding the ROWID Implementation
- Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO
- Additional ProDataSet support

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 Chapter 2, “Initial Programming Considerations.”

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 the “ROWID—Standard ABL behavior” section on page 3–34 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.
Example 3–19 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".
```

Example 3–19: Expected ABL behavior—ROWID value of a Temp-Table buffer

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

Example 3-20 assumes you migrated your database to Oracle using the CREATE RECID option, as discussed in the “ROWID characteristics” section on page 3–34. 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.

**Example 3–20** 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.

```abl
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.
```

**Example 3–20: Simple ProDataSet code**
Example 3–21 combines code from Example 3–19 and Example 3–20 by 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 Example 3–21 are presented following the example.

```abl
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.
```

Example 3–21: Using the LOAD-RESULT-INTO technique to populate the underlying temp-table of a ProDataSet
Keep the following key points in mind as you review Example 3–21:

- The TEMP-TABLE field that is mapped to the PROGRESS_RECID column should be changed from its standard definition of INTEGER to ROWID. In Example 3–21, 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:

```plaintext
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 Example 3–21, shows this definition:

```plaintext
DEFINE DATASET dsCustomer FOR ttCustomer.
```

- The technique, demonstrated in Example 3–21, 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 Example 3–21:

```plaintext
FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-1 = ROWID(customer).
```

In contrast, the following excerpt from Example 3–21 demonstrates an alternative use of the ROWID value with a temp-table:

```plaintext
FIND FIRST customer WHERE customer.custnum = 1 NO-LOCK.
rid-2 = ttCustomer.Recid.
```
Additional ProDataSet support

As demonstrated in Example 3–21, the LOAD-RESULT-INTO functionality provides a very fast and efficient way to populate the temp-table(s) of a ProDataSet Object.

Example 3–22 builds on the techniques demonstrated in Example 3–21, but 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.
```

Example 3–22: Using the LOAD-RESULT-INTO technique with BEFORE-FILL method to fill the temp-table(s) of a ProDataSet
This chapter presents additional features to enhance the Oracle DataServer performance, as described in the following sections:

- Enhancements overview
- SQL and ABL queries
- Bind variables
- Query tuning
- Caching records
- Oracle hints
- Using field lists when updating records
- Join by SQL DB
- Skipping schema verification
- Writing queries for performance
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 the “Query tuning with connection and startup parameters” section on page 6–14 for information on tuning queries at compile time and run time. See Appendix C, “Sample Queries,” 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.
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**

  **Syntax**
  
  ```abl
  FOR EACH table QUERY-TUNING
       ( query-tuning-option query-tuning-option ... )
  ```

- **OPEN QUERY**

  **Syntax**
  
  ```abl
  OPEN QUERY query QUERY-TUNING
       ( query-tuning-option query-tuning-option ... )
  ```

- **DO PRESELECT**

  **Syntax**
  
  ```abl
  DO PRESELECT table QUERY-TUNING
       ( query-tuning-option query-tuning-option ... )
  ```

- **REPEAT PRESELECT**

  **Syntax**
  
  ```abl
  REPEAT PRESELECT table QUERY-TUNING
       ( query-tuning-option query-tuning-option ... )
  ```
Additional Features to Enhance DataServer Performance

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.

Table 4–1 describes the query-tuning options.

**Table 4–1: Query-tuning options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY−MESSAGE</td>
<td>Specifies whether the DataServer sends multiple result rows in a single logical network message, thereby reducing network traffic.</td>
</tr>
<tr>
<td>NO-ARRAY−MESSAGE</td>
<td>Default: ARRAY−MESSAGE, if the query uses a lookahead cursor.</td>
</tr>
<tr>
<td>BIND−WHERE</td>
<td>Specifies whether the DataServer uses Oracle bind variables for values in WHERE clauses. Using bind variables typically improves performance,</td>
</tr>
<tr>
<td>NO-BIND−WHERE</td>
<td>but Oracle provides unexpected results for some operations, such as MATCHES on an indexed field and a trailing wildcard or comparisons of CHAR fields</td>
</tr>
<tr>
<td></td>
<td>that use Oracle’s blank-padding rules.</td>
</tr>
<tr>
<td></td>
<td>Specify NO-BIND−WHERE to use literals. Using NO-BIND−WHERE in queries that do comparisons (MATCHES, BEGINS on an indexed field) can improve</td>
</tr>
<tr>
<td></td>
<td>performance.</td>
</tr>
<tr>
<td></td>
<td>Default: BIND−WHERE.</td>
</tr>
<tr>
<td>CACHE−SIZE integer</td>
<td>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</td>
</tr>
<tr>
<td>BYTE</td>
<td>cause the DataServer to generate identical SQL code except that the second statement specifies a smaller cache size, the DataServer reuses the</td>
</tr>
<tr>
<td></td>
<td>larger cache from the first statement if the cursor is still available. Reusing cache and cursors improves performance.</td>
</tr>
<tr>
<td>ROW</td>
<td>Byte maximum: 65535 bytes.</td>
</tr>
<tr>
<td></td>
<td>Byte minimum: Specify the number of bytes contained in a single record. For joins, specify the number of bytes contained in two joined records.</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>record, you need a cache of at least 1000 bytes.</td>
</tr>
<tr>
<td></td>
<td>Default: 1024 bytes with standard cursors; 8192 with lookahead cursors.</td>
</tr>
<tr>
<td></td>
<td>Row maximum: the number of records that can fit in 65535 bytes.</td>
</tr>
<tr>
<td></td>
<td>See the “Caching records” section on page 4–10 for more information.</td>
</tr>
<tr>
<td></td>
<td>Row minimum: 1 for a single table; 2 for a join.</td>
</tr>
</tbody>
</table>
### Table 4–1: Query-tuning options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEBUG EXTENDED</strong></td>
<td>Specifies whether the DataServer should print debugging information that it generates for the query to the dataserv.1g file.</td>
</tr>
<tr>
<td>DEBUG SQL</td>
<td>Specify DEBUG SQL to print the SQL that the DataServer executes against the Oracle DBMS.</td>
</tr>
<tr>
<td>NO-DEBUG</td>
<td>Specify DEBUG EXTENDED to print additional information, such as cursor statistics.</td>
</tr>
<tr>
<td></td>
<td>There are additional options for collecting advanced statistics with DEBUG. See the “Analyzing performance” section on page 6–17 for more information.</td>
</tr>
<tr>
<td></td>
<td>Default: NO-DEBUG.</td>
</tr>
<tr>
<td><strong>HINT string1 string2 string3</strong></td>
<td>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 (/<em>+) and closing symbols (+/). For example, to pass the /</em>+ORDERED*/ hint syntax, you specify only HINT &quot;ORDERED&quot;. 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.</td>
</tr>
<tr>
<td><strong>INDEX–HINT</strong></td>
<td>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.</td>
</tr>
<tr>
<td>NO–INDEX–HINT</td>
<td>Specify NO–INDEX–HINT to test whether performance for a query improves when the DataServer executes it without hints. See the “Indexes” section on page 2–8 for more information on index hints.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Default: INDEX–HINT.</td>
</tr>
</tbody>
</table>
### Table 4–1: Query-tuning options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN-BY-SQLDB</td>
<td>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.</td>
</tr>
<tr>
<td>NO-JOIN-BY-SQLDB</td>
<td></td>
</tr>
<tr>
<td>LOOKAHEAD</td>
<td>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.</td>
</tr>
<tr>
<td>NO-LOOKAHEAD</td>
<td></td>
</tr>
<tr>
<td>NO-QUERY-ORDER-ADDED</td>
<td>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. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>NO-QUERY-UNIQUE-ADDED</td>
<td>Specifies that OpenEdge should omit the record identifier from the end of the query’s generated ORDER BY clause 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. <strong>Note:</strong> 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.</td>
</tr>
</tbody>
</table>


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:
```
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.1g 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.
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 when you compile.

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:
```
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 \([\text{NO-}]\)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. Table 4–2 describes how these controls interact and affect the behavior.

### Table 4–2: Controlling join by SQLDB

<table>
<thead>
<tr>
<th>QUERY-TUNING</th>
<th>Startup parameter</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN-BY-SQLDB</td>
<td>-nojoinbysqldb</td>
<td>Oracle performs the join if possible</td>
</tr>
<tr>
<td>JOIN-BY-SQLDB</td>
<td>None</td>
<td>Oracle performs the join if possible</td>
</tr>
<tr>
<td>NO-JOIN-BY-SQLDB</td>
<td>-nojoinbysqldb</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>NO-JOIN-BY-SQLDB</td>
<td>None</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>None</td>
<td>-nojoinbysqldb</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>Oracle performs the join if possible and if the join is a true join</td>
</tr>
</tbody>
</table>

Join by SQL DB does not occur by default for the following query:

```sql
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**

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.
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 **Data Administration** → **DataServer** → **Oracle Utilities** 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 the “Analyzing performance” section on page 6–17 for information on collecting statistics.
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, as outlined in the following sections:

- DataServer modules
- Configuring Windows modules
- Configuring UNIX modules
- Configuring an international environment
- Configuring the DataServer in the Unified Broker administration framework
- Creating, Maintaining, and Deploying a schema holder
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. Table 5–1 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 5–1: DataServer Configurations

<table>
<thead>
<tr>
<th>Client</th>
<th>DataServer</th>
<th>Installing and configuring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Local Windows</td>
<td>Use the default OpenEdge client executable (<code>prowin32.exe</code>). This executable also allows you to use Oracle networking to access Oracle.</td>
</tr>
<tr>
<td>Windows</td>
<td>Remote Windows</td>
<td>On the client machine, use the default OpenEdge client executable (<code>prowin32.exe</code>). In the Windows host machine, use either the default broker executable (<code>_probrkr.exe</code>) or configure the broker using Progress Explorer.</td>
</tr>
<tr>
<td>Windows</td>
<td>Remote UNIX</td>
<td>On the client machine, use the default OpenEdge client executable (<code>prowin32.exe</code>). On the UNIX host machine, use the default broker executable (<code>_probrkr</code>) and set <code>ORASRV</code> to point to the <code>_orasrv</code> DataServer executable. This executable supports OpenEdge networking.</td>
</tr>
<tr>
<td>UNIX</td>
<td>Local UNIX</td>
<td>On the UNIX client machine, use the default client executable (<code>_progres</code>). This executable also allow you to use Oracle networking to access Oracle.</td>
</tr>
<tr>
<td>UNIX</td>
<td>Remote UNIX</td>
<td>On the UNIX client machine, use the default client executable (<code>_progres</code>). On the UNIX host machine, use the default broker executable (<code>_probrkr</code>) and set <code>ORASRV</code> to point to the <code>_orasrv</code> DataServer executable. These executables support OpenEdge networking.</td>
</tr>
</tbody>
</table>
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 Appendix D, “Building DataServer Executables,” for instructions.

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 the “Configuring UNIX modules” section on page 5–6. If you are building a remote DataServer configuration with a UNIX client accessing a Windows host, see the “Configuring UNIX modules” section on page 5–6 for instructions on setting up the client and the “Configuring Windows modules” section on page 5–4 for instructions on setting up the host.

---

**Table 5–1: DataServer Configurations**

<table>
<thead>
<tr>
<th>Client</th>
<th>DataServer</th>
<th>Installing and configuring</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>Remote Windows</td>
<td>On the UNIX client machine, use the default client executable (_progres).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the Windows host machine, use either the default broker executable (_probrkr.exe) or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configure using Progress Explorer.</td>
</tr>
</tbody>
</table>

**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 Appendix D, “Building DataServer Executables,” for instructions.
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 Table 5–1 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 the “Configuring UNIX modules” section on page 5–6 for information.
- **In Windows** — See the “Configuring the local DataServer in Windows” section on page 5–4 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 Table 5–2 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. The Progress Explorer utility provides a graphical interface that allows you to manage and configure the DataServer host environment. See the “Configuring the DataServer in the Unified Broker administration framework” section on page 5–10 for instructions.

If you do not choose to use the Progress Explorer and want to set environment variables manually, see Table 5–2.

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 Table 5–2.

Table 5–2: DataServer for Oracle environment variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle_HOME</td>
<td>The pathname of the directory where you installed Oracle.</td>
</tr>
<tr>
<td>Oracle_SID</td>
<td>The identifier for the Oracle instance you are accessing.</td>
</tr>
<tr>
<td>ORASRV</td>
<td>The name of the executable (including the path) of the</td>
</tr>
<tr>
<td></td>
<td>Oracle DataServer. If you are not using the default</td>
</tr>
<tr>
<td></td>
<td>DataServer executable, _orasrv.exe, set ORASRV to point to the</td>
</tr>
<tr>
<td></td>
<td>appropriate executable.</td>
</tr>
<tr>
<td>PROSTARTUP</td>
<td>The pathname of your default OpenEdge startup (.pf) file.</td>
</tr>
<tr>
<td></td>
<td>To set the Message Buffer Size (-Mm) startup parameter for the broker</td>
</tr>
<tr>
<td></td>
<td>to a value different from the 1924 default buffer size, you must set</td>
</tr>
<tr>
<td></td>
<td>the PROSTARTUP environment variable equal to the path and name of the</td>
</tr>
<tr>
<td></td>
<td>.pf file that contains the -Mm parameter.</td>
</tr>
<tr>
<td></td>
<td>This approach ensures that your value is recognized and therefore</td>
</tr>
<tr>
<td></td>
<td>used by the broker when it starts. Only Windows clients must use this</td>
</tr>
<tr>
<td></td>
<td>technique.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you use the command line utility and intend to set the</td>
</tr>
<tr>
<td></td>
<td>-Mm startup parameter to a value other than 1024, you must use the .pf</td>
</tr>
<tr>
<td></td>
<td>file with the PROSTARTUP variable set.</td>
</tr>
</tbody>
</table>

3. Start the broker. See the “Starting the DataServer processes” section on page 5–17 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 the “Creating, Maintaining, and Deploying a schema holder” section on page 5–15 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 Table 5–1 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. Table 5–3 describes the environment variables and how to set them for the local DataServer.

**Table 5–3:** Environment variables for the local DataServer

<table>
<thead>
<tr>
<th>Variable</th>
<th>How to set it</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLOGDIR</td>
<td>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)</td>
</tr>
<tr>
<td>Oracle_HOME</td>
<td>The pathname of the directory where Oracle is installed. (Required)</td>
</tr>
<tr>
<td>Oracle_SID</td>
<td>The identifier for the Oracle instance you are accessing. (Required)</td>
</tr>
</tbody>
</table>

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 the “Creating, Maintaining, and Deploying a schema holder” section on page 5–15 for instructions.
Configuring UNIX modules

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 Table 5–4. 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 Table 5–4 that you set on the client is PROEXE.

Table 5–4: Environment variables for the remote DataServer

<table>
<thead>
<tr>
<th>Variable</th>
<th>How to set it</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLOGDIR</td>
<td>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)</td>
</tr>
<tr>
<td>Oracle_HOME</td>
<td>The pathname of the directory where Oracle is installed. (Required)</td>
</tr>
<tr>
<td>Oracle_SID</td>
<td>The identifier for the Oracle instance you are accessing. (Required)</td>
</tr>
<tr>
<td>ORASRV</td>
<td>The name of the executable (including the path) of the Oracle DataServer. Set ORASRV to point to the appropriate DataServer executable. (Required)</td>
</tr>
<tr>
<td>PROBRKR</td>
<td>The pathname to the executable of the broker. (Optional)</td>
</tr>
<tr>
<td>PROSTARTUP</td>
<td>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. <strong>Note:</strong> 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.</td>
</tr>
</tbody>
</table>

Once you have set up your environment, you are ready to create a schema holder. See the “Creating, Maintaining, and Deploying a schema holder” section on page 5–15 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 the “Support for Unicode” section on page 2–6. For more information on the capabilities OpenEdge has for internationalized applications, see OpenEdge Development: Internationalizing 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 the “Creating, Maintaining, and Deploying a schema holder” section on page 5–15 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:

```bash
prowin32 -cpinternal code-page -cpstream code-page
schema-holder-name -db oracle-dbname
-l 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:

```bash
_prowin32 -cpinternal code-page -cpstream code-page
schema-holder-name -db oracle-dbname
-l 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:

```bash
_progres -cpinternal code-page -cpstream code-page
schema-holder-name -l -db oracle-dbname
-l 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:

```bash
_progres -cpinternal code-page -cpstream code-page
schema-holder-name -l -db oracle-dbname
-l 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:

```bash
_progres -cpinternal code-page -cpstream code-page
schema-holder-name -db oracle-dbname
-l oracle-logical-dbname -S service-name -H host
-U userid -P password -b -p procedure-name
```
Configuring the DataServer in the Unified Broker administration framework

Integrating a DataServer into the Progress Explorer administration 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 Explorer administration framework supports, see *OpenEdge Application Server: Developing AppServer Applications* and *OpenEdge Application Server: Developing WebSpeed Applications*.

Configuring the Explorer administration framework in Windows

In Windows, you use the Progress Explorer tool to configure the OpenEdge DataServer for Oracle in the Explorer administration 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 Progress Explorer.
3. Connect to the local/remote AdminServer.
4. From the Progress Explorer’s left pane, 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 the “Starting the DataServer in the Explorer administration framework” section on page 6–3 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.

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 the Progress Explorer and details on the required settings, see the Progress Explorer online help.

**Configuring the Progress Explorer administration framework on UNIX**

On UNIX, you configure the Progress Explorer administration 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 *OpenEdge Getting Started: Installation and Configuration*.

When you configure the OpenEdge DataServer for Oracle, you are providing information that the Explorer administration 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.
Table 5–5 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 5–5: OpenEdge DataServer for Oracle sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[UBroker.OR]</code></td>
<td>Parent entity of the OpenEdge DataServer for Oracle brokers. It defines default property settings for all OpenEdge DataServer for Oracle Broker instances.</td>
</tr>
</tbody>
</table>
| `[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` — Do 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. |
| `[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:

```plaintext
# # 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 # #
[Environment.orabroker1]
    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 Appendix E, “DataServer Command Line Utilities and Startup Parameters,” 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 Appendix E, “DataServer Command Line Utilities and Startup Parameters,” 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 OpenEdge Getting Started: Installation and Configuration 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.
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.
Table 5–6 describes the permissions that you need for creating a schema holder.

Table 5–6: Required Oracle permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE SESSION</td>
<td>Database</td>
</tr>
<tr>
<td>SELECT</td>
<td>System objects:</td>
</tr>
<tr>
<td></td>
<td>sys.argument$</td>
</tr>
<tr>
<td></td>
<td>sys.col$</td>
</tr>
<tr>
<td></td>
<td>sys.com$</td>
</tr>
<tr>
<td></td>
<td>sys.con$</td>
</tr>
<tr>
<td></td>
<td>sys.dual</td>
</tr>
<tr>
<td></td>
<td>sys.iol$</td>
</tr>
<tr>
<td></td>
<td>sys.ind$</td>
</tr>
<tr>
<td></td>
<td>sys.lnk$</td>
</tr>
<tr>
<td></td>
<td>sys.obj$</td>
</tr>
<tr>
<td></td>
<td>sys.probure$</td>
</tr>
<tr>
<td></td>
<td>sys.seq$</td>
</tr>
<tr>
<td></td>
<td>sys.syn$</td>
</tr>
<tr>
<td></td>
<td>sys.tab$</td>
</tr>
<tr>
<td></td>
<td>sys.ts$</td>
</tr>
<tr>
<td></td>
<td>sys.user$</td>
</tr>
<tr>
<td></td>
<td>sys.view$</td>
</tr>
</tbody>
</table>

Permissions for connecting a schema holder

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.
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 \texttt{ORACLE\_HOME} variable to the main Oracle directory.

2. Set the \texttt{ORACLE\_SID} variable to the Oracle system identifier.

   A process can have the \texttt{ORACLE\_SID} set to only one value at a time. If you want to access multiple Oracle databases on the same machine, set an \texttt{ORACLE\_SID} variable to a different value for each database. Then create a separate broker for each database with the \texttt{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 Chapter 6, “Connecting the DataServer.”

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.

To start a remote DataServer process in Windows using a UNIX client:

1. Start the broker you configured for the Oracle DataServer.

2. Run the client executable (\texttt{_progres}) on the client machine.

To start a remote DataServer process in Windows using a Windows client:

1. Start the broker you configured for the Oracle DataServer.

2. Run the client executable (\texttt{prowin32.exe}) on the client machine.

To start a local DataServer on UNIX, run the client executable (\texttt{_progres}).
To start a remote DataServer on UNIX using a UNIX client:

1. 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

2. Run the client executable (_progres) on the client machine.

To start a remote DataServer on UNIX using a Windows client:

1. 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

2. Run the client executable (prowin32.exe) on the client machine.

To start a local DataServer in Windows, run the client executable (prowin32.exe).

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 OpenEdge Data Management: Database Administration 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:

You do not have to provide any additional connection information. See OpenEdge Data Management: Database Administration 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 OpenEdge Getting Started: ABL Essentials.
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 the “Code pages” section on page 2–3 for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see *OpenEdge Development: Internationalizing 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 the “Connecting a schema holder at startup” section on page 6–9 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. Table 5–6 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.

10. Choose OK.

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 the “Defining the ROWID” section on page 7–17 for instructions. See the “ROWID function” section on page 2–41 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. Chapter 7, “The DataServer Tutorial,” 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.

Table 5–7 lists common database operations and the required techniques.

**Table 5–7: Database operations and the schema holder**

<table>
<thead>
<tr>
<th>Database operation</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify a column in an Oracle table</td>
<td>Update the schema image by using the OpenEdge DataServer for Oracle utilities.</td>
</tr>
<tr>
<td>Add a column to an Oracle table</td>
<td>Update the schema image by using the OpenEdge DataServer for Oracle utilities.</td>
</tr>
<tr>
<td>Insert a row in an Oracle table</td>
<td>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.</td>
</tr>
<tr>
<td>Add an index to an Oracle table</td>
<td>Update the schema image by using the DataServer for Oracle utilities.</td>
</tr>
<tr>
<td><strong>Note:</strong> The index cannot be a function-based index.</td>
<td></td>
</tr>
<tr>
<td>Modify data definitions in the schema image</td>
<td>Update the Oracle database to reflect your changes by using the Incremental Schema Migration utility.</td>
</tr>
</tbody>
</table>

**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.

You can improve the performance of a deployed application by using the -Dsrv skip-schema-check startup parameter. See the “Skipping schema verification” section on page 4–15 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 Progress Explorer. By default, the server will be started as a Windows service. See OpenEdge Data Management: Database Administration for a complete description of how to configure a database server.
Configuring the DataServer
Connecting the DataServer

This chapter provides instructions for running the DataServer after you have configured its modules and created a schema holder, as outlined in the following sections:

- 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:

<table>
<thead>
<tr>
<th>For this variable . . .</th>
<th>Set . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLOGDIR</td>
<td>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)</td>
</tr>
<tr>
<td>ORACLE_HOME</td>
<td>The pathname of the directory where Oracle is installed.</td>
</tr>
<tr>
<td>ORACLE_SID</td>
<td>The identifier for the Oracle instance you are accessing.</td>
</tr>
</tbody>
</table>

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 orademo -dt ORACLE -ld orademo -U scott -P tiger

For Oracle Networking startup information, see the “Connecting through Oracle Networking” section on page 6–12.
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 Explorer administration framework

The Progress Explorer administration 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 Explorer administration framework in Windows and on UNIX.

**In the Windows host**

Use the Progress Explorer tool to run DataServer processes in the Windows host. Optionally you can use the command-line technique described in the “On the UNIX host” section on page 6–4. To configure the DataServer, see the “Configuring the DataServer in the Unified Broker administration framework” section on page 5–10. This section presents basic start, stop, and status check procedures for the DataServer.

To start the DataServer:

1. From the Progress Explorer’s left pane, select the Oracle DataServer folder and double-click. The list of existing DataServer brokers for Oracle appears in the right pane.
2. Select the DataServer broker you want to start and right-click. A pop-up menu appears.
3. Choose the Start option from the pop-up menu. The Status in the right pane changes to Running for that DataServer broker.

To check the status of a DataServer broker:

1. Select the DataServer broker and right-click.
2. Choose the Status option from the pop-up menu.

See the Progress Explorer online Help for more information the various Status tabs.
To stop the DataServer broker:

1. Select the DataServer broker and right-click.

2. Choose the Stop option from the pop-up menu. The Status in the right pane changes to Not Running for that DataServer broker.

For more information on using the Progress Explorer tool, see the online Help.

On the UNIX host

Use the Progress Explorer administration framework command-line interface to run DataServer processes on the UNIX host. To configure the DataServer, see the “Configuring the DataServer in the Unified Broker administration framework” section on page 5–10.

To 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
```

You can stop a broker on a remote machine by adding the -host and -user options.

See Appendix E, “DataServer Command Line Utilities and Startup Parameters,” 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 broker process via the command line

The broker can be started from a Windows or UNIX command line prompt in addition to the Progress 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. Table 6–1 describes how to set them.

### Table 6–1: Environment variables for the remote DataServer

<table>
<thead>
<tr>
<th>Variable</th>
<th>How to set it</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLOGDIR</td>
<td>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)</td>
</tr>
<tr>
<td>ORACLE_HOME</td>
<td>The pathname of the directory where Oracle is installed.</td>
</tr>
<tr>
<td>ORACLE_SID</td>
<td>The identifier for the Oracle instance you are accessing.</td>
</tr>
<tr>
<td>ORASRV</td>
<td>The name of the executable (including the path) of the Oracle DataServer.</td>
</tr>
<tr>
<td>PROBRKR</td>
<td>The pathname to the executable of the broker.</td>
</tr>
</tbody>
</table>

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
```

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 the “Connecting a schema holder at startup” section on page 6–9 and the “Optional connection and startup parameters” section on page 6–13 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 -ld orademo -H host1 -S oserviceA -U scott -P tiger
```

See the “Connecting a schema holder at startup” section on page 6–9 and the “Optional connection and startup parameters” section on page 6–13 for descriptions of the required command line.
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 *OpenEdge Development: 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.

Table 6–2 describes the connection parameters that relate to the remote DataServer and networking options.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name (-H)</td>
<td>Indicates the name of the host machine in the network.</td>
</tr>
<tr>
<td>Service Name (-S)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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
Table 6–3 describes the parameters required for connecting to a schema holder and an Oracle database.

### Table 6–3: Required connection parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID (-U)</td>
<td>Supplies the user ID that the OpenEdge DataServer for Oracle uses to log into the Oracle RDBMS. If you specify <code>-U userid/password</code> or <code>-U userid/password@service-name</code>, the USERID function returns the password information or the password and service in addition to the user ID.</td>
</tr>
<tr>
<td>Password (-P)</td>
<td>Supplies the password that the OpenEdge DataServer for Oracle uses to log into the Oracle RDBMS unless it was specified with the <code>-U</code> parameter.</td>
</tr>
<tr>
<td>Physical Database Name (-db)</td>
<td>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 <code>-db</code> parameter or any other designation, such as <code>oradb</code>.</td>
</tr>
<tr>
<td>Single-user Mode (-1) or Read-only (-RO)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 *OpenEdge Deployment: 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 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:

```
/0<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.

Table 6–4 describes parameters that are especially relevant to using the OpenEdge DataServer for Oracle.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type (<code>-dt ORACLE</code>)</td>
<td>Specifies that the non-OpenEdge type of the target database is Oracle.</td>
</tr>
<tr>
<td>DataServer (<code>-Dsrv</code>)</td>
<td>Specifies parameters that allow you to control how the DataServer processes queries. <code>-Dsrv</code> is a connection parameter. See the “Query tuning with connection and startup parameters” section on page 6–14 for more information.</td>
</tr>
<tr>
<td>Logical Database Name (<code>-ld</code>)</td>
<td>Specifies the logical name for the Oracle database. This is the name by which you refer to the database in your applications.</td>
</tr>
<tr>
<td>Index Cursors (<code>-c</code>)</td>
<td>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 the “Index cursors” section on page 6–18 for more information.</td>
</tr>
</tbody>
</table>
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 the “Query tuning” section on page 4–5 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:

**Syntax**

```sql
CONNECT db-name -U user-name -P password
   -Dsrv query-tuning-option1,value1,
      query-tuning-option2,value2.
```

Table 6–5 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_bind_where</td>
<td>Specifies whether the DataServer uses Oracle bind variables for values in WHERE clauses.</td>
</tr>
<tr>
<td>qt_no_bind_where</td>
<td>Specify qt_no_bind_where to use literals.</td>
</tr>
<tr>
<td></td>
<td>Use at run time only.</td>
</tr>
<tr>
<td></td>
<td>Default: qt_bind_where.</td>
</tr>
<tr>
<td>qt_cache_size,integer, QT_BYTE</td>
<td>Specifies the size of the cache (in bytes or records) for information used by lookahead or standard cursors.</td>
</tr>
<tr>
<td></td>
<td>Byte maximum: 65535 bytes.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Row maximum: The number of records that can fit in 65535 bytes. See the “Caching records” section on page 4–10 for more information.</td>
</tr>
<tr>
<td></td>
<td>Row minimum: 1. If the server performs the join, the minimum is 2.</td>
</tr>
<tr>
<td></td>
<td>If QT_BYTE or QT_ROW is omitted, QT_BYTE is assumed.</td>
</tr>
<tr>
<td></td>
<td>Default: 1024 bytes with standard cursors; 8192 with lookahead cursors.</td>
</tr>
<tr>
<td>qt_lookahead</td>
<td>Specifies whether the DataServer uses lookahead or standard cursors.</td>
</tr>
<tr>
<td>qt_no_lookahead</td>
<td>Specify qt_no_lookahead for behavior that is consistent with OpenEdge.</td>
</tr>
<tr>
<td></td>
<td>Default: qt_lookahead, except with statements that use an EXCLUSIVE lock.</td>
</tr>
</tbody>
</table>
The following example shows how to use the query-tuning options to enhance performance. The multiple records that the lookahead cursors require are stored in a 32K cache. In addition, the DataServer writes an extended report on the SQL statements it executes, as shown:

```
CONNECT oradb -U user -P password
-Dsrv qt_cache_size,32000,qt_debug,EXTENDED.
```

Use startup parameters to override two other query-tuning defaults, INDEX-HINT and JOIN-BY-SQLDB. Table 6–6 describes these startup parameters.

**Table 6–6: Query-tuning startup parameters**

<table>
<thead>
<tr>
<th>Startup Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Index Hint (-noindexhint) | • Specifies that the DataServer 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 (-nojoinsideqldb) | • Specifies that the client evaluates and performs queries that have joins. This might slow performance, but provides results that are consistent with OpenEdge behavior.  
  • Use -nojoinsideqldb at compile time. It has no effect at run time.  
  **Note:** Server joins are never performed for dynamic queries. |
| 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. |
Analyzing performance

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.1g` 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:

Syntax

```
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. Table 6–7 lists the diagnostic capabilities of `qt_debug`.

Table 6–7: Diagnostic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Specify <code>qt_debug,SQL</code> to print the SQL that the DataServer executes against the Oracle DBMS in the <code>dataserv.1g</code> file.</td>
</tr>
<tr>
<td>EXTENDED</td>
<td>Specify <code>qt_debug,EXTENDED</code> to print the SQL information that the SQL option generates, plus additional information such as cursor statistics in the <code>dataserv.1g</code> file.</td>
</tr>
<tr>
<td>CURSOR</td>
<td>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.</td>
</tr>
<tr>
<td>DATA-BIND</td>
<td>Information about the data types, buffer sizes, and addresses that the DataServer and Oracle use when binding variables.</td>
</tr>
<tr>
<td>PERFORMANCE</td>
<td>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 <code>tkprof</code> utility reports might be due to network performance issues, rather than to DataServer or Oracle behavior.</td>
</tr>
</tbody>
</table>
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 `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 TO 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_ TO 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 TO
OCI call OCIStmtExecute <4>
OCI call OCIHandleAlloc <0>
OCI call OCIStmtPrepare <5>     sqlcrc = 48586
   SELECT /*+ INDEX_ASC(T0 ORDER_##CUST_ORDER) */ PROGRESS_RECID unique_id_0,ORDER_NUM,CUST_NUM,ORDER_DATE,SHIP_DATE,PROMISE_DATE,ARRIER,INSTRUCTIONS,PO,TERMS,SALES_REP,PROGRESS_RECID FROM DOCTEST.ORDER_ TO 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.

### 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.
Connecting the DataServer

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:

**Syntax**

```
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 *OpenEdge Getting Started: ABL Essentials* and the “SAVE CACHE Statement” reference entry in *OpenEdge Development: ABL Reference*. 
Table 6–8 lists circumstances under which a connection might fail and describes the responses you might receive.

**Table 6–8: Failure responses**

<table>
<thead>
<tr>
<th>Failure circumstances</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>During startup</td>
<td>The system displays an error message and returns to the operating system prompt.</td>
</tr>
<tr>
<td>During a CONNECT statement</td>
<td>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 <strong>NO-ERROR</strong> modifier with the CONNECT statement to trap run-time errors. If you use the <strong>NO-ERROR</strong> 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.</td>
</tr>
<tr>
<td>During an attempted auto-connect</td>
<td>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.</td>
</tr>
<tr>
<td>During an attempt to connect using the Data Dictionary</td>
<td>The Data Dictionary displays an error message and returns to the main window.</td>
</tr>
<tr>
<td>During an attempt to connect a connected database with a different logical name</td>
<td>The system responds with a run-time error condition and you can continue. You can use the <strong>NO-ERROR</strong> modifier to suppress the error.</td>
</tr>
<tr>
<td>During an attempt to connect a connected database with the same logical name</td>
<td>The system responds with a warning and you can continue. You can use the <strong>NO-ERROR</strong> option to suppress the warning.</td>
</tr>
<tr>
<td>During an attempt to connect an unconnected database, when the logical name is already in use by a connected database</td>
<td>The system responds with a run-time error condition and you cannot connect to the second database.</td>
</tr>
<tr>
<td>During an attempt to connect to an Oracle database that uses a double-byte character set</td>
<td>The system responds with an error that the code page of the database and the <code>-cpinternal</code> value differ. You also receive messages about missing conversion tables.</td>
</tr>
</tbody>
</table>
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

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:

```
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)
```

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.
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.

   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.
This chapter presents step-by-step instructions for tasks associated with the DataServer. The tutorial describes:

- Oracle Utilities menu
- 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. Table 7–1 briefly describes the contents of the menu. The sections which follow will describe each item in detail.

Table 7–1: OpenEdge DataServer for Oracle Utilities menu

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create DataServer Schema...</td>
<td>Creates a schema image for an Oracle database</td>
</tr>
<tr>
<td>Update/Add Table Definitions...</td>
<td>Updates the schema image to reflect any changes you made to the Oracle data definitions</td>
</tr>
<tr>
<td>Verify Table Definition...</td>
<td>Verifies that the data definitions in the schema image match your Oracle data definitions</td>
</tr>
<tr>
<td>Edit Connection Information...</td>
<td>Changes connection information or the logical database name in a schema image</td>
</tr>
<tr>
<td>Change DataServer Schema Code Page...</td>
<td>Changes the code page in a schema image</td>
</tr>
<tr>
<td>Delete DataServer Schema...</td>
<td>Deletes a schema image</td>
</tr>
<tr>
<td>Run ORACLE SQL *Plus...</td>
<td>Modifies your Oracle database with SQL*PLUS commands</td>
</tr>
<tr>
<td>Schema Migration Tools</td>
<td>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</td>
</tr>
</tbody>
</table>

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 security” section on page 5–15 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.
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 the “Running Oracle” section on page 5–17. For information on starting the DataServer processes, see the “Starting the DataServer processes” section on page 5–17.

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:

   ![Create/Modify Database Record for DataServer Schema dialog box]

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 OpenEdge Getting Started: ABL Essentials.

   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 the “Code pages” section on page 2–3 in for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see OpenEdge Development: Internationalizing 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 the “Connecting a schema holder at startup” section on page 6–9 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.
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.

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 the “Defining the ROWID”
section on page 7–17 for instructions and the “ROWID function” section on page 2–41 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

In this exercise, you update a schema image to reflect any changes you might have made to the data definitions for your Oracle database.

The Update/Add Table Definitions utility allows you to:

- 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: 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:

![Select ORACLE Objects dialog box](image1)

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:

![Select Objects by Pattern Match dialog box](image2)

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.

Table 7–2 lists the differences that the utility detects.

Table 7–2: Verify utility report

<table>
<thead>
<tr>
<th>Database object</th>
<th>Difference</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Table</td>
<td>Foreign type</td>
<td>Severe</td>
</tr>
<tr>
<td>Table</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Table</td>
<td>Package name</td>
<td>Severe</td>
</tr>
<tr>
<td>Table</td>
<td>ROWID index</td>
<td>Retained</td>
</tr>
<tr>
<td>Index</td>
<td>Active</td>
<td>Minor</td>
</tr>
<tr>
<td>Index</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Index</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Index</td>
<td>Unique</td>
<td>Retained</td>
</tr>
<tr>
<td>Index field</td>
<td>Abbreviated</td>
<td>Minor</td>
</tr>
<tr>
<td>Index field</td>
<td>Ascending</td>
<td>Severe</td>
</tr>
<tr>
<td>Index field</td>
<td>Order</td>
<td>Severe</td>
</tr>
<tr>
<td>Field</td>
<td>Case-sensitivity</td>
<td>Retained</td>
</tr>
</tbody>
</table>
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.

### Table 7–2: Verify utility report

<table>
<thead>
<tr>
<th>Database object</th>
<th>Difference</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Decimals</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Extent</td>
<td>Severe</td>
</tr>
<tr>
<td>Field</td>
<td>Initial value</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Mandatory</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Order</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>OpenEdge data type</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>OpenEdge format</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Shadow-column Name</td>
<td>Severe</td>
</tr>
</tbody>
</table>

1. 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.

2. 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.
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 `<dbname>+.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:

   ![Create/Modify Database Record for DataServer Schema](image)

   - Logical Database Name: Oracle_2019
   - Server Type: ORACLE
   - Code Page: 1252
   - Collation: Basic
   - Oracle Version: 12c
   - Connection Parameters:

   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 Chapter 6, “Connecting the DataServer,” and OpenEdge Deployment: 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:

![Table Triggers dialog box]

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:

![Field Properties dialog box]

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 the “Data types” section on page 2–16.

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:

![ORACLE Specific Fields dialog box]

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:

   ![Message]

4. If the PROGRESS_RECID message did not appear, the ROWID Choices dialog box appears:

   ![Dialog Box]

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. See OpenEdge Development: Internationalizing 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:

![Dialog box](image)

2. Choose **Yes** to verify your selection. After the schema image is deleted, the Data Admin tool returns to the main window.
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 Table 7–3

Table 7–3: Oracle permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE SESSION</td>
<td>Database</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>Database</td>
</tr>
<tr>
<td>CREATE SEQUENCE</td>
<td>Database</td>
</tr>
<tr>
<td>SELECT</td>
<td>System objects:</td>
</tr>
<tr>
<td></td>
<td>sys.argument$</td>
</tr>
<tr>
<td></td>
<td>sys.col$</td>
</tr>
<tr>
<td></td>
<td>sys.com$</td>
</tr>
<tr>
<td></td>
<td>sys.con$</td>
</tr>
<tr>
<td></td>
<td>sys.dual</td>
</tr>
<tr>
<td></td>
<td>sys.icol$</td>
</tr>
<tr>
<td></td>
<td>sys.ind$</td>
</tr>
<tr>
<td></td>
<td>sys.link$</td>
</tr>
<tr>
<td></td>
<td>sys.obj$</td>
</tr>
<tr>
<td></td>
<td>sys.procedure$</td>
</tr>
<tr>
<td></td>
<td>sys.seq$</td>
</tr>
<tr>
<td></td>
<td>sys.syn$</td>
</tr>
<tr>
<td></td>
<td>sys.tab$</td>
</tr>
<tr>
<td></td>
<td>sys.ts$</td>
</tr>
<tr>
<td></td>
<td>sys.user$</td>
</tr>
<tr>
<td></td>
<td>sys.view$</td>
</tr>
<tr>
<td>CREATE SEQUENCE</td>
<td>Database</td>
</tr>
<tr>
<td>Enough quota</td>
<td>Default table space</td>
</tr>
</tbody>
</table>
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. Table 7–4 describes the changes to a database that result from making an OpenEdge database compatible with Oracle by selecting the Create RECID option when using the OpenEdge-to-Oracle utility.

Table 7–4: An OpenEdge DB-to-Oracle database conversion

<table>
<thead>
<tr>
<th>OpenEdge database</th>
<th>Oracle database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record ID</td>
<td>An additional NUMBER column named PROGRESS_RECID</td>
</tr>
<tr>
<td>Case-insensitive index</td>
<td>The UPPER(column-name) syntax is used by the index</td>
</tr>
<tr>
<td>Array</td>
<td>SQL</td>
</tr>
<tr>
<td>Sequence</td>
<td>A column for each array element named field##1,</td>
</tr>
<tr>
<td></td>
<td>field##2, etc.</td>
</tr>
<tr>
<td></td>
<td>A sequence with the same characteristics</td>
</tr>
<tr>
<td></td>
<td>(incrementation, beginning value, etc.)</td>
</tr>
</tbody>
</table>

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 7–4). 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. Table 7–5 lists how many columns each OpenEdge object requires on an Oracle table.

Table 7–5: Columns required by OpenEdge objects

<table>
<thead>
<tr>
<th>Database object</th>
<th>Number of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record ID</td>
<td>One per table</td>
</tr>
<tr>
<td>Array</td>
<td>One per array element</td>
</tr>
</tbody>
</table>
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 utility uses either a field’s format or width information (.width) value) 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, you can select Use Width Field to create a column that is wider than the format indicates. If a column generated by the utility is not wide enough to hold the data, the utility does not load the remainder of the data for that table into the Oracle database.

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 OpenEdge Data Management: Database Administration.

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.

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.
5. 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.
6. 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 Table 7–6.

<table>
<thead>
<tr>
<th>OpenEdge field</th>
<th>Oracle column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record ID</td>
<td>PROGRESS_RECID</td>
</tr>
<tr>
<td>Array with three elements (for example, MONTH)</td>
<td>field##1, field##2, field##3 (MONTH##1, MONTH##2, MONTH##3)</td>
</tr>
</tbody>
</table>

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 must match the code page that the Oracle database uses.

6. In Windows, from the Data Admin tool, choose DataServer→ORACLE Utilities→Schema Migration Tools→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 predict/ora/protoora.p
   ```
7. The following screen appears and prompts you for information:

![OpenEdge DB-to-Oracle Conversion dialog box]

Table 7–7 describes each of the elements on the OpenEdge to Oracle Conversion dialog box.

<table>
<thead>
<tr>
<th>Interface Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original OpenEdge Database</td>
<td>The source database name. Accept the default value.</td>
</tr>
<tr>
<td>Connect parameters for OpenEdge</td>
<td>The parameters for the connection to the source OpenEdge database, which is the current working database. Accept the default value.</td>
</tr>
<tr>
<td>Name of Schema holder Database</td>
<td>Enter the name for the schema holder. The utility creates the schema holder if it does not exist.</td>
</tr>
<tr>
<td>Logical name for Oracle Database</td>
<td>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.</td>
</tr>
<tr>
<td>What version of Oracle</td>
<td>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.</td>
</tr>
<tr>
<td>Oracle Owner’s Username</td>
<td>Enter the Oracle database owner’s name.</td>
</tr>
<tr>
<td>Interface Element</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Oracle User’s password</strong></td>
<td>Enter the owner’s password.</td>
</tr>
<tr>
<td><strong>Oracle connect parameters</strong></td>
<td>Enter additional connection parameters as needed.</td>
</tr>
<tr>
<td><strong>Code page for Schema Image</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Collation Name</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Maximum char length</strong></td>
<td>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 the “Handling character length during database migration” section on page 7–32 for details.</td>
</tr>
<tr>
<td><strong>Tablespace tables</strong></td>
<td>Enter the name of the Oracle tablespace where you want to store schema information. The default is the Oracle user’s default tablespace.</td>
</tr>
<tr>
<td><strong>Tablespace index</strong></td>
<td>Enter the name of the Oracle tablespace where you want to store index information. The default is the Oracle user’s default tablespace.</td>
</tr>
<tr>
<td><strong>Create RECID field</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Load SQL</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 7–7: OpenEdge DB to Oracle UI settings  

<table>
<thead>
<tr>
<th>Interface Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Move Data</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **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. |
| **Include Default** | Check this toggle box to include initial values in column definitions.                                                                                                                                 |
| **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. |
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 “OpenEdge DB-to-Oracle utility” section on page 7–22 and the “Handling character length during database migration” section on page 7–32.

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 Table 7–8.

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 “OpenEdge DB-to-Oracle utility” section on page 7–22 and the “Handling character length during database migration” section on page 7–32.

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 Table 7–8.
## Table 7-8: Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODBNAME</td>
<td>Specify the source OpenEdge database name.</td>
</tr>
<tr>
<td>PROCONPARMS</td>
<td>Specify parameters for the connection to the source OpenEdge database.</td>
</tr>
<tr>
<td>SHDBNAME</td>
<td>Specify the new schema-holder name.</td>
</tr>
<tr>
<td>ORADBNAME</td>
<td>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.</td>
</tr>
<tr>
<td>ORAVERSION</td>
<td>Specify the version of Oracle you are using. Possible values are 8, 9, or 10. The default is 8.</td>
</tr>
<tr>
<td>ORAUSERNAME</td>
<td>Specify the Oracle user’s name.</td>
</tr>
<tr>
<td>ORAPASSWORD</td>
<td>Specify the user’s password.</td>
</tr>
<tr>
<td>ORACONPARMS</td>
<td>Specify additional connection parameters for the schema holder.</td>
</tr>
<tr>
<td>ORACODEPAGE</td>
<td>Specify the OpenEdge name for the code page that the Oracle Database uses.</td>
</tr>
<tr>
<td>TABLEAREA</td>
<td>Enter the name of the Oracle tablespace where you want to store schema information. The default is the SYSTEM tablespace.</td>
</tr>
<tr>
<td>INDEXAREA</td>
<td>Enter the name of the Oracle tablespace where you want to store index information. The default is the SYSTEM tablespace.</td>
</tr>
<tr>
<td>COMPATIBLE</td>
<td>Specify YES to create an Oracle database that supports FIND PREV/LAST and the OpenEdge ROWID function. The default is YES.</td>
</tr>
<tr>
<td>SQLWIDTH</td>
<td>Specify YES to use the _width field to calculate column size instead of using the format field.</td>
</tr>
<tr>
<td>ORACOLLNAME</td>
<td>Enter the collation name.</td>
</tr>
<tr>
<td>CRTDEFAULT</td>
<td>Specify YES to include initial values in column definitions.</td>
</tr>
<tr>
<td>LOADSQL</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 predict/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 7–9.

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.

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.

Table 7–9 provides details on character length settings during migrations:

<table>
<thead>
<tr>
<th>Character semantics</th>
<th>Byte semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Unicode Types toggle box</td>
<td>National character set</td>
</tr>
<tr>
<td>Selected</td>
<td>UTF-8 N/A</td>
</tr>
<tr>
<td>Selected</td>
<td>UTF-16 N/A</td>
</tr>
<tr>
<td>Unselected</td>
<td>N/A Non-Unicode</td>
</tr>
<tr>
<td>Unselected</td>
<td>N/A UTF-8</td>
</tr>
<tr>
<td>Unselected</td>
<td>N/A AL32UTF8</td>
</tr>
</tbody>
</table>

¹. 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:

![Delta df to ORACLE Conversion dialog box](image-url)

- Delta DF File
- Schema Holder Database: Sparta2007
- Connect parameters for Schema Holder Database: Current Working database
- Logical name for ORACLE Database: 10m
- ORACLE Object Owner Name: 
- ORACLE tablespace for Tables: 
- ORACLE tablespace for Indexes: 
- Maximum char length: 
- Include Default: 
- Include Shadow Columns: 
- Include model types: 
- Expand d8 to d10: 
- OK

---

7–34
2. Provide the information as described in Table 7–10.

Table 7–10: Delta df to Oracle Conversion UI elements

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DF File</td>
<td>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.</td>
</tr>
<tr>
<td>Schema Holder Database</td>
<td>The name of the schema holder.</td>
</tr>
<tr>
<td>Connect parameters for Schema</td>
<td>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.</td>
</tr>
<tr>
<td>Logical name for Oracle database</td>
<td>Specify the Oracle database logical name, that is, the name by which you refer to the Oracle database in your application.</td>
</tr>
<tr>
<td>Oracle Object Owner Name</td>
<td>Enter the name of the owner.</td>
</tr>
<tr>
<td>Oracle tablespace for Tables</td>
<td>Enter the names of any tablespaces to be used here.</td>
</tr>
<tr>
<td>Oracle tablespace for Indexes</td>
<td>Enter the names of any tablespaces to be used for indexes here.</td>
</tr>
<tr>
<td>Maximum char length</td>
<td>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 the &quot;Handling character length during database migration&quot; section on page 7–32 for details.</td>
</tr>
<tr>
<td>Create RECID Field</td>
<td>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.</td>
</tr>
<tr>
<td>Include Default</td>
<td>Check this toggle box to include initial values in column definitions.</td>
</tr>
<tr>
<td>Create schema holder delta df</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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.

### 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 the “Adjusting field widths during migration” section on page 7–39.
- **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. Table 7–11 describes the Oracle equivalents of OpenEdge object types.

**Table 7–11: Oracle equivalents of OpenEdge objects**

<table>
<thead>
<tr>
<th>OpenEdge object</th>
<th>Oracle equivalent objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case-insensitive Index</td>
<td>The index definition uses the <code>UPPER</code> function.</td>
</tr>
<tr>
<td>Array</td>
<td>One column for each extent of the OpenEdge array. The columns are named <code>field-name##extent-number</code>. For example, an OpenEdge field called <code>monthly-amount</code> with an extent of 12 has 12 columns in Oracle with names such as <code>MONTHLY_AMOUNT##1</code> through <code>MONTHLY_AMOUNT##12</code>.</td>
</tr>
<tr>
<td>Table</td>
<td>If <strong>Create RECID Field</strong>, is selected, for any new table, a <code>PROGRESS_RECID</code> column is added. This indexed column provides a unique key on the Oracle table. A sequence named <code>table-name_SEQ</code> is also added. This sequence populates the <code>PROGRESS_RECID</code> column for each row in the Oracle table.</td>
</tr>
<tr>
<td>Deleted Field</td>
<td>The column is dropped from Oracle.</td>
</tr>
</tbody>
</table>
Not all OpenEdge objects can be converted to Oracle by this utility. Table 7–12 details restrictions on the update.

### Table 7–12: Database modifications not converted to Oracle

<table>
<thead>
<tr>
<th>Database object</th>
<th>Modification in OpenEdge</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>Starting value altered</td>
<td>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.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Applied to schema holder. ABL triggers are not converted to Oracle SQL.</td>
</tr>
<tr>
<td>Character field</td>
<td>Format altered</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 7–13 shows how the fields of an OpenEdge table convert to Oracle equivalents.

### Table 7–13: Sample object equivalents

<table>
<thead>
<tr>
<th>OpenEdge state table</th>
<th>Oracle STATE table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character field: State-Name</td>
<td>STATE_NAME</td>
</tr>
<tr>
<td>Array with 3 Extents: State-Fact</td>
<td>STATE_FACT##1, STATE_FACT##2, STATE_FACT##3</td>
</tr>
<tr>
<td>Default record identifier object</td>
<td>STATE##PROGRESS_RECID, STATE_SEQ</td>
</tr>
</tbody>
</table>

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

Before you migrate a Unicode UTF-8 OpenEdge database with Unicode data content, Progress Software recommends that you execute the Adjust Field Width utility from the Options menu of the Data Dictionary to size the character width of columns created by the migration utility in the foreign database. The width value for each field will be calculated by the number of characters and not bytes. This is only necessary if you choose Width under the For fields widths use options of the Incremental Schema Migration utility.

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.
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:

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original OpenEdge DB</td>
<td>Enter the name of the source OpenEdge database.</td>
</tr>
<tr>
<td>Schema Image</td>
<td>Enter the Oracle database logical name. The logical database name is the name of the schema image.</td>
</tr>
<tr>
<td>Files To Compare</td>
<td>Either leave the default <strong>all</strong> 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.</td>
</tr>
</tbody>
</table>
Appendix A addresses two upgrade issues that users may encounter, as described in the following sections:

- 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.
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:

   ```sql
   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:

   ```sql
   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:

   ```sql
   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:

   ```sql
   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:

   ```sql
   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:

   ```sql
   SELECT table-name_SEQ FROM sys.dual;
   ```

8. Connect to Oracle and use the Data Dictionary’s Oracle utilities to update the schema holder.
Environment Variables

Appendix B lists the Oracle, OpenEdge, and system environment variables that affect building and running the OpenEdge DataServer for Oracle. See *OpenEdge Getting Started: Installation and Configuration* for more information on environment variables.
Table B–1 lists the environment variables and describes how to set them.

**Table B–1: Environment variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>How to set it</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLC (Required)</td>
<td>The pathname of the directory where you installed OpenEdge.</td>
</tr>
<tr>
<td>DLCDB (Optional)</td>
<td>The pathname of the directory where your databases reside.</td>
</tr>
<tr>
<td>DSLOGDIR (Optional)</td>
<td>The pathname of the log file that OpenEdge uses to keep track of DataServer processes and error messages.</td>
</tr>
<tr>
<td>NLS_LANG (Optional)</td>
<td>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.</td>
</tr>
<tr>
<td>ORACLE_HOME (Required)</td>
<td>The pathname of the directory where you installed Oracle.</td>
</tr>
<tr>
<td>ORACLE_SID (Optional)</td>
<td>The identifier for the Oracle instance you are accessing.</td>
</tr>
<tr>
<td>ORASOPATHNAME (Optional)</td>
<td>The complete file specification of the Oracle client shared library.</td>
</tr>
<tr>
<td>ORASRV (Optional)</td>
<td>The pathname 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.</td>
</tr>
<tr>
<td>ORAVERSION (Optional)</td>
<td>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.</td>
</tr>
<tr>
<td>PROBRKR (Optional)</td>
<td>The pathname to the executable of the broker.</td>
</tr>
<tr>
<td>PROEXE (Optional)</td>
<td>The pathname 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.</td>
</tr>
<tr>
<td>TNS_ADMIN (Optional)</td>
<td>The pathname of the directory where listener.ora and tnsnames.ora configuration files are found for the server. Or for the client, where the tnames.ora file is found.</td>
</tr>
</tbody>
</table>

**Note:** If you change 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.
Appendix C 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.
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 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>
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.
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>
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 OCIStmtPrepare <3>
OCI call omru   <2>
OCI call OCIStmtPrepare <3>
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 OCIStmtPrepare <4>
OCI call omru   <2>
OCI call OCIStmtPrepare <4>
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.CUSTOMER 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 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 <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.CUSTOMER TO 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 uniquie_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 <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.

```sql
FOR EACH customer FIELDS (name custnum postalcode) NO-LOCK:
    DISPLAY customer.name customer.custnum customer.postalcode.
END.
```

```sql
OCI call OCIStmtPrepare <2>     sqlcrc = 4163
    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 = 40887
    SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID unique_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.

```sql
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>

OCI call OCIHandleFree <0>

```sql
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>

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.CUSTOMER TO 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 TO
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)
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,'YYYYMMDDHH24MISS'),
        TO_CHAR(PROMISE_DATE,'YYYYMMDDHH24MISS'),CARRIER,INSTRUCTIONS,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-DHH24MISS'),TO_CHAR(PROMISE_DATE,'YYYYMMDDHH24MISS'),CARRIER,INST
   RUCATIONS,PO,TERMS,SALES_REP,PROGRESS_RECID FROM DOCTEST.ORDER_  
   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)
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.

(1 of 2)
OCI call OCIStmtPrepare <2>  sqlcrc = 41633
SELEC /+/ 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
SELEC /+/ 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
SELEC /+/ 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
SELEC /+/ INDEX_ASC(T0 ORDER_##CUST_ORDER) /* PROGRESS_RECID un
ique_id_0,CUST_NUM,ORDER_DATE,SALES_REP FROM DOCTEST.ORDER_ TO 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
Building DataServer Executables

This appendix describes how to build DataServer executables for all possible configurations, as outlined in the following sections:

- 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 or 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 `_orarsrv` 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 (`_orarsrv`), 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.
Table D–1 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 D–1: Building DataServer processes**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Building and configuring</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>Local UNIX</td>
<td>Use a single executable on a UNIX client process that includes the Oracle DataServer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use this configuration for local access or if you want the DataServer to access the remote Oracle instance through Oracle Networking.</td>
</tr>
<tr>
<td>UNIX</td>
<td>Remote UNIX</td>
<td>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</td>
</tr>
<tr>
<td>UNIX</td>
<td>Remote Windows</td>
<td>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 Progress Explorer.</td>
</tr>
<tr>
<td>Windows</td>
<td>Remote UNIX</td>
<td>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</td>
</tr>
</tbody>
</table>
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:

```
------------------------------------------------------------
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 OpenEdge Getting Started: Installation and Configuration.

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.

To use your new executable once it is built, you must replace your default executable or set the PROEXE environment variable.
To replace your default client executable:

1. Make a backup copy of the original executable. For example:

   ```
cp $DLC/bin/_progres $DLC/bin/_progres.orig
   
   cp $DLC/oebuild/orarx $DLC/bin/_progres
   
   chmod u+s $DLC/bin/_progres
   ```

To identify 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 oebuildd/make directory under the root of your OpenEdge install.

To build 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:

   ```bash
   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:

   ```bash
   ${DLC}/oebuildd/_orasrv
   ```

5. Set the environment variable ORACLE_HOME to the top level directory of your Oracle installation. For example:

   ```bash
   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:

   ```bash
   unset LD_LIBRARY_PATH LIBPATH SHLIB_PATH
   ```

8. Run the script ${DLC}/oebuildd/make/build_orasrv.sh. If you did not set the IMAGE variable, the Oracle DataServer server executable is built as:

   ```bash
   ${DLC}/oebuildd/_orasrv
   ```

   otherwise the executable is built as specified by $IMAGE
To use your new executable once it is built, you must set the ORASRV environment variable.

To replace your default server executable:

1. Make a backup copy of the original executable, if one exists. For example:

   ```bash
   cp $DLC/bin/_orasrv $DLC/bin/_orasrv.orig
   ```

2. Copy the newly built executable into your installed bin directory. For example:

   ```bash
   cp $DLC/oebuild/_orasrv $DLC/bin/_orasrv
   ```

3. Change permissions on the newly created executable to include the Set UID bit. For example:

   ```bash
   chmod u+s $DLC/bin/_orasrv
   ```

4. Define the environment variable ORASRV to your executable. For example:

   ```bash
   ORASRV=$DLC/bin/_orasrv
   export ORASRV
   ```

To identify your new executable with an environment variable and without relocating:

1. Define the environment variable ORASRV. For example:

   ```bash
   ORASRV=$DLC/oebuild/_orasrv
   export ORASRV
   ```

2. Change permissions on the newly created executable to include the Set UID bit. For example:

   ```bash
   chmod u+s $DLC/oebuild/_orasrv
   ```

Once you create your executables, see Chapter 5, “Configuring the DataServer,” for instructions to configure and use the DataServer executables.
Appendix E 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 OpenEdge Getting Started: Installation and Configuration 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 OpenEdge Deployment: Startup Command and Parameter Reference for additional information about syntax and usage.
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

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
</table>

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 path name to a file that contains the property settings to be validated, for example `test.properties`. If a filename or path name 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 Progress 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. Progress 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:

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[UBroker]</code></td>
<td>Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.</td>
</tr>
<tr>
<td><code>[NameServer]</code></td>
<td>Defines default property settings for all instances of a NameServer.</td>
</tr>
<tr>
<td><code>[NameServer.product-instance-name]</code></td>
<td>Defines property settings for this instance of a NameServer. The <code>ubroker.properties</code> file can contain several of these entities, each with a unique <code>product-instance-name</code>.</td>
</tr>
</tbody>
</table>

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 Progress 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

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| 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**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>oraconfig [ [ [ -name DataServer-name ] [ -propfile path-to-properties-file ] [ -validate ] ] [ -help ] ]</td>
</tr>
</tbody>
</table>

**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. Progress 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 OpenEdge Getting Started: Installation and Configuration.
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

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>oraman</td>
</tr>
<tr>
<td>Windows</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>{ -name DataServer-name</td>
</tr>
<tr>
<td></td>
<td>{ -kill</td>
</tr>
<tr>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td>-host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>[ -port port-number ]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

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.

Note

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**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>probrkr -S service-name [ -H host-name ]</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

**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 *OpenEdge Deployment: 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 *OpenEdge Deployment: 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 *Chapter 5, “Configuring the DataServer,”* 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>db-name</td>
<td>proshut { db-name</td>
</tr>
</tbody>
</table>

Note

For a complete listing of PROSHUT parameters and their functions, see *OpenEdge Data Management: Database Administration*. 
## Index

### Numbers
-1 parameter 6–10

### A
- Abbreviated index 2–46
- Aggregates
  - views 2–11, 2–12
- Analyzing performance 6–17
- ARRAY-MESSAGE option 4–6
- Arrays 2–28, 7–23
  - joins 4–13
- Auto-connect
  - failure 6–21

### B
- BEGINS operator 2–46
- BFILE 2–17
- Bind variables 4–4
  - queries 4–6
- BIND-WHERE option 4–6
- Blanks
  - leading and trailing 2–10
- Broker
  - running multiple 5–4, 5–6, 6–3
  - starting in Windows 6–6
- Buffers
  - defining for stored procedures 3–20
- BY option
  - joins 4–13

### C
- -c startup parameter 6–13
- -cache startup parameter 6–14
- CACHE-SIZE option 4–6
- Caching
  - schema 1–6
- Case-insensitive indexes 2–10, 7–23
- CHARACTER data type 2–16
- Character sets 2–3
  - changing in schema holders 7–19
- Clear dynamic temp-tables
  - defined 3–27
- CLOB data type 2–27
- CLOSE QUERY statement 2–40
- CLOSE STORED-PROCEDURE statement
  - complete syntax reference 3–8
  - retrieving parameter values 3–17
- CLOSED STORED-PROCEDURE statement
  - complete syntax reference 3–10
  - description 3–3
Code pages 2–3
changing in schema holders 7–19
 collation table 2–5
 conversion table 2–5

Column name 7–24
Column width 7–24
COMPILE statement 2–45
CONNECT statement 6–8, 6–21
Connecting schema holders 6–8
Connection failures
 with ORACLE 6–21
Connection parameters 6–9
 network options 6–9
CONTAINS operator 2–46
COUNT-OF function 2–46
-cpinternal startup parameter 2–5, 5–9, 6–14, 7–26
-cpstream startup parameter 5–9, 6–14, 7–26
CREATE statement 2–31, 2–46
Creating a temp-table layout plan
 options 3–26
Creating a temp-table plan layout 3–25
Creating at temp-table plan layout
 dynamic temp-table in a prepared state 3–28
 dynamic temp-table in an unprepared state 3–27
Creating schema holders 5–15
CURRENT-VALUE function 2–46
CURRENT-VALUE statement 2–46
Cursor repositioning 2–39, 2–44
Cursors 2–39, 4–6, 4–8, 6–18
 defined 2–39
 stored procedures 3–18

D
Data definitions 1–6
Data Dictionary 7–14
Data output and retrieval options
 results retrieved from a database 3–14
 return codes 3–14
 values of output parameters you define when creating a procedure 3–14

Data Service 6–14
Data types 2–16
 changing 7–16
 CHARACTER 2–16
 CLOB 2–27
 DATE 2–19, 2–30
 DECIMAL 2–17
 display format 7–16
 INT64 2–16, 2–17
 INTEGER 2–18
 LOB 2–27
 LOGICAL 2–18
 LONG 2–17, 2–44
 NUMBER 2–17
 RAW 2–19
 stored procedures 3–4

Database conversion
 OpenEdge-to-ORACLE 7–23, 7–26
Database objects
 defined 2–2
Database Type (-dt) startup parameter 6–13
Databases
 naming conventions 7–25
 objects 5–21, 7–4
 OpenEdge objects 2–2
 schema 1–6
 triggers 2–13
DataServer (-Dsrv) startup parameter 6–15
qt_cache_size 2–44
stored procedures 3–5

DataServers
 architecture 1–3
 broker 5–6, 6–22
 building executables D–4
 compiling applications 2–45
 cursor repositioning 2–39
 internal logic 1–4
 log file 5–6, 5–7, 6–2, 6–6, B–2
 networks 1–10
 record creation 2–31
 starting a broker E–10
 starting UNIX client 6–7
 starting Windows client 6–7
 transactions 2–36
DATE data type 2–19, 2–30
-db startup parameter 6–10
DBA privileges 5–15, 5–16
DBE DataServer
 startup parameters 5–9, 6–14, 7–26
DBRESTRICTIONS function 2–45
Deadlock
  stored procedures 3–5
DEBUG EXTENDED option 4–7
DEBUG SQL option 4–7
DECIMAL data type 2–17
DEFINE BROWSE statement 2–43
DEFINE BUFFER statement 3–19
Defining a view to use as a buffer 3–20
Deleting schema holders 7–20
Deploying schema holders 5–22
Direct Connect 6–14
Display formats 7–16
Distributed databases 5–21, 7–5
  connecting 6–12
  synonyms 2–15
DLC environment variable B–2
DSLOGDIR environment variable 5–6, 5–7, 6–2, 6–6, B–2
-Dsrv startup parameter 2–44, 6–15
-dt startup parameter 6–13
Dynamic temp-table 3–9

E
Environment variables
  DLC B–2
  DSLOGDIR 5–6, 5–7, 6–2, 6–6, B–2
  ORACLE_HOME 5–6, 5–7, 5–17, 6–2, 6–6, B–2
  ORACLE_SID 5–6, 5–7, 5–17, 6–2, 6–6, 6–22, B–2
  ORASRV 5–7, 6–6, B–2
  ORAVERSION B–2
  PROBRKR 5–7, 6–6, B–2
  PROEXE B–2
  TNS_ADMIN B–2
  UNIX D–4
Error handling 2–37
Error messages
  incomplete 5–21, 7–6
  ORACLE 5–21, 7–6
  related to the LOAD-RESULT-INTO phrase 3–32
  troubleshooting 6–21
Errors
  NO-ERROR option 3–32
  EXCLUSIVE-LOCK option 2–35, 4–8, 4–13
External data types 2–16
  defined 2–16

F
Field extents. See also Arrays
Field lists 2–43
  updating records 4–12
Field properties
  modifying 7–16
FIELDS option 2–43
  SHARE-LOCK 2–47
FIND LAST statement 2–44
FIND PREV statement 2–44
FIND statement 2–8, 2–32
  QUERY option 2–47
FOR EACH statement 2–8
Formats 7–16, 7–24
Functions
  CURRENT-VALUE 2–46
  DBRESTRICTIONS 2–45
  MATCHES 2–46
    double-strings 2–47
  NEXT-VALUE 2–46
  PROC-HANDLE 3–21
  PROC-STATUS 3–17
  RECID 2–32, 2–41
  ROWID 2–41, 7–6
  ROWID data type 2–17
  SETUSERID 2–47
  USERID 2–47, 6–12

H
-H startup parameter 6–9
handles
  SESSION:TIME-SOURCE 2–47
Handling errors 3–32
HINT option 4–7
Hints 2–9, 4–7, 4–11
Host Name (-H) startup parameter 6–9
Index

I

Index Cursors (-c) startup parameter 6–13
Index Hint (-noindexhint) startup parameter 2–10, 4–7, 6–16
Index hints 2–9
Indexes 2–8
  abbreviated 2–46
  case-insensitive 2–10, 7–23
  INDEX-INFORMATION 2–10
  repositioning 2–10
INDEX-HINT option 4–7
INPUT option
  stored procedures 3–10
INPUT-OUTPUT option
  retrieving parameter values 3–17
  stored procedures 3–10
Instances 1–5
  defined 1–5
INTEGER data type 2–18
Internal Code Page (-cpinternal) startup parameter 2–5

J

JOIN-BY-SQLDB option 4–8, 4–13
Joins 2–11, 2–12

L

-ld startup parameter 6–13
Literals
  queries 4–6
Loading result sets into temp-tables 3–23
LOAD-RESULT-INTO phrase
  description 3–3, 3–8
LOB data type 2–27
Local machines
  defined 5–2
 Locks
  stored procedures 3–5
Log files 5–6, 5–7, 6–2, 6–6, B–2
LOGICAL data type 2–18
Logical Database Name (-ld) startup parameter 6–13
Logical database names 5–19, 6–13, 7–3
  schema holders 7–13
LONG column 2–44
LONG data type 2–17
Lookahead cursors 4–6, 4–8
LOOKAHEAD option 4–8

M

MATCHES function 2–46
  double-byte strings 2–47
mergeprop utility 1–13
Messages
  ORACLE 5–21, 7–6
Mixed arrays
  defined 3–26
Modifying schema holders 7–14
Multi-table views 2–12

N

Naming conventions 2–3, 7–25
National Language Support (NLS)
  DataServer environment 5–8
  stored procedure 2–48
Net 8 6–12
NEXT-VALUE function 2–46
NO-ARRAY-MESSAGE option 4–6
NO-BIND-WHERE option 4–6
NO-DEBUG option 4–7
NO-ERROR option 6–21
  description 3–9
NO-INDEX-HINT option 2–10, 4–7
  -noindexhint startup parameter 2–10, 4–7, 6–16
NO-JOIN-BY-SQLDB option 4–8, 4–13
  -nojoinbysqldb startup parameter 4–8, 4–13, 6–16
NO-LOOKAHEAD option 4–8
Non-unique table names 2–3
NOT NULL 2–29
NSMAN E–4
NUMBER data type 2–17
Number of Databases (-h) startup parameter 5–22, 7–2, 7–6

O
OCI. See also ORACLE Call Interface
OE Build utility D–4
OF option 2–8
OPEN QUERY statement 2–32, 2–47
OpenEdge
 CHARACTER data type 2–16
 Data Dictionary 7–14
data types 2–16
database objects 2–2
DECIMAL data type 2–17
INTEGER data type 2–18
LOGICAL data type 2–18
naming conventions 2–3, 7–25
ORACLE utilities 5–19
size limitations 7–23
triggers 2–13
OpenEdge DB-to-ORACLE utility 7–22, 7–26
batch mode 7–30
ORACLE
data types
 external 2–16
 internal 2–16
DATE data type 2–19
DBA privileges 5–15
distributed database 5–21, 6–12, 7–5
LONG data type 2–17
naming conventions 2–3, 7–25
 Net 8 6–12
 non-unique table names 2–3
 NOT NULL 2–29
 NUMBER data type 2–17
 password 2–47, 5–16, 6–10, 6–22
 permissions 5–15
 RAW data type 2–19
 record locking 2–35
 ROWID 2–17
 ROWID data type 2–17
 running 5–17
 schema objects 2–2
 size limitations 7–23
 SQL*Net 6–12
 triggers 2–13
user ID 2–47, 5–16, 6–10, 6–22
 views 2–11
 WHERE clause 2–47
ORACLE 8
 creating schema holder 5–20, 7–4
ORACLE Call Interface 1–5
ORACLE utilities 5–19
ORACLE_HOME environment variable 5–6, 5–7, 5–17, 6–2, 6–6, B–2
ORACLE_SID environment variable 5–6, 5–7, 5–17, 6–2, 6–6, 6–22, B–2
ORACONFIG E–6
ORAMAN E–8
ORASRV
 setting in ProControl 5–5
 ORASRV environment variable 5–7, 6–6, B–2
 ORAVERSION environment variable B–2
ORDERED-JOIN option 4–9
Ordering results
 USE-INDEX 2–9
OUTPUT option
 retrieving parameter values 3–17
 stored procedures 3–10
P
-P startup parameter 6–10
package
 definition 3–2
PARAM option
 retrieving parameter values 3–17
 stored procedures 3–10
PARAM phrase
 description 3–3, 3–9
Parameter files 6–10
Password (-P) startup parameter 6–10
Passwords 2–47, 5–16, 6–10, 6–22
Performance
 analyzing 6–17
Permissions 5–15
Physical Database Name (-db) startup parameter 6–10
Index

Physical database names 6–10
PRO*C 1–5
PRO/SQL utility 7–21
PROBRK command E–10
PROBRK environment variable 5–7, 6–6, B–2
Procedures
  compiling 2–45
PROC-HANDLE function 3–21
PROC-HANDLE phrase
description 3–3, 3–9
PROC-STATUS function 3–17
PROC-STATUS phrase
description 3–3, 3–9
PROEXE environment variable B–2
Progress Explorer
  Data Service 6–14
  Direct Connect 6–14
Progress Explorer Framework
  and the mergeprop utility 1–13
  and the Progress Explorer tool 1–12
  configuring DataServer applications using 1–12
Progress Explorer tool
definition 1–12
Progress_RECID column 2–44
PROLOAD environment variable D–4
PROSHUT command E–11
protoora utility 7–26

Q
qt_bind_where option 6–15
qt_cache_size option 6–15
qt_lookahead option 6–15
qt_no_bind_where option 6–15
qt_no_lookahead option 6–15
Queries
  records not visible 2–32, 2–47
  USING option 2–46
Query tuning
  startup parameters 6–16
QUERY-TUNING phrase 2–10, 4–5

R
RAW data type 2–19
r-code
  size 2–45
Read-only (-RO) startup parameter 6–10
RECID function 2–32, 2–41
  joins 4–13
  replacing with ROWID 2–42
  transactions 2–32
Record creation 2–31
Record ID 7–23
Record locking 2–34
RELEASE statement 2–32
Remote databases
  starting a DataServer broker E–10
Remote DataServer
  starting 6–3
Remote machines
  defined 5–2
Repositioning indexes 2–10
Result set
  defined 3–3
Results retrieved from a database 3–14
  defining a special view on ORACLE data source to use as a buffer for row results 3–14
  loading results into a temp-table 3–15
  using OpenEdge-supplied proc-text-buffer from row results 3–14
Retrieving output parameter values 3–17
Retrieving result sets using proc-text-buffer 3–19
Retrieving results with cursor arguments 3–18
Return codes 3–14, 3–17
Return status
  defined 3–9
REVERSE-FROM option 4–9
-RO startup parameter 6–10
ROWID data type 2–17
ROWID function 2–17, 2–41, 5–21, 7–6
joins 4–13
record creation 2–41
selecting support 7–17
ROWID support 3–34
additional ProDataSet support 3–39
characteristics 3–34
understanding ROWID implementation 3–34
using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO 3–35
RUN STORED-PROCEDURE statement complete syntax reference 3–7
description 3–3
retrieving parameter values 3–17
send-sql-statement option 2–12, 3–13
with LOAD-RESULT-INTO option 3–13
without LOAD-RESULT-INTO option 3–13
syntax 3–12
with LOAD-RESULT-INTO phrase 3–12
with LOAD-RESULT-INTO phrase syntax 3–12
with send-sql-statement option 3–12
without LOAD-RESULT-INTO phrase 3–11
without LOAD-RESULT-INTO phrase syntax 3–11
Run Stored-Procedure statement details 3–8

S
-S startup parameter 6–9
PROSHUT command E–11
SAVE CACHE statement 6–20
Schema 1–6
Schema Cache File (-cache) startup parameter 6–14
Schema caching 1–6, 6–19
Schema cachings defined 1–6
Schema holders 1–4, 1–6
connecting 6–8
creating 5–15
defined 1–4, 1–6
deleting 7–20
deploying 5–22
logical names 7–13
modifying 7–14
updating 7–7
verifying 7–9
version 7 to 9 A–2
version 8 to 9 A–2
Schema images defined 1–6
Schema objects defined 2–2
ORACLE 2–2
Schemas defined 1–6
SELECT . . . FOR UPDATE (SQL) 2–35
send-sql-statement option
accessing views 2–12
RUN STORED-PROCEDURE statement 3–13
Sequence generators 2–13
defined 2–13
Sequences 7–23
Server Join (-nojoinbysqldb) startup parameter 4–8, 4–13, 6–16
Service Name (-S) startup parameter 6–9
PROSHUT E–11
SESSION:TIME-SOURCE handle 2–47
SETUSERID function 2–47
SHARE-LOCK option
FIELDS 2–47
Shutdown commands E–11
Single-user Mode (-1) startup parameter 6–10
Size limitations 7–23
column name 7–24
column width 7–24
Spaces leading and trailing 2–10
.sql script generating for ORACLE 7–21
SQL statements 2–35
SELECT 2–35
SQL*DBA 5–16
SQL*Net 5–2, 6–12, D–3
Index

SQL-92
data types 2–27
sqlnet connection string 6–12
Stale cursors
  defined 2–39
Starting a DataServer broker E–10
Startup parameters
  Database Type (-dt) 6–13
  DataServer (-Dsrv) 6–15
  qt_cache_size 2–44
  Index Cursors (-c) 6–13
  Index Hint (-noindexhint) 2–10, 4–7, 6–16
  Internal Code Page (-cpinternal) 2–5, 5–9, 6–14, 7–26
  Logical Database Name (-ld) 6–13
  network options
    Host Name (-H) 6–9
    Service Name (-S) 6–9
  optional 6–13
  Password (-P) 6–10
  Physical Database Name (-db) 6–10
  query tuning 6–16
  Read-only (-RO) 6–10
  Schema Cache File (-cache) 6–14
  Server Join (-nojoinbysqldb) 4–8, 4–13, 6–16
  Single-user Mode (-1) 6–10
  Stream Code Page (-cpstream) 5–9, 6–14, 7–26
  User ID (-U) 6–10
Statements
  COMPILE 2–45
  CONNECT 6–8, 6–21
  CREATE 2–31, 2–46
  CURRENT-VALUE 2–46
  DEFINE BROWSE 2–43
  FIND 2–8, 2–32, 2–47
  FIND LAST 2–44
  FIND PREV 2–44
  FOR EACH 2–8
  INDEX-INFORMATION 2–10
  OPEN QUERY 2–32, 2–47
  RUN STORED-PROCEDURE 2–12, 3–7
  SAVE CACHE 6–20
  UPDATE 2–35
  VALIDATE 2–32
Static temp-table 3–9
Stored functions
  definition 3–2
Stored procedures
  and dataserver performance improvement 3–2
  basics 3–3
cursors 3–18
  data types 3–4
  defining buffers 3–20
  definition 3–2
  implementing and running points 3–4
  INPUT option 3–10
  INPUT-OUTPUT option 3–10
  NLS support 2–48
  OUTPUT option 3–10
  PARAM option 3–10
  returning results 3–18
  returning status values 3–10
Stream Code Page (-cpstream) startup parameter 5–9, 6–14, 7–26
Synonyms 2–15
  distributed databases 2–15
System-owned objects 5–21, 7–4
T
  Table properties
    modifying 7–14
  Table validation
    adding to schema holder 7–15
Temp-tables 3–15
  and ProDataSets functionality 3–25
  loading result sets into 3–23
  planning a layout for stored procedure result sets 3–24
  types that support result sets 3–9
Terminology 2–2
TNS_ADMIN environment variable B–2
Transactions 2–32, 2–36, 2–41
Triggers 2–13
  adding to schema holder 7–15
U
  -U startup parameter 6–10
ubroker.properties file
  hierarchy E–3
UNIX environment variable D–4
Unknown value (?) 2–29
UPDATE statement 2–35
Updating schema holders 7–7
Upgrading to the DataServer 1–16
USE-INDEX option 2–8, 2–9, 2–42
User ID (-U) startup parameter 6–10
User IDs 2–47, 5–16, 6–10, 6–22
USERID function 2–47, 6–12
USING option 2–46
  joins 4–13

V

VALIDATE statement 2–32
Validation
  expression 7–16
Values of output parameters you define
  when creating a procedure 3–14

Verifying schema holders 7–9
Views 2–11
  defined 2–11
  multi-table 2–12

W

WHERE clause
  literals 4–6
  ORACLE time option 2–47
Word indexing
  CONTAINS 2–46

Z

-znotrim startup parameter 2–10