

Progress DataServer for ORACLE Guide

© 2001 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

Progress, Progress Results, Provision and WebSpeed are registered trademarks of Progress Software Corporation in the United States and other countries. Apptivity, AppServer, ProVision Plus, SmartObjects, IntelliStream, and other Progress product names are trademarks of Progress Software Corporation.

SonicMQ is a trademark of Sonic Software Corporation in the United States and other countries.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997 and the IBM XML Parser for Java Edition.

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Any other trademarks and/or service marks contained herein are the property of their respective owners.

.

May 2001



Product Code: 4508
Item Number: 81076;9.1C

Contents

Preface . . . . . **xi**
Purpose xi
Audience xi
Organization of This Manual xi
Typographical Conventions xiii
Syntax Notation xiv
Progress Messages xvii
Other Useful Documentation xix
Getting Started xix
Development Tools xx
Reporting Tools xxi
4GL xxii
Database xxiii
DataServers xxiii
SQL-89/Open Access xxiii
SQL-92 xxiv
Deployment xxiv
WebSpeed xxv
Reference xxv
SQL-92 Reference xxvi

1. Introduction **1–1**
1.1 DataServer Components 1–3
1.1.1 The DataServer for ORACLE Logic 1–5
1.1.2 The Schema Holder 1–6
1.1.3 Security 1–7
1.1.4 DataServer Utilities 1–8
1.1.5 DataServer Demonstration Database 1–8
1.1.6 DataServer Configurations 1–9

1.1.7 Distributed DataServer Applications 1–14
1.1.8 Configuring Distributed DataServer Applications Using Progress Explorer 1–15
1.2 Software Requirements 1–16
1.2.1 ORACLE8 Support 1–16
1.3 Guidelines for Using the DataServer 1–17
1.3.1 Guide to Related Topics 1–19

2. Programming Considerations 2–1
2.1 Database Design Issues 2–2
2.1.1 ORACLE and Progress Objects and Terminology 2–2
2.1.2 Naming Conventions 2–3
2.1.3 Code Pages 2–4
2.1.4 Indexes 2–6
2.1.5 Case-insensitive Indexes 2–8
2.1.6 Index Repositioning 2–9
2.1.7 Initial Values 2–10
2.1.8 ORACLE Views 2–10
2.1.9 Database Triggers 2–12
2.1.10 Sequence Generator 2–12
2.1.11 ORACLE Synonyms 2–14
2.2 Data Types 2–15
2.3 Arrays 2–18
2.4 Unknown Value 2–19
2.4.1 Zero-length Character Strings 2–20
2.5 Record Creation 2–22
2.6 Record Locking 2–23
2.6.1 Monitoring Locks 2–25
2.7 Transactions 2–25
2.7.1 Two-phase Commit 2–25
2.8 Error Handling 2–26
2.9 Cursors 2–27
2.9.1 Cursor Repositioning 2–28
2.9.2 Stale Cursors 2–28
2.9.3 Allocating Cursors 2–29
2.10 Progress 4GL Issues 2–29
2.10.1 RECID Function 2–29
2.10.2 ROWID Function 2–29
2.10.3 DEFINE BROWSE Statement 2–31
2.10.4 Field Lists 2–31
2.10.5 FIND PREV/LAST Statements 2–33
2.10.6 FIND FIRST/LAST Statements 2–33
2.10.7 Compiling Progress Procedures 2–33
2.10.8 Unsupported 4GL Statements 2–34

2.11 ORACLE Stored Procedures 2–37
2.11.1 Running a Stored Procedure 2–40
2.11.2 Retrieving Return Codes 2–41
2.11.3 Retrieving Output Parameter Values 2–41
2.11.4 Retrieving Results with Cursor Arguments 2–42
2.11.5 Transaction Processing 2–43
2.11.6 Returning Errors. 2–43
2.11.7 National Language Support 2–45
2.12 Sending SQL Statements 2–45
2.12.1 Defining a Buffer 2–46
2.13 Enhancing DataServer Performance 2–48
2.13.1 SQL and Progress Queries 2–48
2.13.2 Bind Variables 2–49
2.13.3 Query Tuning 2–49
2.13.4 Caching Records 2–56
2.13.5 ORACLE Hints 2–57
2.13.6 Using Field Lists When Updating Records. 2–58
2.13.7 Join by SQL DB 2–58
2.13.8 Skipping Schema Verification 2–61
2.13.9 Writing Queries for Performance 2–62

3. Configuring the DataServer 3–1
3.1 DataServer Modules 3–2
3.2 Configuring NT Modules 3–4
3.2.1 Configuring the Local DataServer on NT 3–5
3.2.2 Configuring the DataServer on the NT Host 3–5
3.3 Configuring UNIX Modules 3–7
3.3.1 Configuring the Local DataServer on UNIX 3–8
3.3.2 Configuring the DataServer on the UNIX Host. 3–9
3.4 Configuring Windows Modules 3–10
3.5 Configuring an International Environment 3–11
3.6 Configuring the DataServer in the Explorer Administration Framework . 3–14
3.6.1 Configuring the Explorer Administration Framework on NT 3–14
3.6.2 Configuring the Explorer Administration Framework on UNIX . 3–15
3.7 Creating a Schema Holder 3–18
3.7.1 Schema-holder Security. 3–18
3.7.2 Running ORACLE 3–20
3.7.3 Starting the DataServer Processes 3–20
3.7.4 Creating an Empty Progress Database 3–21
3.7.5 Using the DataServer Utility to Create a Schema Holder 3–22
3.7.6 Maintaining a Schema Holder 3–26
3.7.7 Deploying a Schema Holder 3–27
3.7.8 Setting Up a Schema Holder As an NT Service. 3–28

4. Connecting the DataServer 4–1
4.1 Starting the Local DataServer 4–2
4.2 Starting the Remote DataServer 4–3
4.2.1 Starting the DataServer in the Explorer Administration Framework 4–3
4.2.2 Starting and Stopping the NT Broker Process 4–6
4.2.3 Starting and Stopping the UNIX Broker Process 4–6
4.2.4 Starting the UNIX Client Process 4–8
4.2.5 Starting the NT or Windows Client Process 4–8
4.3 Connecting a Schema Holder 4–9
4.3.1 Connecting a Schema Holder at Startup. 4–10
4.3.2 Connecting Through SQL*Net or Net 8. 4–13
4.3.3 Unsupported Connection Parameters 4–15
4.3.4 Optional Connection and Startup Parameters 4–15
4.3.5 Query Tuning with Connection and Startup Parameters. 4–16
4.3.6 Analyzing Performance 4–19
4.3.7 Index Cursors 4–21
4.3.8 Local Schema Caching 4–22
4.4 Failures and Progress Responses 4–23
4.4.1 Connection Troubleshooting 4–24
4.4.2 Accessing the DataServer Log 4–25

5. The DataServer Tutorial 5–1
5.1 ORACLE Demonstration Database 5–2
5.2 Updating a Schema Image 5–5
5.3 Verifying a Schema Image 5–7
5.4 Changing Logical Name and Connection Information 5–13
5.5 Modifying a Schema Image 5–14
5.5.1 Modifying Table-level Information 5–14
5.5.2 Modifying Field-level Information 5–15
5.5.3 Defining the ROWID 5–17
5.6 Changing a Code Page 5–19
5.7 Deleting a Schema Image 5–20
5.8 Migrating a Progress Database to ORACLE with PRO/SQL 5–20
5.9 The Progress-to-ORACLE Utility 5–21
5.9.1 Preparing a Database for the Utility 5–22
5.9.2 ORACLE Size Limitations 5–23
5.9.3 Naming Conventions. 5–25
5.9.4 ORACLE Tablespaces for Tables and Indexes 5–26
5.9.5 Unknown Value 5–27
5.9.6 Running the Progress-to-ORACLE Utility 5–27
5.10 Progress-to-ORACLE Incremental Schema Migration Utility 5–32
5.10.1 Updating the ORACLE Database 5–37
5.11 Adjust ORACLE Schema Image to Progress Database 5–37

A. Upgrading DataServer Applications A–1
A.1 Upgrading Schema Holders from Progress Version 7 or 8 . . . . . A–2
A.1.1 Using the Manual Upgrade Technique A–2
A.2 Upgrading from ORACLE7 to ORACLE8 . . . . . A–6

B. Stored Procedure Reference B–1
CLOSE STORED-PROCEDURE Statement B–2
PROC-HANDLE Function B–4
PROC-STATUS Function B–5
RUN STORED-PROCEDURE Statement B–6

C. Environment Variables C–1

D. Sample Queries D–1

E. Building DataServer Executables E–1
E.1 Overview E–2
E.2 PROBUILD and the DataServer E–4
E.2.1 PROBUILD Requirements E–4
E.3 Building Executables on the UNIX Client E–5
E.4 Building Executables on the UNIX Host E–7
E.5 Setting Environment Variables E–8
E.5.1 Linking DataServer Executables E–10
E.5.2 Setting ORALIB E–11
E.5.3 Rebuilding Libclnt.so E–14

F. DataServer Command Line Utilities and Startup Parameters F–1
F.1 Command Line Utilities for the DataServer F–2
F.1.1 NSCONFIG Utility F–2
F.1.2 NSMAN Utility F–5
F.1.3 ORACONFIG Utility F–7
F.1.4 ORAMAN Utility F–10
F.1.5 PROBRKR Command F–12
F.1.6 PROSHUT Command F–13
F.2 DataServer Startup Parameters for UNIX and Windows F–16

Index Index–1

Figures

Figure 1–1: DataServer Architecture 1–4
Figure 1–2: The DataServer for ORACLE Logic 1–5
Figure 1–3: The Schema-loading Process 1–7
Figure 1–4: The Local DataServer 1–10
Figure 1–5: A Local DataServer and Remote ORACLE Through SQL*Net 1–10
Figure 1–6: The DataServer for ORACLE-Remote 1–11
Figure 1–7: The DataServer for ORACLE-Windows Client to NT Server 1–12
Figure 1–8: Progress for Windows to ORACLE on UNIX 1–13
Figure 2–1: DataServer Processes and Code Pages 2–4

Tables

Table 1–1: Supported Configurations 1–13
Table 1–2: How to Use This Manual 1–18
Table 1–3: DataServer-related Topics 1–20
Table 2–1: ORACLE and Progress Objects 2–2
Table 2–2: Supported ORACLE Views 2–10
Table 2–3: Progress and ORACLE Sequence Generators 2–13
Table 2–4: Supported Synonyms 2–15
Table 2–5: ORACLE and Progress Data Types 2–16
Table 2–6: LOGICAL Data Type and ORACLE Equivalents 2–17
Table 2–7: UNKNOWN and NULL Values 2–19
Table 2–8: Progress and ORACLE Locks 2–23
Table 2–9: ORACLE Locking 2–24
Table 2–10: Progress and ORACLE Differences 2–35
Table 2–11: Argument Data Types 2–38
Table 2–12: Returning Values from Stored Procedures 2–39
Table 2–13: Query-tuning Options 2–51
Table 2–14: Controlling Join by SQLDB 2–60
Table 3–1: Installing DataServer Modules 3–2
Table 3–2: DataServer for ORACLE Environment Variables 3–6
Table 3–3: Environment Variables for the Local DataServer 3–8
Table 3–4: Environment Variables for the Remote DataServer 3–9
Table 3–5: Supported DBE Code Pages 3–12
Table 3–6: DataServer for ORACLE Sections 3–16
Table 3–7: Required ORACLE Permissions 3–19
Table 3–8: Database Operations and the Schema Holder 3–26
Table 4–1: Environment Variables for the Local DataServer 4–2
Table 4–2: Environment Variables for the Remote DataServer 4–7
Table 4–3: Remote DataServer Connection Parameters 4–10
Table 4–4: Required Connection Parameters 4–12
Table 4–5: Optional Connection and Startup Parameters 4–15
Table 4–6: Connection Query-tuning Options 4–17
Table 4–7: Query-tuning Startup Parameters 4–19
Table 4–8: Diagnostic Options 4–20
Table 4–9: Failure Responses 4–23
Table 5–1: Progress-to-ORACLE Utility 5–2
Table 5–2: DataServer for ORACLE Utilities Menu 5–4
Table 5–3: Verify Utility Report 5–7
Table 5–4: Permissions for the Progress-to-ORACLE Utility 5–21
Table 5–5: A Progress-to-ORACLE Database Conversion 5–22
Table 5–6: Columns Required by Progress Objects 5–23
Table 5–7: Progress-to-ORACLE Naming Conventions 5–25
Table 5–8: Unknown Value Mapping 5–27
Table 5–9: Progress-to-ORACLE Utility 5–28

Table 5–10: Progress-to-ORACLE Utility Batch Parameters 5–31
Table 5–11: Generate Delta.sql Progress-to-ORACLE Utility 5–34
Table 5–12: ORACLE Equivalents of Progress Objects 5–36
Table 5–13: Sample Object Equivalents 5–37
Table 5–14: Adjust Schema Utility Progress-to-ORACLE Utility 5–38
Table C–1: Environment Variables C–2
Table E–1: Building DataServer Processes E–3
Table E–2: Environment Variables for the Local DataServer E–8
Table E–3: Environment Variables for the Remote DataServer E–9
Table F–1: DataServer Parameters for UNIX and Windows F–16

Preface

Purpose

This manual explains how to use the Progress 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 the Progress and ORACLE database management systems.

Audience

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

Organization of This Manual

Chapter 1, "Introduction"

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

Chapter 2, "Programming Considerations"

Discusses the differences between ORACLE and Progress and how the DataServer resolves them.

Chapter 3, "Configuring the DataServer"

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

Chapter 4, “Connecting the DataServer”

Presents instructions for connecting the DataServer and a schema holder.

Chapter 5, “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 Progress-to-ORACLE migration utility.

Appendix A, “Upgrading DataServer Applications”

Provides information on the how to upgrade from an earlier version of the DataServer and from an earlier version of ORACLE.

Appendix B, “Stored Procedure Reference”

Describes the Progress 4GL statements and functions that support running ORACLE stored procedures.

Appendix C, “Environment Variables”

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

Appendix D, “Sample Queries”

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

Appendix E, “Building DataServer Executables”

Provides instructions for building DataServer executables using the PROBUILD utility.

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

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

Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
  - Commands or characters that the user types
  - That a word carries particular weight or emphasis
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

  END-ERROR, **GET**, **GO**
  ALT, **CTRL**, **SPACEBAR**, **TAB**

- When you have to press a combination of keys, they are joined by a dash. You press and hold down the first key, then press the second key.

  CTRL-X

- When you have to press and release one key, then press another key, the key names are separated with a space.

  ESCAPE H
  ESCAPE CURSOR-LEFT

Syntax Notation

The syntax for each component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in a procedure.

  In this example, ACCUM is a keyword:

  SYNTAX

  ```
  ACCUM aggregate expression
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the ACCUM function above, the *aggregate* and *expression* options are defined with the syntax for the ACCUM function in the *Progress Language Reference*.

- You must end all statements (except for DO, FOR, FUNCTION, PROCEDURE, and REPEAT) with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon, as in this example:

  ```
  FOR EACH Customer:
     DISPLAY Name.
  END.
  ```

- Square brackets (**[ ]**) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

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

  SYNTAX

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

  In some instances, square brackets are not a syntax notation, but part of the language.

For example, this syntax for the INITIAL option uses brackets to bound an initial value list for an array variable definition. In these cases, normal text brackets ([ ]) are used:

SYNTAX

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

NOTE: The ellipsis (**...**) indicates repetition, as shown in a following description.

- Braces (**{ }**) around an item indicate that the item, or a choice of one of the enclosed items, is required.

  In this example, you must specify the items BY and *expression* and can optionally specify the item DESCENDING, in that order:

  SYNTAX

  ```
  { BY expression [ DESCENDING ] }
  ```

  In some cases, braces are not a syntax notation, but part of the language.

  For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. In these cases, normal text braces ({ }) are used:

  SYNTAX

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

- A vertical bar (**|**) indicates a choice.

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

  SYNTAX

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

In this example, you must select one of *logical-name* or *alias*:

SYNTAX

```
CONNECTED ( { logical-name | alias } )
```

- Ellipses (**. . .**) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

  In this example, you must include two expressions, but you can optionally include more. Note that each subsequent expression must be preceded by a comma:

  SYNTAX

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

  In this example, you must specify MESSAGE, then at least one of *expression* or SKIP, but any additional number of *expression* or SKIP 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" ] ... }
  ```

- In some examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

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

SYNTAX

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

In this example, ASSIGN requires one of two choices: either one or more of *field*, or one of *record*. Other options available with either *field* or *record* are grouped with braces and brackets. The open and close braces indicate the required order of options:

SYNTAX

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

Progress Messages

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

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to 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 Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress 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 following execution messages.

- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

Use Progress online help to get more information about Progress messages. On the Windows platform, many Progress tools include the following Help menu options to provide information about messages:

- Choose Help→ Recent Messages to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose Help→ Messages, then enter the message number to display a description of any Progress message. (If you encounter an error that terminates Progress, make a note of the message number before restarting.)
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL-W**).

On the UNIX platform, you can use the Progress PRO command to start a single-user mode character Progress client session and view a brief description of a message by providing its number. Follow these steps:

1 ♦ Start the Progress Procedure Editor:

```
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.

Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows and Character platforms and are provided in electronic documentation format on CD-ROM.

Getting Started

Progress Electronic Documentation Installation and Configuration Guide (Hard copy only)

A booklet that describes how to install the Progress EDOC viewer and collection on UNIX and Windows.

Progress Installation and Configuration Guide Version 9 for UNIX

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

Progress Installation and Configuration Guide Version 9 for Windows

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

Progress Version 9 Product Update Bulletin

A guide that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

Progress Application Development Environment — Getting Started (Windows only)

A practical guide to graphical application development within the Progress Application Development Environment (ADE). This guide includes an overview of the ADE and its tools, an overview of Progress SmartObject technology, and tutorials and exercises that help you better understand SmartObject technology and how to use the ADE to develop applications.

Progress Language Tutorial for Windows and *Progress Language Tutorial for Character*

Platform-specific tutorials designed for new Progress users. The tutorials use a step-by-step approach to explore the Progress application development environment using the 4GL.

Progress Master Glossary for Windows and *Progress Master Glossary for Character* (EDOC only)

Platform-specific master glossaries for the Progress documentation set. These books are in electronic format only.

Progress Master Index and Glossary for Windows and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

Progress Startup Command and Parameter Reference

A reference manual that describes the Progress startup commands and parameters in alphabetical order.

Welcome to Progress (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

Development Tools

Progress ADM 2 Guide

A guide to using the Application Development Model, Version 2 (ADM 2) application architecture to develop Progress applications. It includes instructions for building and using Progress SmartObjects.

Progress ADM 2 Reference

A reference for the Application Development Model, Version 2 (ADM 2) application. It includes descriptions of ADM 2 functions and procedures.

Progress AppBuilder Developer's Guide (Windows only)

A programmer's guide to using the Progress AppBuilder visual layout editor. AppBuilder is a Rapid Application Development (RAD) tool that can significantly reduce the time and effort required to create Progress applications.

Progress Basic Database Tools (Character only; information for Windows is in online help)

A guide for the Progress Database Administration tools, such as the Data Dictionary.

Progress Basic Development Tools (Character only; information for Windows is in online help)

A guide for the Progress development toolset, including the Progress Procedure Editor and the Application Compiler.

Progress Debugger Guide

A guide for the Progress Application Debugger. The Debugger helps you trace and correct programming errors by allowing you to monitor and modify procedure execution as it happens.

Progress Help Development Guide (Windows only)

A guide that describes how to develop and integrate an online help system for a Progress application.

Progress Translation Manager Guide (Windows only)

A guide that describes how to use the Progress Translation Manager tool to manage the entire process of translating the text phrases in Progress applications.

Progress Visual Translator Guide (Windows only)

A guide that describes how to use the Progress Visual Translator tool to translate text phrases from procedures into one or more spoken languages.

Reporting Tools

Progress Report Builder Deployment Guide (Windows only)

An administration and development guide for generating Report Builder reports using the Progress Report Engine.

Progress Report Builder Tutorial (Windows only)

A tutorial that provides step-by-step instructions for creating eight sample Report Builder reports.

Progress Report Builder User's Guide (Windows only)

A guide for generating reports with the Progress Report Builder.

Progress Results Administration and Development Guide (Windows only)

A guide for system administrators that describes how to set up and maintain the Results product in a graphical environment. This guide also describes how to program, customize, and package Results with your own products. In addition, it describes how to convert character-based Results applications to graphical Results applications.

Progress Results User's Guide for Windows and *Progress Results User's Guide for UNIX*

Platform-specific guides for users with little or no programming experience that explain how to query, report, and update information with Results. Each guide also helps advanced users and application developers customize and integrate Results into their own applications.

4GL

Building Distributed Applications Using the Progress AppServer

A guide that provides comprehensive information about building and implementing distributed applications using the Progress AppServer. Topics include basic product information and terminology, design options and issues, setup and maintenance considerations, 4GL programming details, and remote debugging.

Progress External Program Interfaces

A guide to accessing non-Progress applications from Progress. This guide describes how to use system clipboards, UNIX named pipes, Windows dynamic link libraries, Windows dynamic data exchange, Windows ActiveX controls, and the Progress Host Language Call Interface to communicate with non-Progress applications and extend Progress functionality.

Progress Internationalization Guide

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

Progress Language Reference

A three-volume reference set that contains extensive descriptions and examples for each statement, phrase, function, operator, widget, attribute, method, and event in the Progress language.

Progress Programming Handbook

A two-volume handbook that details advanced Progress programming techniques.

Database

Progress Database Design Guide

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

Progress Database Administration Guide and Reference

This guide describes Progress database administration concepts and procedures. The procedures allow you to create and maintain your Progress databases and manage their performance.

DataServers

Progress DataServer Guides

These guides describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial. Each DataServer has its own guide, for example, the *Progress DataServer for ODBC Guide* or the *Progress/400 Product Guide*.

MERANT ODBC Branded Driver Reference

The Enterprise DataServer for ODBC includes MERANT ODBC drivers for all the supported data sources. For configuration information, see the MERANT documentation, which is available as a PDF file in *installation-path*\odbc. To read this file you must have the Adobe Acrobat Reader Version 3.1 or higher installed on your system. If you do not have the Adobe Acrobat Reader, you can download it from the Adobe Web site at: http://www.adobe.com/prodindex/acrobat/readstep.html.

SQL-89/Open Access

Progress Embedded SQL-89 Guide and Reference

A guide to Progress Embedded SQL-89 for C, including step-by-step instructions on building ESQL-89 applications and reference information on all Embedded SQL-89 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-89 and the ANSI standards upon which it is based.

Progress Open Client Developer's Guide

A guide that describes how to write and deploy Java and ActiveX applications that run as clients of the Progress AppServer. The guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server.

Progress SQL-89 Guide and Reference

A user guide and reference for programmers who use interactive Progress/SQL-89. It includes information on all supported SQL-89 statements, SQL-89 Data Manipulation Language components, SQL-89 Data Definition Language components, and supported Progress functions.

SQL-92

Progress Embedded SQL-92 Guide and Reference

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

Progress JDBC Driver Guide

A guide to the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface.

Progress ODBC Driver Guide

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

Progress SQL-92 Guide and Reference

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

Deployment

Progress Client Deployment Guide

A guide that describes the client deployment process and application administration concepts and procedures.

Progress Developer's Toolkit

A guide to using the Developer's Toolkit. This guide describes the advantages and disadvantages of different strategies for deploying Progress applications and explains how you can use the Toolkit to deploy applications with your selected strategy.

Progress Portability Guide

A guide that explains how to use the Progress toolset to build applications that are portable across all supported operating systems, user interfaces, and databases, following the Progress programming model.

WebSpeed

Getting Started with WebSpeed

Provides an introduction to the WebSpeed Workshop tools for creating Web applications. It introduces you to all the components of the WebSpeed Workshop and takes you through the process of creating your own Intranet application.

WebSpeed Installation and Configuration Guide

Provides instructions for installing WebSpeed on Windows and UNIX systems. It also discusses designing WebSpeed environments, configuring WebSpeed Brokers, WebSpeed Agents, and the NameServer, and connecting to a variety of data sources.

WebSpeed Developer's Guide

Provides a complete overview of WebSpeed and the guidance necessary to develop and deploy WebSpeed applications on the Web.

WebSpeed Product Update Bulletin

A booklet that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

Welcome to WebSpeed! (Hard copy only)

A booklet that explains how WebSpeed software and media are packaged. *Welcome to WebSpeed!* also provides descriptions of the various services Progress Software Corporation offers.

Reference

Pocket Progress (Hard copy only)

A reference that lets you quickly look up information about the Progress language or programming environment.

Pocket WebSpeed (Hard copy only)

A reference that lets you quickly look up information about the SpeedScript language or the WebSpeed programming environment.

SQL-92 Reference

(These are non-Progress resources available from your technical bookseller.)

A Guide to the SQL Standard

Date, C.J., with Hugh Darwen. 1997. Reading, MA: Addison Wesley.

Understanding the New SQL: A Complete Guide

Melton, Jim (Digital Equipment Corporation) and Alan R. Simon. 1993. San Francisco: Morgan Kaufmann Publishers.

1

Introduction

The DataServer for ORACLE allows you to access your ORACLE database with the Progress Fourth Generation Language (4GL) and develop applications within the Progress Application Development Environment (ADE). The ADE is a set of tools that helps you maintain databases and develop applications with graphical user interfaces. In addition to providing you with the ADE tools, the DataServer allows you to implement the Progress Version 8 database features and Progress 4GL expansions in applications that run with ORACLE. Some of these tools and features are:

- **The AppBuilder** — Use the AppBuilder to design and generate code for your graphical user interfaces. For example, you can use the AppBuilder to create a browser for viewing data stored in your ORACLE database.
- **The Progress event-driven programming model** — Use this programming model to develop applications that respond to choices that your end users make.
- **The Data Dictionary** — Use the Data Dictionary to modify database schema, create indexes, and define database triggers, validation expressions, and help messages.
- **Database triggers** — Use database triggers to fire a block of Progress 4GL code whenever a specific database event occurs, such as creating a record, deleting a record, or assigning a value to a field.

The DataServer for ORACLE does not support Progress word indexing.

The DataServer for ORACLE runs with ORACLE Version 7.3 and higher and Version 8.

This chapter presents an overview of the DataServer for ORACLE. It describes how Progress applications and databases work together with ORACLE through the DataServer. Specifically, it discusses DataServer architecture and guidelines for using the DataServer.

1.1 DataServer Components

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

Adding the DataServer to the standard Progress architecture allows the Progress client to access data managed by non-Progress data managers. The Progress 4GL 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 Progress 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 Progress architecture—the Progress client, the DataServer, the schema holder, and the DataServer utilities. These components work together to create and support a software module that allows Progress applications to access an ORACLE database.

The DataServer runs in a variety of configurations that can be local or include Progress or ORACLE networking. Depending on its configuration, a DataServer is a single, Progress executable or a set of Progress executables that you can distribute across operating systems. The "DataServer Configurations" section describes the configuration options.

Figure 1–1 illustrates the DataServer modules as they appear in a networked, or remote, configuration.

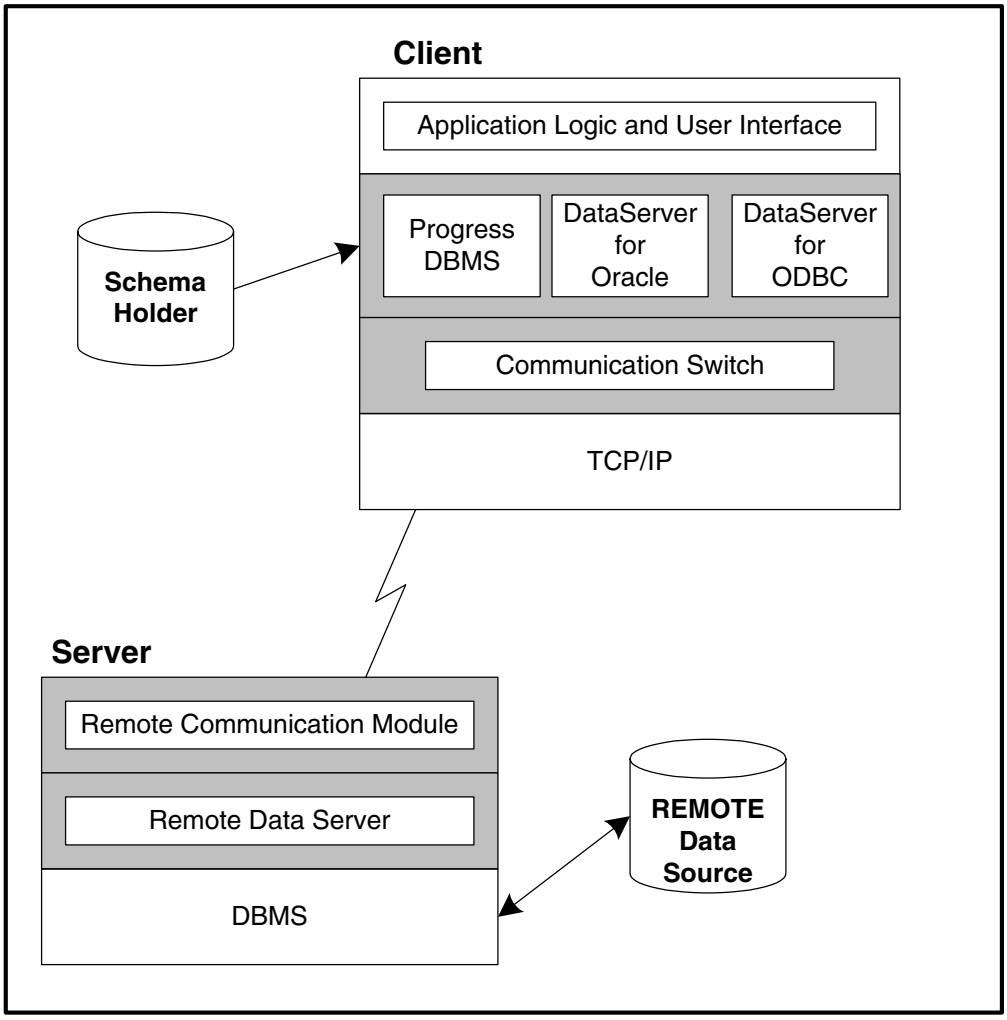


Figure 1–1: DataServer Architecture

1.1.1 The DataServer for ORACLE Logic

The DataServer places Progress 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-Progress databases. When you issue 4GL statements, they are compiled into calls to the ORACLE database. When the Progress client executes 4GL statements and retrieves information from the ORACLE database, it relies on data definitions maintained by the data dictionary in the Progress schema holder. Figure 1–2 shows the internal logic of the DataServer for ORACLE.

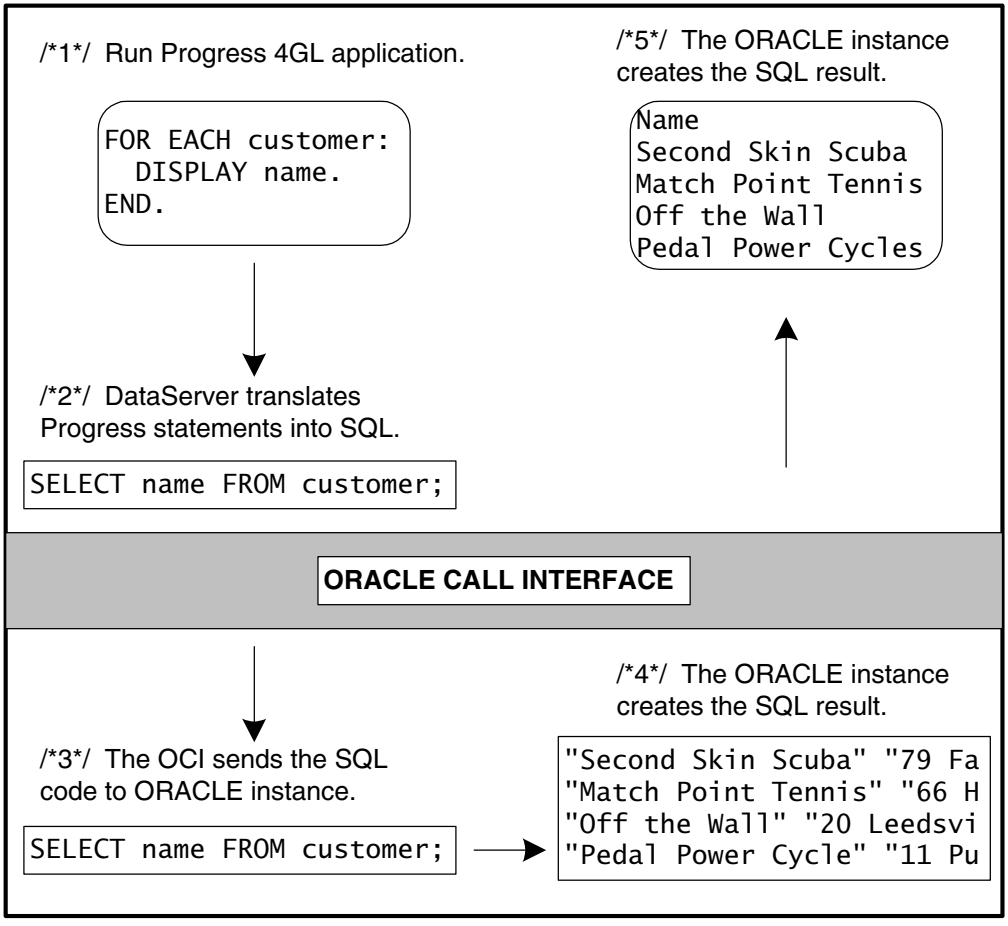


Figure 1–2: The DataServer for ORACLE Logic

The DataServer uses the ORACLE PRO*C OCI (ORACLE Call Interface) 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 a Progress application that accesses an ORACLE database, the Compiler translates the 4GL statements in the Progress 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 Progress through the OCI.

You can also send SQL statements directly to ORACLE. The DataServer passes SELECT SQL statements that you reference in a 4GL application directly to ORACLE without translating them. The DataServer also allows you to issue PL/SQL statements from a Progress application. See the "Sending SQL Statements" section in Chapter 2, "Programming Considerations," for more information.

1.1.2 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 Progress DataServer architecture. The *schema* of a database is a description of its structure, the tables, the fields within the tables, and the indexes. That information in the schema holder is called the *schema image*. When you use a Progress client with an ORACLE database, the DataServer uses the schema image to translate Progress 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 a Progress DataServer application, which lets you develop and compile applications for an ORACLE database without being connected to it.

Progress 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 a Progress 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 Progress database features, such as validation expressions or messages.

Figure 1–3 illustrates the schema-loading process.

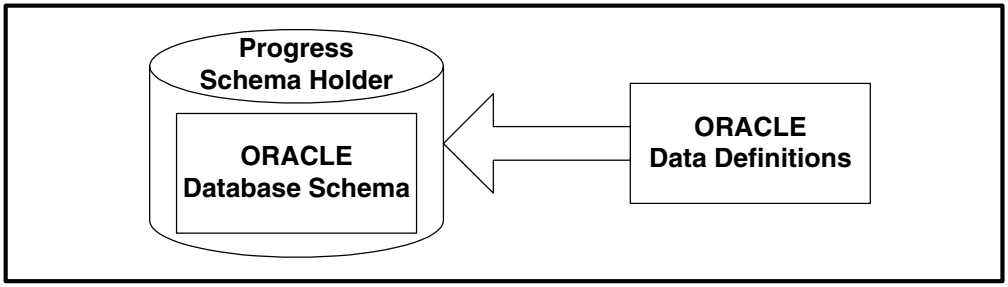


Figure 1–3: The Schema-loading Process

1.1.3 Security

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

Progress Security

The Progress database management system has no minimum security requirements. You can, however, impose security features on any Progress database or schema holder. There are four levels of application security you can impose: database-connection security, schema security, compile-time security, and run-time security. See the *Progress Programming Handbook* for more information.

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 over 60 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 DataServer for ORACLE, there are three security requirements:

- To create the schema holder for the ORACLE database, you must have SELECT privileges for system objects in the ORACLE database. The privileges required to access the schema holder and connect to the ORACLE database with the DataServer depend on whether and how your application modifies the 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. Progress provides connection parameters that the DataServer uses to pass this information to the ORACLE Database Manager.

The "Schema-holder Security" section in Chapter 3, "Configuring the DataServer," describes the required privileges in detail.

1.1.4 DataServer Utilities

Progress 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 Progress database to ORACLE
- Creating an incremental schema description of an ORACLE database

Chapter 5, "The DataServer Tutorial," describes how to use these utilities.

1.1.5 DataServer Demonstration Database

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

Chapter 5, "The DataServer Tutorial," describes how to create the Sports demonstration database, which you can use to run sample code or when learning how to work with the DataServer utilities.

1.1.6 DataServer Configurations

The Progress DataServer supports many possible configurations. There are two general types of configuration—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 SQL*Net or Net 8.

- **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 Progress 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.

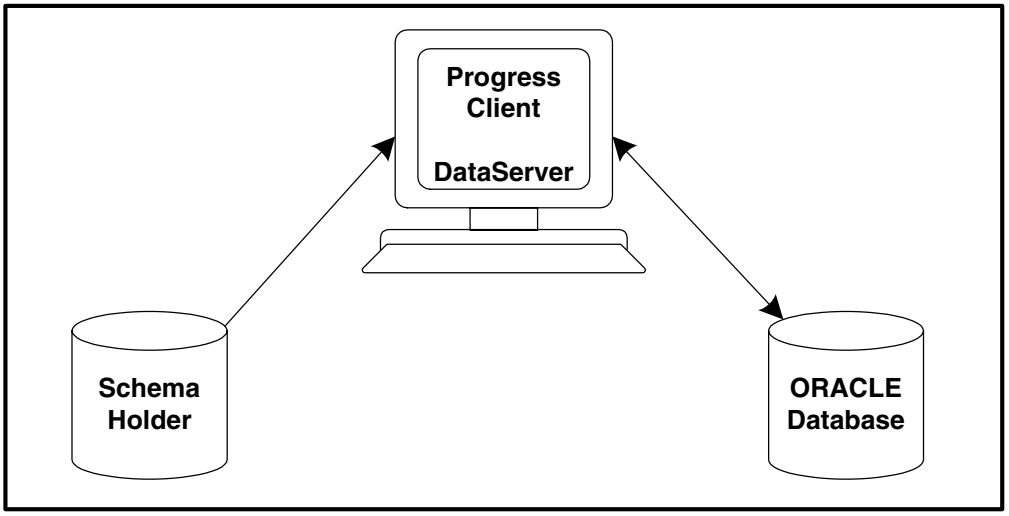


Figure 1–4: The Local DataServer

Figure 1–5 shows a local DataServer accessing an ORACLE database on another machine through SQL*Net. To access an ORACLE8 database, this configuration can use SQL*Net or Net 8. You must install SQL*Net or Net 8 on the client machine.

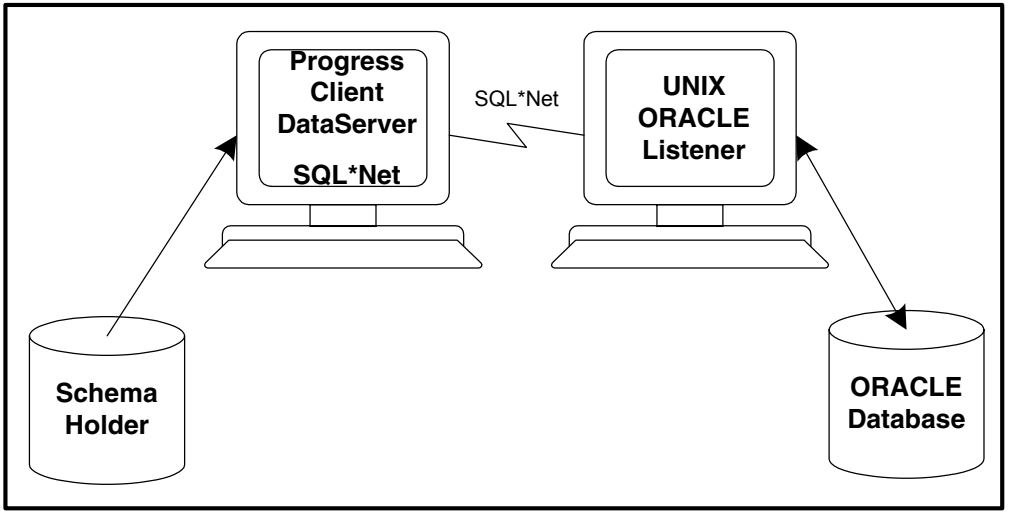


Figure 1–5: A Local DataServer and Remote ORACLE Through SQL*Net

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 Progress DataServer are running on the same machine.

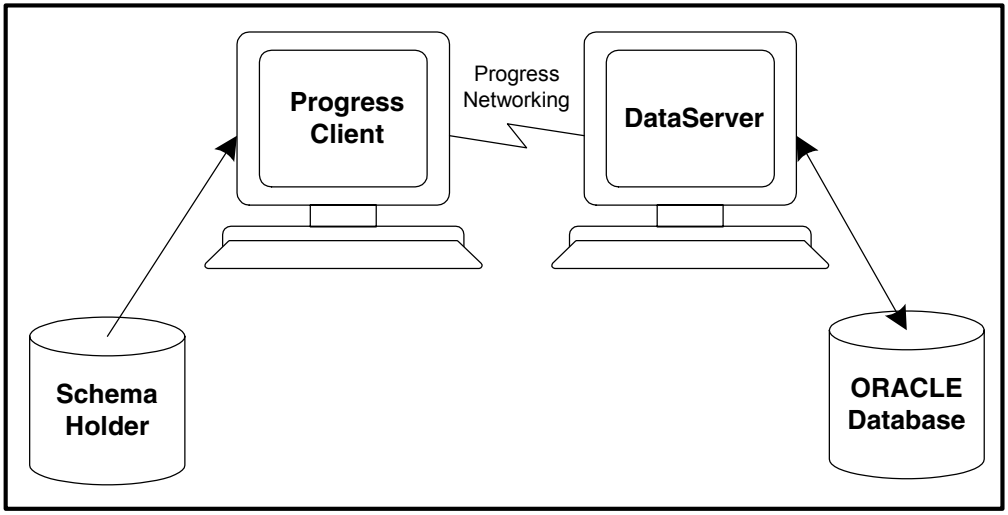


Figure 1–6: The DataServer for ORACLE-Remote

Figure 1–7 shows how you can combine clients and DataServers that run on different platforms. From your PC you can run a Progress client that accesses a remote DataServer on an NT workstation. This sample configuration has the schema holder on the PC client. You can optionally locate the schema holder on any host that is accessible on your Microsoft network.

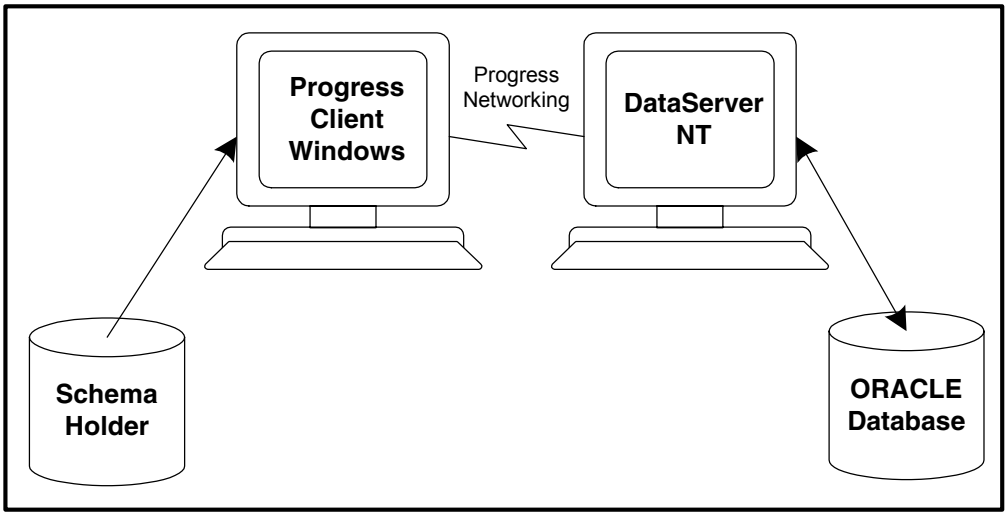


Figure 1–7: The DataServer for ORACLE-Windows Client to NT Server

Progress 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. Figure 1–8 shows a local self-serving client on a PC accessing an ORACLE database through SQL*Net. To access an ORACLE8 database, the same configuration can use Net 8 or SQL*Net.

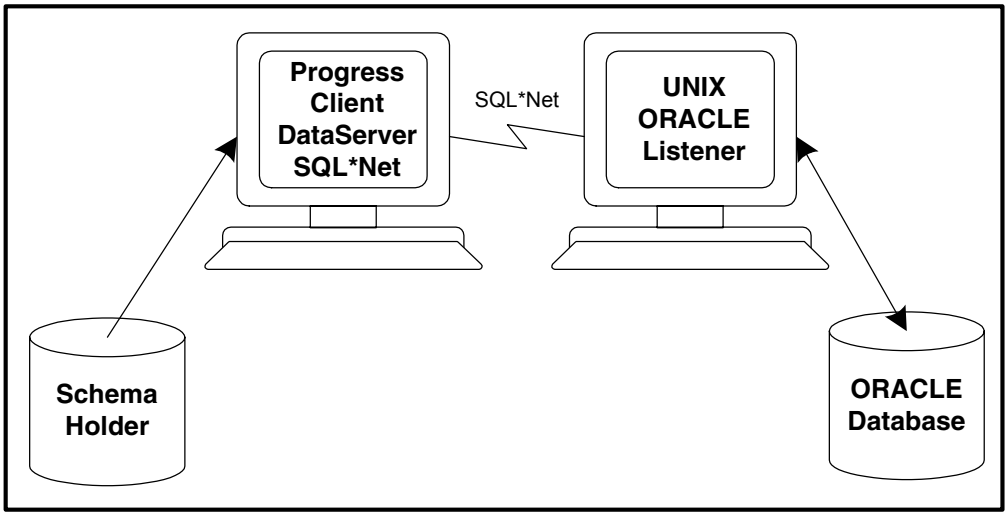


Figure 1–8: Progress for Windows to ORACLE on UNIX

NOTE: Currently, it is not possible to use dblinks in an ORACLE7 database to point to an ORACLE8 database, nor is it possible to use dblinks in an ORACLE8 database to point to an ORACLE7 database.

In configurations that include SQL*Net or Net 8 there are no Progress processes running on the machine where ORACLE is running. You must install SQL*Net or Net 8 on the client machine.

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 *(1 of 2)*

Client	Networking	DataServer
Progress on UNIX	None	DataServer, client, and ORACLE on the same machine.
Progress on UNIX	SQL*Net or Net 8	DataServer, client, and SQL*Net or Net 8 on the client machine, and ORACLE on a server.

Table 1–1: Supported Configurations *(2 of 2)*

Client	Networking	DataServer
Progress on UNIX	Progress	Client on the client machine, and the DataServer and ORACLE on a server.
Progress on Windows	SQL*Net or Net 8	DataServer, client, and SQL*Net or Net 8 on the client machine, and ORACLE on a server.
Progress on Windows	Progress	Client on the client machine, and the DataServer and ORACLE on a server.

1.1.7 Distributed DataServer Applications

Progress has integrated much of its server technology into a single framework, which allows you to administer distributed components as a single system. The Progress products that are dependent on this single administrative system, known as the Progress Explorer administration framework, are the Progress AppServer and the WebSpeed Transaction Server. Both of these products can support DataServers that allow you to include a non-Progress data source into this distributed system.

AppServer applications can use the DataServer to access data from a non-Progress data source. WebSpeed Agents can connect to a non-Progress 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 AdminServer) 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. On NT, 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 *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX* for an overview of the Explorer administration framework.

1.1.8 Configuring Distributed DataServer Applications Using Progress Explorer

The Progress Explorer tool is a graphical user interface that provides an easy way for you to manage Progress servers. Progress Explorer is one of the primary components of the Progress administration framework.

The Progress Explorer tool runs as a Windows client and works with another administration component, the Progress AdminServer, to provide a client/server framework for managing the following Progress servers:

- AppServer
- Database
- DataServer for ODBC
- DataServer for ORACLE
- NameServer
- WebSpeed Messenger
- WebSpeed Server

Using the Progress Explorer tool, you can:

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

For more information about working with the Progress Explorer tool, see the Progress Explorer online help.

See the "Configuring the DataServer in the Explorer Administration Framework" section in Chapter 3, "Configuring the DataServer," and the "Starting the DataServer in the Explorer Administration Framework" section in Chapter 4, "Connecting the DataServer," for more information.

1.2 Software Requirements

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

- Progress installation that includes the DataServer for ORACLE
- ORACLE Version 7.3 or higher, ORACLE8 DBMS, or Personal ORACLE
- Net 8 on the client machine if you want to connect to an ORACLE8 database using ORACLE networking
- SQL*Net on the client machine if you want to connect to an ORACLE7 database using ORACLE networking

Building and linking customized DataServer executables requires this additional software:

- ORACLE PRO*C for linking ORACLE libraries to executables
- C compiler recommended by ORACLE for the platform on which you are running ORACLE for linking executables
- Linker for building executables

1.2.1 ORACLE8 Support

The DataServer for ORACLE allows Progress and WebSpeed applications to access ORACLE8. ORACLE8 removes the restrictions of table, index, and column size that made migrating some Progress databases to ORACLE difficult. The DataServer does not support some ORACLE8 features because of their incompatibility with Progress, such as:

- Data clusters
- Large object data types (CLOBS and BLOBS)
- Object data types and methods

You cannot use the DataServer to access tables that include any of the new object data types.

Accessing ORACLE8 is similar to accessing ORACLE7 from the point of view of setting up the DataServer's components. The same permissions are required on the same system tables. The only difference is that you must specify which version of ORACLE you are pulling schema information from when you create your schema holder or migrate your Progress database to ORACLE. See the "Creating a Schema Holder" section in Chapter 3, "Configuring the DataServer," for more information.

1.3 Guidelines for Using the DataServer

Progress 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 Progress and ORACLE databases.

The DataServer allows you to use Progress features as extensions to ORACLE. Some of the Progress 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, "Programming Considerations."

When you create an ORACLE database from an existing Progress database with the Progress-to-ORACLE migration utility if you select the Create Extended 4GL Objects option, you can use the FIND PREV/LAST statements, arrays, and case-insensitive indexes with the resulting ORACLE database, in addition to taking advantage of Progress-like cursor behavior.

How you use the DataServer depends on whether you plan to access information in an ORACLE database through a Progress or WebSpeed application, or whether you plan to migrate a Progress 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 your *WebSpeed Installation and Configuration Guide*.

Using the DataServer for ORACLE for the First Time

These are the preparations you have to make before using the DataServer.

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

Chapter 3, "Configuring the DataServer," provides information about where to install DataServer modules and creating a schema holder.

Using the DataServer to Migrate a Database from Progress to ORACLE

These are the steps you follow when setting up and using the DataServer:

1. Install the DataServer modules on the machines your configuration requires.
2. Run the Progress-to-ORACLE migration utility.

The "The Progress-to-ORACLE Utility" section in Chapter 5, "The DataServer Tutorial," provides specific instructions.

Upgrading to the Progress Version 9 DataServer for ORACLE

These are the preparations you have to make before using the DataServer if you want to take advantage of Progress Version 9 features:

1. Install the Version 9 DataServer modules on the machines your configuration requires.
2. Create a schema holder.
3. Upgrade Version 7 or 8 schema holders to Version 9 by dumping data definitions and loading them into the new schema holder.

See Chapter 3, "Configuring the DataServer,"for information about where to install DataServer modules and creating a schema holder. See Appendix A, "Upgrading DataServer Applications," for more information on upgrading your DataServer.

Table 1–2 suggests paths through this manual that accommodate different approaches to using the DataServer for ORACLE.

Table 1–2: How to Use This Manual

If You Are . . .	Read This . . .
New to the DataServer for ORACLE	Chapter 2, "Programming Considerations" Chapter 3, "Configuring the DataServer" Chapter 4, "Connecting the DataServer" Chapter 5, "The DataServer Tutorial"
Migrating a Progress database to ORACLE	"The Progress-to-ORACLE Utility" section in Chapter 5, "The DataServer Tutorial" Chapter 2, "Programming Considerations" Chapter 3, "Configuring the DataServer" Chapter 4, "Connecting the DataServer" Chapter 5, "The DataServer Tutorial"
Upgrading to the DataServer to Progress Version 9	Chapter 2, "Programming Considerations" Chapter 3, "Configuring the DataServer" Appendix A, "Upgrading DataServer Applications" Chapter 5, "The DataServer Tutorial"
Upgrading from ORACLE7 to ORACLE8	Appendix A, "Upgrading DataServer Applications" Chapter 3, "Configuring the DataServer"

1.3.1 Guide to Related Topics

These manuals in the Progress documentation set contain useful information on different aspects of using a DataServer:

- The *Progress Database Design Guide* explains how to design a Progress database and provides good relational database design guidelines.
- The *Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* and the *Progress Programming Handbook* provide information on how to develop applications in the Progress 4GL.
- The *Progress Application Development Environment — Getting Started* describes how to develop an application with the Progress AppBuilder.

If you have never used Progress Version 9 and need an introduction, start with the *Progress Application Development Environment — Getting Started* manual or the *Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character*.

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

Topic	Progress Manuals
Starting up Progress	*Progress Installation and Configuration Guide Version 9 for UNIX* *Progress Installation and Configuration Guide Version 9 for Windows* *Progress Database Administration Guide and Reference*
Using the Progress Data Dictionary	*Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character*
Defining security for a Progress database	*Progress Database Administration Guide and Reference*
Writing applications in the Progress 4GL	*Progress Application Development Environment — Getting Started* *Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* *Progress Programming Handbook*
Upgrading applications from Progress Version 8.3	*Progress Version 9 Product Update Bulletin*

2

Programming Considerations

The Progress ADE used in conjunction with the DataServer allows you to develop applications that transparently access data from multiple sources. The DataServer for ORACLE achieves database transparency for applications running against Progress and ORACLE, but some of the ways it accomplishes this might affect how you develop applications. This chapter discusses the differences between Progress 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 and differences between ORACLE and Progress in the following areas:

- Database design issues, including database objects, naming conventions, character sets, indexes, views, database triggers, and sequence generators
- Application development issues including, data types, arrays, record creation, record locking, transactions, error handling, and cursor repositioning
- Progress 4GL issues, including the ROWID function, the FIND statement, and the COMPILE statement

Mapping Progress to ORACLE

The material in this chapter is also of interest to users who plan to migrate a Progress database to ORACLE. However, this migration raises several additional issues that are discussed in the "The Progress-to-ORACLE Utility" section in Chapter 5, "The DataServer Tutorial."

2.1 Database Design Issues

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

2.1.1 ORACLE and Progress Objects and Terminology

ORACLE and Progress 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 Progress. *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 Progress Objects *(1 of 2)*

ORACLE Schema Object	Progress Equivalent
Table	Table
Column	Field
Row	Record
Index	Index
Sequences	Sequences
Integrity constraint[1]	Validation expression
No equivalent	Validation message
Trigger	Trigger
Stored procedures	Stored procedures (SQL Engine)
View	View (SQL Engine)
Clusters	No equivalent
ORACLE8 Object methods	No equivalent

Table 2–1: ORACLE and Progress Objects *(2 of 2)*

ORACLE Schema Object	Progress Equivalent
Synonyms	Synonyms (SQL Engine)

[1] The ORACLE Integrity constraint enforces data integrity and business rules. However, unlike the Progress 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.

2.1.2 Naming Conventions

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

Field and Column Names

These are the restrictions for naming Progress fields and ORACLE columns:

- **Progress** — 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 Progress names, except % and -.

The set of characters that Progress allows includes all of the characters ORACLE allows. The Progress-to-ORACLE utility replaces characters that ORACLE does not accept with the underscore (_). For example, the Progress cust-num 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. Progress requires unique table names.

Progress resolves non-unique table names for you. When Progress 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 Progress 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 has a DataServer option that shows you which ORACLE table mapped to the schema-holder table.

2.1.3 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.

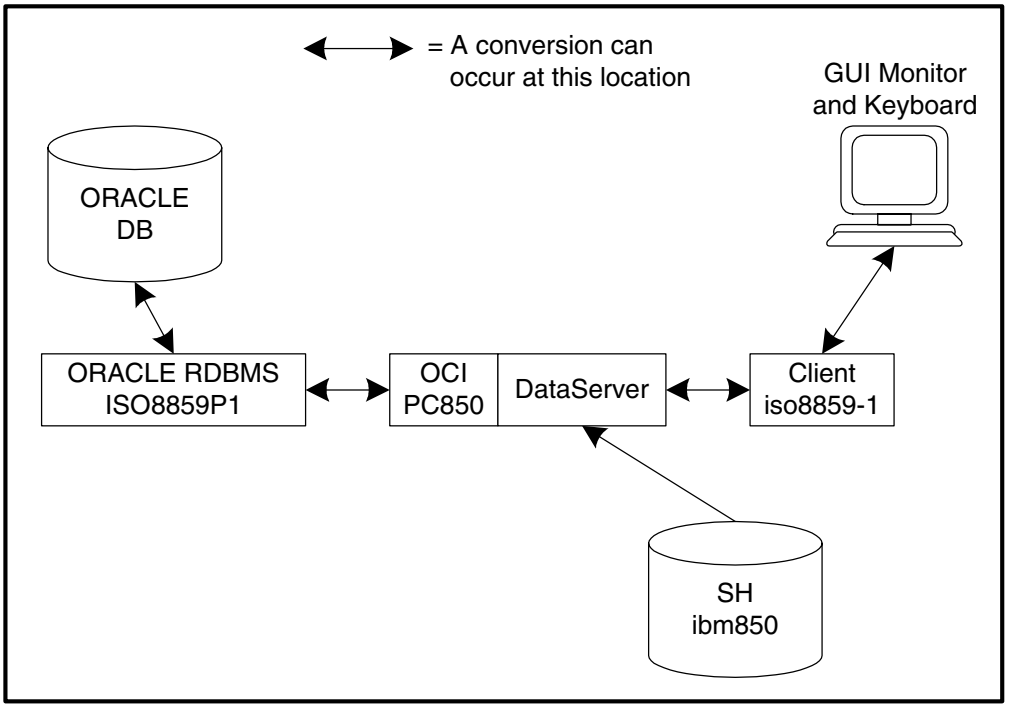


Figure 2–1: DataServer Processes and Code Pages

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

For Progress 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. A system environment variable determines the code page that the OCI uses. 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 Progress recognizes the code page. For example, in this configuration, the OCI uses a code page named PC850. This is the same code page that Progress recognizes as ibm850. You then specify ibm850 as the code page for the schema image.

The default code page setting for the schema image is ibm850. 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 Progress 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 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 in Chapter 3, "Configuring the DataServer," for more information.

The Internal Code Page (-cpinternal) startup parameter determines the code page that the Progress 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, Progress allows you to define your own conversion table.

Progress 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.

For a complete discussion of how Progress handles code-page issues, see the *Progress Internationalization Guide*.

2.1.4 Indexes

You create and maintain all indexes from within ORACLE. When you create a schema image from a target ORACLE database, Progress automatically copies the ORACLE index definitions.

Indexes allow you to use the OF keyword in Progress 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 the Progress 4GL USE-INDEX modifier. Progress 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 Progress statements are equivalent when accessing an ORACLE database:

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

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

NOTE: If you do not specify USE-INDEX or ORDER-BY, 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 or ORDER-BY in your query definition.

ORACLE chooses which index, if any, to use when the Progress 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 the Progress 4GL 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 a 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 Progress 4GL code, you enhance ORACLE performance, especially in cases where your application returns records in a descending sequence.

2. If you do not use the Progress 4GL 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 cust-num for the following query:

```
FOR EACH customer WHERE address = "55 Cambridge" BY cust-num:
```

   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" and "ORACLE Hints" sections for more information.

Leading and Trailing Spaces

Progress 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 Progress client session.

2.1.5 Case-insensitive Indexes

ORACLE supports only case-sensitive indexes, but Progress 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 Progress and ORACLE databases. When a field is not case sensitive, Progress 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.

Modifying Tables to Support Case-insensitive Indexes

To use case-insensitive indexes with an ORACLE database, make the following changes to your ORACLE database table:

1 ♦ Add a column of the same data type adjacent to the indexed column. The new column must precede the indexed column.

2 ♦ Name the column U##*column-name*.

For example, if your table has an indexed column named emp_id, name the column U##emp_id. The new column accommodates the uppercase version of the index.

3 ♦ Set the U## column to the uppercase value of the original column. The SQL statements have the following syntax:

```
UPDATE table-name
SET U##column-name = UPPER(column-name);
```

4 ♦ If non-Progress applications will be modifying the original column, you must create an ORACLE trigger to populate the U##*column-name* with the uppercase value of *column-name*.

5 ♦ Re-create the index with the U## column as a component in place of the original column.

6 ♦ If you have already created your schema holder, use the DataServer utilities in the Data Administration tool to update your schema image. See Chapter 5, "The DataServer Tutorial," for instructions on updating a schema image.

When you use the Progress-to-ORACLE migration utility, Progress automatically modifies ORACLE data definitions to support case-insensitive indexes. For example, if your Progress database has an indexed and case-insensitive field named emp_id, the utility creates an additional column named U##emp_id in the ORACLE table and copies the uppercase value of the data into it.

When the DataServer creates uppercase values for U##*column-name*, it might not create expected values for 0-127 range characters in double-byte code pages. This might cause the "CHARACTER" option of the LENGTH, OVERLAY, and SUBSTRING functions to return unexpected results. Use the "RAW" or "FIXED" options. You should pass "RAW" as a string constant, not an expression.

2.1.6 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 shows:

SYNTAX

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

SYNTAX

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

The DataServer enforces the ordering of the query according to the index that the Progress 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 Progress ROWID function, that is, a PROGRESS_RECID column, the native ROWID, or a unique integer index. Performing an index reposition may cause the DataServer to issue a new query to ORACLE. This new result set may contain rows that were added or changed since the original query was ordered. Note, 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. Performing an index reposition may cause the DataServer to issue a new query to ORACLE. This new result set may contain rows that were added or changed since the original query was opened.

2.1.7 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 Progress 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.

2.1.8 ORACLE Views

ORACLE schema objects include views. A *view* is a presentation of data in one or more tables. The DataServer for ORACLE fully supports views that are subsets of a single table. Table 2–2 lists the level of support available for various types of views.

Table 2–2: Supported ORACLE Views *(1 of 2)*

View	Supported 4GL

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

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

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

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

Table 2–2: Supported ORACLE Views *(2 of 2)*

View	Supported 4GL

Columns from multiple tables
`SELECT cust-num, name, zip, order-num FROM customer, order WHERE sales-rep = 'HXM'`
	The only supported statements are:
	`FOR EACH ... NO-LOCK | SHARE-LOCK:`
	`FIND ... NO-LOCK | SHARE-LOCK:`

With aggregates, functions, or joins that contain a LONG column	You cannot access the LONG column if it contains more than 255 bytes of data. The column is truncated, the query stops, and you receive an error message.

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-Progress views. Nor can you access them through the PRO/SQL menu functions.

In addition, Progress 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, Progress returns a run-time error. See the *Progress Programming Handbook* for information on subtransactions.

Multi-table Views

The DataServer supports direct access to multi-table views. Use the following 4GL syntax to read rows from multi-table views:

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

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

Progress 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, Progress returns run-time errors if you try to update them with a Progress application.

Views Containing Aggregates

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

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

Use the following 4GL syntax to read rows from views that contain aggregates or functions:

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

You can also access the view by using the RUN STORED-PROC send-sql-statement option to send an SQL statement to select the data from the view. You can access a view by using the send-sql-statement option without adding index definitions for the view in the schema holder. See the "Sending SQL Statements" section for more information.

2.1.9 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 Progress and ORACLE triggers. In an application that combines both Progress and ORACLE triggers, the Progress trigger (CREATE, UPDATE, DELETE) executes first. As each Progress 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.

2.1.10 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, incrementation, and whether numbers are reused. For example, this is the code for creating a sequence named *table-name*_SEQ that starts with 1 and increments by 1:

```
CREATE SEQUENCE table-name_SEQ START WITH 1 INCREMENT BY 1;
```

ORACLE does not allow you to change the starting value of a sequence. If you change the value in Progress and then run the DELTA SQL utility, the ORACLE SQL file has an entry like this:

```
ALTER SEQUENCE NEXT_CUST_NUM
START WITH 1500
```

If you load this SQL into ORACLE, you get an ORACLE error telling you that you cannot change the starting sequence number. The workaround for this is to drop the sequence and add it again.

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.

Table 2–3 compares the features of the Progress and ORACLE sequence generators. An application that relies on sequence generators can access a Progress database and, through the DataServer, an ORACLE database. See the chapter on database access in the *Progress Programming Handbook* for information on defining and using Progress sequences in your database applications.

Table 2–3: Progress and ORACLE Sequence Generators

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

If you add a sequence to a table in your supporting ORACLE database, you must update the schema image to reflect this addition. See Chapter 5, "The DataServer Tutorial," for instructions on updating a schema image.

There are two Progress 4GL 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.

2.1.11 ORACLE Synonyms

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

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

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

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

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

Synonyms and Distributed Databases

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

Synonyms for stored procedures and functions in linked databases are not supported.

Table 2–4 describes the support for synonyms in local and distributed (remote) databases.

Table 2–4: Supported Synonyms

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

2.2 Data Types

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

The DataServer translates the ORACLE data types into Progress equivalents. When an ORACLE data type has more than one Progress equivalent, the DataServer supplies a default data type. The schema image contains the Progress 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 Progress equivalent, DECIMAL. You can then change the data type from DECIMAL to INTEGER or LOGICAL. The "Modifying a Schema Image" section in Chapter 5, "The DataServer Tutorial," explains how to change Progress 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.

Table 2–5 lists the ORACLE internal data types supported by the DataServer and their Progress equivalents. The table also shows the default equivalent supplied by the DataServer for those ORACLE data types with more than one Progress equivalent.

The only ORACLE internal data type that the DataServer does not support is ROWID. The ORACLE ROWID, however, has its programming equivalent in the Progress ROWID. See the "ROWID Function" section for more information.

Table 2–5: ORACLE and Progress Data Types

ORACLE Data Type	Progress Equivalent
CHAR[1]	CHARACTER
VARCHAR2[2,3]	CHARACTER
NUMBER	DECIMAL[4,5] INTEGER LOGICAL[6]
DATE[7,8]	DATE[8] INTEGER[8]
LONG[9]	CHARACTER
RAW (limited support)[10]	RAW
LONG RAW[10]	CHARACTER

The following notes provide additional information on the ORACLE data types and their Progress equivalents:

1. The maximum length of an ORACLE7 CHAR field is 255 bytes. The maximum length of an ORACLE8 CHAR field is 2000 bytes.

2. The maximum length of an ORACLE7 VARCHAR2 field is 2000 bytes. The maximum length of an ORACLE8 VARCHAR2 field is 4000 bytes.

3. 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 only for the two characters. The VARCHAR2 data type is more consistent with the Progress CHARACTER data type.

4. ORACLE has only one numeric data type, which the DataServer translates to a Progress DECIMAL, INTEGER, or LOGICAL data type depending on the scale and precision of the NUMBER column. However, Progress handles the Progress INTEGER data type more efficiently than it does a DECIMAL data type. You can use the Progress Data Dictionary

to change the data type from DECIMAL to INTEGER in the schema image. See the "Modifying a Schema Image" section in Chapter 5, "The DataServer Tutorial," for instructions.

5. 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 Progress seeing a different radix separator.

6. 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. Progress then reads the numeric values stored in the ORACLE column, as Table 2–6 shows.

Table 2–6: LOGICAL Data Type and ORACLE Equivalents

Progress	ORACLE
True	Any non-zero value
False	0

The DataServer stores the contents of a Progress 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.

7. 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 Progress 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.

   Progress converts the time component of the ORACLE date to an INTEGER value. To convert the INTEGER value into time, use the STRING and TIME functions, as described in the *Progress Language 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.

8. 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. The DataServer converts all years greater than 4712 to 4712. For example, if your Progress application updates a DATE column with the year 4750, the DataServer converts it to 4712 before writing it to the ORACLE database.

9. ORACLE allows only one LONG column per table.

10. Progress supports RAW data types for non-Progress databases. For information about programming using the RAW data type, see the *Progress Programming Handbook*. For information about the specific statements and functions, see the *Progress Language Reference*.

The DataServer does not support the ORACLE BFILE, CLOB, or LOB data types nor any Progress SQL-92 data types.

2.3 Arrays

The DataServer supports arrays, which are also called field extents. When you create a schema image for an ORACLE database, Progress interprets a number of specially named ORACLE columns of the same data type as a single Progress 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.

Modifying Tables to Support Arrays

The DataServer allows you to extend Progress's ability to support database arrays to your ORACLE database. To have access to this functionality, you must make the following changes to your ORACLE database table:

1 ♦ Name the columns of an ORACLE database table that you want the DataServer to include in an array *column-name*##1, *column-name*##2, etc. The columns must be adjacent and in sequence.

2 ♦ Make sure that these columns are of the same data type.

For example, if you want the schema image to include an array named MONTH with 12 elements, the ORACLE table must have 12 adjacent columns of the same data type named MONTH##1, MONTH##2, MONTH##3, etc. Progress names the corresponding field in the schema image MONTH. In your applications, refer to each element of the array as MONTH[1], MONTH[2], MONTH[3], etc.

In addition to making the columns for each element the same data type,you must also make them the same length. Otherwise you receive the error message, "Schema holder does not match database schema — file *filename* field/*column-name*##1. (1461)."

3 ♦ If you have already created your schema image, use the DataServer utilities to update your schema image so that it reflects the changes you made to the ORACLE table. See Chapter 5, "The DataServer Tutorial," for instructions on updating a schema image.

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

2.4 Unknown Value

The DataServer supports the unknown value, which is represented by a question mark (?) in Progress. In ORACLE the unknown value is stored as a NULL column. This section describes how the DataServer handles the unknown value. Table 2–7 summarizes how Progress and ORACLE unknown and null values map to each other.

Table 2–7: UNKNOWN and NULL Values

Progress Value	ORACLE Equivalent
UNKNOWN Value (?)	NULL
"" (Zero-length string)	" " (One space)

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

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

The Progress unknown value sorts high in DataServer applications, as it does in Progress applications. Unknown values always appear at the beginning of a sort, regardless of whether the field was sorted in ascending or descending order. For example, the following code displays the names of all customers with unknown customer numbers listed first:

```
FOR EACH customer BY cust-num:
  DISPLAY name.
END.
```

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 a Progress database, the query returns all the customers whose names are not Smith, including those whose names are unknown:

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

However, 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 Progress database.

2.4.1 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 Progress 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 address2 = " ".

FIND FIRST customer WHERE 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 name BEGINS "":
```

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

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

Because unknown 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 address2 <> "foo":
  DISPLAY 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 address2 < ?.
```

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

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

2.5 Record Creation

Record creation behavior differs between a Progress 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 cust with a field called cust-num defined as an indexed field, and you write the following procedure:

```
CREATE cust.
name = "SMITH".
cust-num = 10.
address = "1 Main St".
```

- **Progress** — 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, Progress writes the record after the statement `cust-num = 10.`
- **DataServer for ORACLE** — Writes the record later than Progress does; it writes it at the end of the record scope.

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

```
DEFINE BUFFER xcust FOR cust.
CREATE cust.
cust-num = 111.
FIND xcust WHERE xcust.cust-num = 111.
DISPLAY xcust.
```

- **Progress** — Displays customer 111.
- **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 this program:

```
DEFINE BUFFER xcust FOR cust.
CREATE cust.
cust-num = 111.
VALIDATE cust. /* or RELEASE cust. */
FIND xcust WHERE xcust.cust-num = 111.
DISPLAY xcust.
```

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 a Progress database. If you don't 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.

2.6 Record Locking

Progress applications rely on the ORACLE RDBMS to handle all record locking for the target ORACLE database. Progress locks do not apply to your ORACLE database. Table 2–8 compares Progress locks to their ORACLE equivalents.

Table 2–8: Progress and ORACLE Locks

Progress Lock	ORACLE Lock	Transaction Processing Option Lock[1]
NO-LOCK	None	None
SHARE-LOCK	None	None
EXCLUSIVE-LOCK	Share Update (row-level)	Share Update (row-level)
EXCLUSIVE-LOCK . . . UPDATE	Share Update (row-level) . . . Exclusive (table-level)	Share Update (row-level) . . . Row Exclusive (row-level)

[1] The ORACLE Transaction Processing Option provides a row-level locking manager.

In applications that use the DataServer, locks occur as a result of Progress statements that the DataServer translates into SQL statements and sends to the ORACLE RDBMS. Table 2–9 shows examples of Progress 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 2–9: ORACLE Locking

Progress 4GL Statement	SQL Statements Generated	ORACLE Locks
FIND customer.	SELECT . . . FROM customer;	None
FIND customer EXCLUSIVE-LOCK.	SELECT . . . FROM customer FOR UPDATE;	Share Update
FIND customer. . . . UPDATE customer.[1]	SELECT . . . FROM customer SELECT . . . FROM customer FOR UPDATE; Compares records UPDATE customer . . . ;[1]	None Share Update {Row} Exclusive[2]

[1] When Progress 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, Progress 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, Progress returns a run-time error. When the SELECT . . . FOR UPDATE statement completes successfully, the UPDATE starts.

When the Progress UPDATE completes, Progress 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 Progress 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.

Progress and ORACLE release locks at different points in a transaction. When an application issues an UPDATE, Progress releases the lock once the new data is input. ORACLE does not release the lock until the application issues a COMMIT or ROLLBACK. Progress 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.

See the ORACLE documentation for details on ORACLE locking. See the *Progress Programming Handbook* for details on how Progress transactions and locks work.

2.6.1 Monitoring Locks

Since the DataServer relies only on ORACLE locks, the Progress PROMON utility cannot provide you with information on locks in your database applications. Instead, you can use the SQL MONITOR LOCKS statement to view the current locks for an ORACLE instance. Follow these steps:

1 ♦ Run SQL*DBA.

2 ♦ CONNECT to the database with a DBA user name. For example, this command connects to a database with the DBA user name SYSTEM and password MANAGER.

```
SQLDBA> CONNECT SYSTEM/MANAGER
```

3 ♦ Enter the following command.

```
SQLDBA> MONITOR LOCKS
```

2.7 Transactions

The ORACLE RDBMS handles transaction roll back and recovery, but the Progress transaction scoping rules apply.

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

See the *Progress Programming Handbook* for more information on how Progress handles transactions and error conditions.

2.7.1 Two-phase Commit

If you access more than one database in a transaction (for example, a Progress database and an ORACLE database), Progress uses a two-phase commit protocol to minimize the chances of database corruption. The DataServer supports two-phase commit in a configuration that uses Progress networking and includes one ORACLE database and any number of Progress databases. If a configuration uses SQL*Net or Net 8, the DataServer supports two-phase commit to multiple ORACLE databases.

The DataServer treats a distributed ORACLE database as a single database. Any ORACLE tables accessed through database links do not count as additional database connections. The

DataServer ensures a two-phase commit to the first ORACLE instance; the ORACLE RDBMS and SQL*Net or Net 8 are responsible for distributing the COMMIT to all linked instances.

2.8 Error Handling

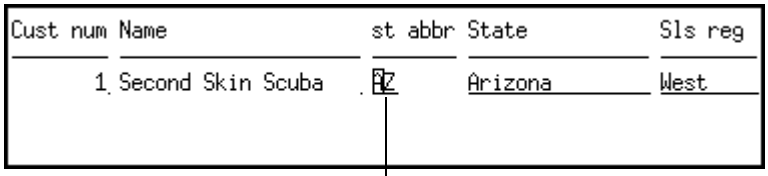
Attempting to add a duplicate record is a common data-entry error. For example, the user might try to create a record, such as a customer record, using a unique key that already exists in the database. If a customer record already exists where the cust-num equals 1, and the user tries to add another customer with the same cust-num, Progress generates an error. When this type of error occurs, Progress 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 the *Progress Programming Handbook* for more information about error handling.

Because the DataServer is accessing a non-Progress database, it cannot detect duplicate key errors until the end of a transaction block. As a result, if the error occurs in a subtransaction, Progress 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 Progress and DataServer error handling:

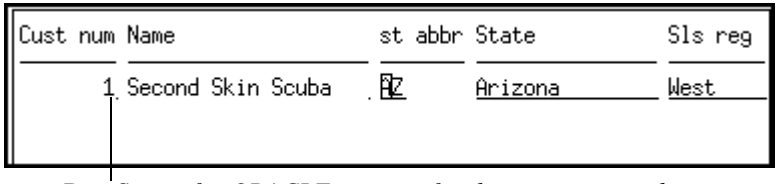
```
rep-blk:
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num NO-ERROR.
  IF AVAILABLE customer THEN
    UPDATE customer.cust-num name.
  do-blk:
  DO ON ERROR UNDO do-blk, RETRY do-blk:
    FIND state WHERE st.st = customer.st.
    DISPLAY state.
    SET state.
  END.
END.
```

This procedure displays the screen shown below. The procedure prompts the user to enter data into the cust-num field, and then into the st-abbr field. Suppose a user enters an existing state (for example, NH) while Progress is processing the DO block. If a duplicate key entry occurs in the DO block, Progress returns control to the DO block. After Progress displays a message that the record exists, it prompts the user for the state abbreviation.



Progress reprompts for the state abbreviation.

With the DataServer for ORACLE, if a duplicate key entry occurs in the DO block, Progress returns control to the REPEAT block. The procedure prompts the user to enter the customer number after the inner transaction completes.



DataServer for ORACLE prompts for the customer number.

When two users simultaneously attempt to create records with duplicate keys, another difference in behavior occurs. Progress 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.

2.9 Cursors

Progress 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, Progress uses cursors when it processes FOR EACH types of statements. Progress 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 Progress-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 CUST-NUM index, and your "current" record is customer number 50. This means that Progress has a cursor positioned at cust-num 50.

2.9.1 Cursor Repositioning

With the DataServer, cursor repositioning will work with an ORACLE database the same as it does with Progress, except when the ORACLE Database Manager fails to find a record. In Progress, 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 cust-num 50 and a request to find customer "smith" in an ORACLE table fails, the cursor remains at cust-num 50. The same failed request from an Progress 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 or new queries. 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 a DEFINE QUERY statement does not reposition cursors in other queries.

2.9.2 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, PREV, and LAST 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 for instructions.

2.9.3 Allocating Cursors

ORACLE allocates a maximum number of cursors that your application can use based on system configuration, the version of ORACLE, and `init.ora` 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 in Chapter 4, "Connecting the DataServer," 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.

2.10 Progress 4GL Issues

The following sections present information about using various 4GL 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.

2.10.1 RECID Function

For backward compatibility, the DataServer supports the RECID function for ORACLE tables that have a PROGRESS_RECID column. The Progress-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 Appendix A, "Upgrading DataServer Applications" 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 for more information.

2.10.2 ROWID Function

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

NOTE: The ROWID function might cause a newly created record to be written earlier to your ORACLE database than to a Progress database. If you do not assign values to all fields that are defined as mandatory in ORACLE for a record, the ROWID function will fail.

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 does 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 in Chapter 5, "The DataServer Tutorial," for instructions on changing how a table supports ROWID.

The native ROWID typically provides the fastest access to a record. However, the native ROWID does not support the Progress 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:

```
FIND FIRST customer USE-INDEX cust-num.
```

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, such as DB2, DB2/400, Informix, ORACLE, Progress, and SYBASE:

- 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.

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

2.10.3 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 Progress 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 Progress browse has by default:

```
/* Define a query named q for customer scrolling. */
DEFINE BROWSE b QUERY q DISPLAY cust-num name address with 10 down.
DEFINE BUTTON upd.
OPEN QUERY q FOR EACH customer.
ENABLE upd b WITH FRAME x.
ON CHOOSE OF upd DO:
  DEFINE VAR n as INT.
  n = CURRENT-RESULT-ROW("q").
  GET PREV q.
  GET NEXT q EXCLUSIVE-LOCK.
  IF CURRENT-RESULT-ROW("q") = n 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 address WITH BROWSE b.
    END.
  END.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

2.10.4 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 Progress and ORACLE databases:

```
DEFINE QUERY myquery FOR customer FIELDS (cust-num name) SCROLLING.
```

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

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

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

```
FOR EACH customer FIELDS (cust-num 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 D, "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 zip = 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 non-lookahead cursors or SHARE-LOCK.

See the Record Phrase entry in the *Progress Language Reference* for more information on the FIELDS option.

2.10.5 FIND PREV/LAST Statements

Progress applications that use the FIND PREV or FIND LAST statements work on ORACLE tables in a manner consistent with Progress.

To support these statements, an ORACLE table must include support for the Progress ROWID function (either through a PROGRESS_RECID column or an indexed NUMBER column with unique values). See the "ROWID Function" section for more information.

2.10.6 FIND FIRST/LAST Statements

There is a difference in performance between FIND FIRST and FIND LAST statements that access a Progress database or an ORACLE database. When you issue a FIND FIRST or FIND LAST statement in a DataServer application, the ORACLE DBMS builds a results set that might include every record in a table. To achieve comparable performance, qualify your FIND statements and queries with a WHERE clause. For example, the following statement retrieves the last record in a table.

```
FIND LAST customer WHERE date > 4-20-94.
```

The WHERE clause qualifies the query by specifying a recent date. ORACLE then includes only records that contain a more recent date in the results set.

The recommendation to use a WHERE clause also applies to the QUERY option. Similarly, when you use a FIND NEXT statement without first using a FIND FIRST or FIND...WHERE, ORACLE builds a results set that includes every record.

2.10.7 Compiling Progress Procedures

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

To compile procedures that access an ORACLE database, start up Progress 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 the *Progress Language Reference*.

R-code

The compiled r-code is portable among like user interfaces. For example, code that you compile in character mode on a Sun machine can run on any other UNIX machine in character mode. Code compiled on Windows 95 can run on NT or Windows 98. But code compiled on Windows cannot run in character mode on a Sun machine.

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 a Progress database.

The size of r-code grows when you compile procedures against an ORACLE database as compared with compiling against a Progress database. The r-code for a DataServer application contains as text the portions of SQL statements that the DataServer passes to ORACLE.

2.10.8 Unsupported 4GL Statements

The DataServer supplies you with the complete functionality of Progress when accessing ORACLE databases. Nearly all 4GL language elements (statements, functions, etc.) and Progress Data Dictionary features work the same whether your application accesses an ORACLE database through the DataServer or a Progress database. You can use the DBRESTRICTIONS function to find out which Progress features that your non-Progress 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 the *Progress Language Reference* for information on syntax.

Table 2–10 lists the 4GL statements and functions that the DataServer for ORACLE does not support.

Table 2–10: Progress and ORACLE Differences *(1 of 3)*

Feature	Description
BEGINS function Abbreviated Index USING option	When you use these 4GL elements to access data in an ORACLE database, you will have different results than you would expect from Progress 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". Progress 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 Progress 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 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 Progress, using this kind of criteria results in poor performance with a Progress 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.

Table 2–10: Progress and ORACLE Differences *(2 of 3)*

Feature	Description
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 DataServer for ORACLE does not support setting a sequence generator's current value.
DBTASKID function	The DataServer for ORACLE does not support this function.
FIND statements QUERY option	To reduce the number of records ORACLE includes in the results set, you should qualify your FIND statements and queries with a WHERE clause. This achieves a performance rate that is closer to Progress 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.
MATCHES function	If you want to use the MATCHES function for a string containing double-byte characters, you must fill out the character expression with periods (.). For example, where J is a double-byte character, the following statement does not find a match: FIND FIRST Customer WHERE Name MATCHES "J." 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.

Table 2–10: Progress and ORACLE Differences *(3 of 3)*

Feature	Description
OPEN QUERY statement	Newly created records might not appear in the results set of queries that you opened before you created the records. Reopen the query to access the new records.
INDEXED REPOSITION clause	Index reposition may cause new SQL to be executed thereby causing newly created records to appear.
SESSION: TIME-SOURCE handle	This system handle returns the ORACLE server's time information.
SETUSERID function	You cannot use this function to change the user ID and password of an ORACLE login.
SHARE-LOCK option	You cannot use this option for a query with the FIELDS option. SHARE-LOCK is NO-LOCK for ORACLE.
Time in the WHERE clause	Progress supports this option only if you have mapped ORACLE DATE columns to Progress CHARACTER fields in the schema image.
USERID function	For a connection to an ORACLE database, the USERID function returns the value that you specified for the -U parameter. For example, if you specify -U bob/password, USERID returns bob/password.

2.11 ORACLE Stored Procedures

The DataServer allows you to call ORACLE PL/SQL stored procedures, stored functions, and packages from within Progress 4GL procedures. *Stored procedures* and *stored functions* are groups of PL/SQL and control-of-flow statements that access the ORACLE database. Stored procedures and functions can return values or result codes. A *package* is a group of stored procedures, functions, and related program objects.

The first time you run an ORACLE stored procedure, the ORACLE RDBMS creates an execution plan for it and stores the plan in the database. The next time you run the stored procedure, it runs as a precompiled procedure. This makes accessing the ORACLE database much quicker and more efficient than accessing it with new queries each time.

When you create or update your schema image, the stored procedures, functions, and packages appear in the list of accessible objects along with tables, views, and sequences. Progress allows you to run the stored procedures you create in ORACLE from within Progress procedures. For complete information about how to create and use stored procedures, see the *ORACLE PL/SQL User's Guide and Reference.*

The Progress Data Dictionary represents each parameter as a field in the schema image, and you can view its properties by opening its Field Property Sheet.

You run stored procedures from within Progress procedures by using the 4GL RUN STORED-PROC statement. An ORACLE stored procedure that you run with this statement can return two types of values:

- Return codes that indicate whether the stored procedure succeeded
- Values of output parameters that you define when you create the procedure

Stored procedures called from within Progress applications cannot return Boolean and native ROWID data types. Table 2–11 lists issues that occur when you pass other data types as parameters.

Table 2–11: Argument Data Types

Data Type	Behavior
DECIMAL FLOAT INTEGER	The DataServer represents all three data types as the Progress INTEGER data type in the schema image. 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 Progress Data Dictionary to update the data type and format information in the Field Property Sheet for the parameter.
VARCHAR2	In ORACLE7, VARCHAR2 parameters cannot be greater than 255. If they exceed this limit, they cause an ORACLE error. In ORACLE8, the limit is 2000 characters.

DATE	If you are passing a DATE data type as an input parameter and using it in an equality test, the equality test might fail. In this case, use the trunc function in the stored procedure. For example:
	
	`procedure x_date (indate in date, outdate out date)`
	`as begin`
	`select date_terminate into outdate from datetbl`
	`  where trunc(hire_date) = trunc (indate);`
	`end;`

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.

The Progress 4GL has statements and functions that allow you to use the return codes and the values of the output parameters. Table 2–12 lists these statements and functions.

Table 2–12: Returning Values from Stored Procedures

Progress 4GL	Description
CLOSE STORED-PROC statement	Retrieves the values from the output parameters you defined for the stored procedure and tells Progress that the stored procedure has ended.
PROC-HANDLE function	Allows you to specify a handle to identify a stored procedure.
PROC-STATUS function	Reads the return value.

The following sections describe how to run ORACLE stored procedures and retrieve return codes, output parameter values, and results sets. All of the following examples of how to run stored procedures in Progress use this stored procedure created in ORACLE:

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

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

NOTE: Typically, you can have only fifty stored procedures running at one time. This number, however, is further restricted by the number of open cursors you specified for your ORACLE database or for the current session. See the "Index Cursors" section in Chapter 4, "Connecting the DataServer," for information on specifying

open cursors. Cursor limitations also vary across platforms. See your ORACLE documentation for more information.

2.11.1 Running a Stored Procedure

The Progress 4GL statement RUN STORED-PROCEDURE allows you to run an ORACLE stored procedure. You must indicate the end of a stored procedure in your Progress procedure by using the CLOSE STORED-PROCEDURE statement.

This is the partial syntax for the RUN STORED-PROCEDURE statement:

```
RUN STORED-PROC procedure  [ integer-variable = PROC-HANDLE ]
  [ ( [ output ] parameter , [ output ] parameter ) ]
```

This is the partial syntax for the CLOSE STORED-PROCEDURE statement:

```
CLOSE STORED-PROC procedure
```

For example, the following Progress 4GL code runs the ORACLE stored procedure pcust:

```
DEFINE VAR h1 AS int.
RUN STORED-PROC pcust h1 = PROC-HANDLE (20, OUTPUT 0, OUTPUT 0).
CLOSE STORED-PROC pcust WHERE PROC-HANDLE = h1.
 DISPLAY pcust.orders pcust.states.
```

This code defines an integer variable that serves as a handle for identifying the stored procedure. If you have only one active stored procedure, you do not have to specify a handle. However, it is good programming practice to use handles to identify all your stored procedures.

The pcust stored procedure passes the values 20, 0, and 0 to the three parameters (specifying orders and states as output parameters) and displays the results. The Progress procedure uses the CLOSE STORED-PROC statement to fetch the orders and states output parameters, then displays them. The stored procedure does not return the value of the output parameters unless you request them with the keyword OUTPUT or INPUT-OUTPUT when you execute the procedure.

You can close all stored procedures at once with the following statement:

```
RUN STORED-PROC closeallprocs.
```

See Appendix B, "Stored Procedure Reference," for a description of the complete syntax for the Progress statements and functions that support running ORACLE stored procedures.

2.11.2 Retrieving Return Codes

An ORACLE stored procedure returns codes that provide information on its status. These codes indicate whether the stored procedure was successful, whether it found a missing object, whether it was in a deadlock, or whether it encountered an error condition. The following example of 4GL 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:

```
/* Return status */

DEFINE VAR stat AS int.
RUN STORED-PROC pcust (PARAM num = ?, output PARAM orders = 0, output param
states = 0).
CLOSE STORED-PROC pcust stat = PROC-STATUS.
IF stat = 0 then
DISPLAY pcust.orders pcust.states.
ELSE
DISPLAY stat.
```

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.

2.11.3 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 an ORACLE stored procedure, you request them with the keyword OUTPUT or INPUT-OUTPUT when you execute the procedure. Output parameters are available only after you close the stored procedure. The following 4GL procedure uses the second option for passing parameters—it passes parameters by name with the PARAM option:

```
/* Parameters by name  */

RUN STORED-PROC pcust (PARAM num = 20, output PARAM states = 0,
 output PARAM orders = 0).
CLOSE STORED-PROC pcust.
DISPLAY pcust.orders pcust.states.
```

When you name parameters with PARAM, you can name only the parameters that you want to use, in any order. When you call parameters by position, you must specify all the parameters expected by the stored procedure and in the expected order.

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 when you want to pass values that are different from the default.

ORACLE allows you to overload procedures, that is, to create several procedures with the same name but that take different arguments. For example, you could have a second stored procedure named pcust that expects the arguments num, orders, and zip. You can call this overloaded procedure by specifying its parameters, num, orders, and zip.

Returning Parameter Values as Arrays

The RUN STORED-PROC statement allows you to return results as an array. To support this feature, you must designate the stored procedure parameter as an extent in the Progress Data Dictionary. Specify the number of elements that you need to accommodate your data.

2.11.4 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:

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

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

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

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

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

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

This 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.

2.11.5 Transaction Processing

When you run an ORACLE stored procedure from a Progress procedure, different scoping rules apply. A stored procedure has its own transaction context-it is not considered an operation within the Progress transaction from which you run it.

When you undo an entire transaction, any data manipulation that a stored procedure called from within that transaction executes (for example, as a result of SELECT, INSERT, DELETE, UPDATE, etc. statements) is undone. However, any data manipulation by a stored procedure that occurs in a subtransaction is not undone, if only the subtransaction is undone. To undo an entire transaction, the DataServer tells ORACLE to perform a ROLLBACK. To undo a subtransaction, the DataServer issues individual SQL statements to reverse the actions taken in the subtransaction. For example, if a subtransaction includes a CREATE statement, the DataServer tracks that information and issues a DELETE statement to undo the subtransaction. With stored procedures, however, the DataServer cannot track individual operations, so it cannot issue the appropriate statements to reverse them.

Any data definition that occurs in a stored procedure (for example, as a result of CREATE, DROP, ALTER, GRANT, etc. statements) is visible to other ORACLE users before the transaction is committed.

2.11.6 Returning Errors

There are two types of errors that stored procedures can encounter. The first type of error can be trapped by the stored procedure. The stored procedure can call raise_application_exception with a value ranging from -20000 to -20999 and continue processing. The DataServer checks whether an error is the result of raise_application_error. If this is the case, the DataServer stores the value, which you can retrieve with the PROC-STATUS function when you close the stored

procedure. The Progress application continues to execute. The same behavior occurs when you send an SQL statement that causes a trigger to fire that results in a raise_application_error.

The second type of error is caught by the RUN STORED-PROC statement itself and the procedure stops. This error might be due to a PL/SQL compilation error or another error that is not handled.

The RUN STORED-PROC statement supports the NO-ERROR option. The following example shows how to trap errors within a procedure:

```
DEFINE VAR h1 AS INTEGER.
DEFINE VAR j AS INTEGER.
RUN STORED-PROC send-sql-statement h1 = PROC-HANDLE NO-ERROR
  ("select count (*) from xxx.customer where name between 'A' and 'Z' ").

IF ERROR STATUS:ERROR THEN DO:
  DO j = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE "error" ERROR-STATUS:GET-NUMBER(j)
        ERROR-STATUS:GET-MESSAGE(j).
  END.
END.

CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = h1.
```

The PROC-STATUS clause of the CLOSE STORED-PROCEDURE statement allows the DataServer to retrieve the text of an error message that was passed to raise_application_error. Use the ERROR-STATUS:GET-MESSAGE handle to retrieve the message as in the following example:

```
DEFINE VARIABLE st AS INTEGER INITIAL 0.
DEFINE VARIABLE h  AS INTEGER.

RUN STORED-PROC p1 h = PROC-HANDLE NO-ERROR.

CLOSE STORED-PROC p1 st = PROC-STATUS WHERE PROC-HANDLE = h.

DISPLAY st.

IF ERROR-STATUS:ERROR
     THEN
   MESSAGE ERROR-STATUS:GET-MESSAGE(1) ERROR-STATUS:GET-NUMBER(1)
          VIEW-AS ALERT-BOX.
```

In this example, the PROC-STATUS clause is necessary to enable the ERROR-STATUS:GET-MESSAGE handle to retrieve the text of the error message.

2.11.7 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.

2.12 Sending SQL Statements

Progress uses stored-procedure syntax to allow you to send SQL statements directly to the ORACLE RDBMS. You can send either one SQL statement or one PL/SQL block. The DataServer uses the RUN STORED-PROCEDURE statement with the send-sql-statement option to pass SQL statements to ORACLE. Although Progress SQL also allows you to use SQL statements, send-sql-statement option gives you access to the complete ORACLE PL/SQL syntax. For example, you can issue SQL statements from within Progress procedures to modify the data definitions of your ORACLE database.

You use the RUN STORED-PROC statement with the send-sql-statement option and pass the PL/SQL statements as a parameter. The SQL statements that you pass must conform to PL/SQL syntax. For example, this code passes a SELECT statement as a parameter:

```
DEFINE VAR h1 AS INTEGER.
RUN STORED-PROC send-sql-statement h1 = PROC-HANDLE
("SELECT cust_num, name
FROM customer WHERE name like 'A_'")
FOR EACH proc-text-buffer WHERE PROC-HANDLE = h1:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = h1.
```

This example returns the customer number and name of each customer whose name begins with A. You can use the results just as you use results from your ORACLE or Progress database. This procedure reads the results into the proc-text-buffer, which is a buffer defined by Progress that accepts only CHARACTER data. A buffer allows you to access individual columns. If you do not use a buffer, each database row returns as a character string. You can also define a buffer that accepts a data type other than the CHARACTER data type, such as LONG or LONG RAW. See the "Defining a Buffer" section, below.

NOTE: Specify a handle for the stored procedure, as in the example. Do not use the default system handle with the send-sql-statement option. The DataServer passes the SQL statement directly to ORACLE. The Progress compiler does not process it, so errors (including syntactical errors) occur only at run time and not when you compile a procedure.

2.12.1 Defining a Buffer

You can read results from a send-sql-statement into a buffer. Progress provides that proc-text-buffer that reads all data as CHARACTER data. Defining your own buffers in an ORACLE database allows you to retrieve database results in their original data types.

Follow these steps to define a buffer.

1 ♦ Define a view in ORACLE:

- By following the naming convention BUFFER_*buffername*
- With the same number of columns and data types that the SQL statement returns
- With the columns in the order that the SQL statement returns them

For example, to return two types of values, an integer and a character string, use an ORACLE SQL utility to define the following buffer for your ORACLE database:

```
CREATE VIEW BUFFER_custorder AS SELECT customer.cust_num,
customer.name FROM customer
```

2 ♦ Update your schema image using the Update/Add Table Definitions DataServer utility for ORACLE. The utility adds the view to the list of accessible objects. The DataServer defines the view as a buffer that Progress can use.

See the "Updating a Schema Image" section in Chapter 5, "The DataServer Tutorial," for instructions on using this utility.

This buffer defines two returned values-an INTEGER and a CHARACTER value, in that order. The order of the fields is significant. If the data types do not match those returned by the SQL statement, the procedure returns the values in a different order than you specified, and you receive a run-time error. For this buffer, the SQL statement must return two data types, INTEGER and CHARACTER, in that order. If it returns only the CHARACTER data type or first the CHARACTER and then the INTEGER data type, you receive a run-time error. If it returns only the INTEGER data type, the CHARACTER field will have the unknown value.

The easiest way to create a buffer that accepts data from ORACLE stored procedures is to use the text of the SQL SELECT statement from the send-sql-statement option. This way, you ensure that you define your data types correctly, and in the correct order.

The following code example creates a view that never returns any results. Defining a view this way indicates that you will not use it as a view, but as a buffer. The BUFFER_ *prefix* allows the Data Dictionary to determine that this view defines a buffer. 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 in the schema image:

```
CREATE VIEW BUFFER_pcust_orders AS SELECT customer.cust_num,
    customer.name, order_num FROM customer, order_
     WHERE 1 = 2
```

Defining a separate buffer for each data type allows the return values to maintain their data types. You can then use the returned values in calculations without first converting them back to their original data types. Reading your results into an explicitly defined buffer also allows you to manipulate the data just as you would manipulate data from a Progress database, with Frame phrases and FORM statements, for example. You might also want to define separate buffers for columns. Separate buffers make your output more readable, because Progress builds a new default frame for each buffer.

This example runs the send-sql-option twice; procedure handles (through the PROC-HANDLE function) identify the different results from the ORACLE database:

```
/* Procedure handles */

DEFINE VAR handle1 AS integer.
DEFINE VAR handle2 AS integer.
DEFINE VAR x AS character.
RUN STORED-PROC send-sql-statement handle1 =
 PROC-HANDLE ("SELECT cust_num, state FROM customer").

FOR EACH cust-buf WHERE PROC-HANDLE = handle1:
    x = "SELECT state, state-name, region FROM state
      WHERE state = '" + cust-buf.state + "'".

  RUN STORED-PROC send-sql-statement handle2 = PROC-HANDLE (x).
  FOR EACH state-buf WHERE PROC-HANDLE = handle2:
    DISPLAY state-buf.
  END.
  CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle2.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle1.
```

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

2.13 Enhancing DataServer Performance

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 in Chapter 4, "Connecting the DataServer," for information on tuning queries at compile time and run time. See Appendix D, "Sample Queries," for examples of how the DataServer handles queries with and without query tuning options.

2.13.1 SQL and Progress Queries

The DataServer allows you to use different approaches for querying an ORACLE database. Your application might be able to take advantage of their strengths depending on the kind of query you are writing and the kind of data you are accessing. The approaches are:

- **Progress 4GL** — The DataServer generates optimal PL/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.
- **Progress SQL SELECT** — When you use a SQL SELECT statement in a Progress 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.
- **Progress SQL-92** — Do not use SQL-92 syntax in applications that access the DataServer. The Progress SQL Engine (which compiles SQL-92) is not integrated into the DataServer architecture.
- **ORACLE PL/SQL** — If you want to use specialized query syntax supported only by PL/SQL, you can use RUN-STORED-PROC send-sql-statement to send the syntax to ORACLE. Use this approach to modify the data definitions of your ORACLE database from the Progress client, for example. Or, if you want to use BEGINS as a search criterion,

a PL/SQL query can result in better performance. Note, however, that Progress and PL/SQL queries produce different results when accessing CHAR data because Progress uses bind variables. Use the QUERY-TUNING NO-BIND-WHERE option in the Progress query for results that are more similar to results from a PL/SQL query.

Another factor to keep in mind when deciding which technique to use for issuing queries is whether a query is better served by being processed by the client or the server. A Progress 4GL query is processed by the client (except in the cases of most joins), and SQL SELECT statements or PL/SQL statements are processed by the server (the ORACLE DBMS).

2.13.2 Bind Variables

The DataServer has enhanced memory management for bind variables, which causes variables in SQL statements to be re-bound less frequently.

SQL SELECT statements use bind variables. When you use the SQL SELECT statement in a Progress procedure, the DataServer for ORACLE uses bind variables and generates reusable SQL.

For Progress queries that reference a DATE in the WHERE clause, the DataServer uses a bind variable instead of a literals for the DATE value and generates reusable SQL.

NOTE: The ORACLE DataServer does not substitute bind variables for values in SQL statements that you send directly to ORACLE using the Progress RUN STORED-PROC send-sql-statement syntax.

2.13.3 Query Tuning

In addition to the standard approach of using selection criteria to refine access to data, you can further influence how the DataServer executes a query through the Progress 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 in these Progress statements:

- FOR EACH

SYNTAX

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

- OPEN QUERY

  SYNTAX

```
OPEN QUERY query QUERY-TUNING
  ( query-tuning-option query-tuning-option ... )
```

- DO PRESELECT

  SYNTAX

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

- REPEAT PRESELECT

  SYNTAX

```
REPEAT PRESELECT table QUERY-TUNING
  ( query-tuning-option query-tuning-option ... )
```

Place the QUERY-TUNING phrase after the last record phrase. For example, place it near the end of the statement where you also place block modifier phrases such as BREAK, ON ERROR, and TRANSACTION. Separate multiple query-tuning options by a single space. The QUERY-TUNING options have equivalent startup parameters. You cannot use the startup parameters to override the QUERY-TUNING settings.

Table 2–13 describes the query-tuning options.

Table 2–13: Query-tuning Options *(1 of 5)*

Option	Description
ARRAY-MESSAGE NO-ARRAY-MESSAGE	Specifies whether the DataServer sends multiple result rows in a single logical network message, thereby reducing network traffic. Default: ARRAY-MESSAGE, if the query uses a lookahead cursor
BIND-WHERE NO-BIND-WHERE	Specifies whether the DataServer uses ORACLE bind variables for values in WHERE clauses. Using bind variables typically improves performance, but ORACLE provides unexpected results for some operations, such as a MATCHES on an indexed field and a trailing wildcard or comparisons of CHAR fields that use ORACLES blank-padding rules. Specify NO-BIND-WHERE to use literals. Using NO-BIND-WHERE in queries that do comparisons (MATCHES, BEGINS on an indexed field) can improve performance. Default: BIND-WHERE

Table 2–13: Query-tuning Options *(2 of 5)*

Option	Description
CACHE-SIZE *integer* BYTE CACHE-SIZE *integer* ROW	Specifies the size of the cache for information (in bytes or records) used by lookahead or standard cursors. If you have two Progress statements that cause the DataServer to generate identical SQL code except that the second statement specifies a smaller cache size, the DataServer reuses the larger cache from the first statement if the cursor is still available. Reusing cache and cursors improves performance. Byte maximum: 65535 bytes Byte minimum: Specify the number of bytes contained in a single record. For joins, specify the number of bytes contained in two joined records. By default, the DataServer sizes the cache to accommodate one record or, for a join, two joined records. For example, if a join returns a 500-byte record, you need a cache of at least 1000 bytes. Default: 1024 bytes with standard cursors; 8192 with lookahead cursors Row maximum: the number of records that can fit in 65535 bytes. See the "Caching Records" section for more information. Row minimum: 1 for a single table; 2 for a join

Table 2–13: Query-tuning Options *(3 of 5)*

Option	Description
DEBUG EXTENDED DEBUG SQL NO-DEBUG	Specifies whether the DataServer should print debugging information that it generates for the query to the `dataserv.lg` file. Specify DEBUG SQL to print the SQL that the DataServer executes against the ORACLE DBMS. Specify DEBUG EXTENDED to print additional information, such as cursor statistics. There are additional options for collecting advanced statistics with DEBUG. See the "Analyzing Performance" section in Chapter 4, "Connecting the DataServer." Default: NO-DEBUG
HINT *string1 string2 string3*	Specifies the ORACLE hint syntax that the DataServer passes directly to the ORACLE DBMS as part of the query. This allows you to control which hints are passed as opposed to the index hints that the DataServer passes when appropriate. When you have to specify an index name in the hint syntax, use the name defined in the ORACLE database. Because the DataServer generates aliases for ORACLE tables using names from T0 through T9, use these aliases to refer to tables in the hint syntax. The DataServer passes the opening symbols (/*+) and closing symbols (*/). For example, to pass the /*+ORDERED*/ hint syntax, you specify only HINT "ORDERED". Passing incorrect hint syntax, inappropriate hints, or conflicting hints will not return an error but might give you unpredictable results. See ORACLE documentation for information on hint syntax.

Table 2–13: Query-tuning Options *(4 of 5)*

Option	Description
INDEX-HINT NO-INDEX-HINT	Specifies whether the DataServer should provide index hints to the ORACLE DBMS. Generally index hints improve performance, but ORACLE's responses to hints vary between releases. Specify NO-INDEX-HINT to test whether performance for a query improves when the DataServer executes it without hints. See the "Indexes" section for more information on index hints. By default, the DataServer passes index hints. You can turn off the default globally at compile time or run time by specifying the -noindexhint startup parameter when you start a Progress session. Use INDEX-HINT to retain the behavior for individual queries. Default: INDEX-HINT
JOIN-BY-SQLDB NO-JOIN-BY-SQLDB	Specifies whether the DataServer allows the ORACLE DBMS to perform a join, which usually improves performance. Default: JOIN-BY-SQLDB You can turn off the JOIN-BY-SQLDB default globally at compile time by specifying the -nojoinbysqldb startup parameter when you start a Progress session. The -nojoinbysqldb parameter does not override the explicit use of JOIN-BY-SQLDB in the QUERY-TUNING phrase.

Table 2–13: Query-tuning Options *(5 of 5)*

Option	Description
LOOKAHEAD NO-LOOKAHEAD	Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as fit in the allocated cache (CACHE-SIZE), which limits the number of database accesses thereby improving performance. Using lookahead cursors results in behavior that is different from Progress because the client does not see any changes made to the records in the cache. Specify NO-LOOKAHEAD for behavior that is consistent with Progress. Default: LOOKAHEAD, except with FIND statements and statements that use an EXCLUSIVE lock
ORDERED-JOIN	Specifies that the DataServer embed the ORDERED hint syntax in the SQL it generates.
REVERSE-FROM	Specifies that ORACLE join tables in the reverse order in which they appear in the FROM clause. The DataServer generates a new SQL FROM clause with the tables in reverse order. REVERSE-FROM is independent of the ORDERED-JOIN option.

NOTE: All of the query-tuning options take effect at both compile and run time except for the INDEX-HINT, JOIN-BY-SQLDB, and NO-JOIN-BY-SQLDB options, which apply only at compile time.

The following example shows how to use the QUERY-TUNING phrase to enhance performance. It includes a join which the DataServer instructs ORACLE to perform by default. The QUERY-TUNING options specify that no lookahead cursors will be used. In addition, the DataServer will write an extended report on the SQL statements it executes:

```
FOR EACH customer, EACH order OF customer WHERE order-num > 20
  BY cust-num
  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:

```
FOR EACH customer, EACH order OF customer WHERE order-num > 20
  QUERY-TUNING(CACHE-SIZE 20 ROW HINT "FIRST_ROWS" DEBUG EXTENDED)
  TRANSACTION:
```

2.13.4 Caching Records

The DataServer caches results sets from the ORACLE database to enhance performance. It caches as much data as fits in its allocated cache size. Depending on what kind of cursor a query is using, the DataServer caches row identifiers or records:

- **Standard cursors** — The DataServer caches row identifiers for the results set. If the database table is using the native ROWID as the row identifier, each identifier requires 18 bytes of cache. If the table is using a PROGRESS_RECID column or another index as the row identifier, each identifier requires 4 bytes of cache. Therefore, a results set of 100 records requires either 1800 or 400 bytes of cache.

- **Lookahead cursors** — The DataServer caches complete records or partial records as specified by a field list. The DataServer uses the maximum length allowed for a row as defined in the ORACLE database to calculate the record length, not the actual contents of the record. In addition to the defined row length, the record consists of a row identifier field. Therefore, a row with a defined maximum length of 100 bytes and a native ROWID field (used by the DataServer as the row identifier) requires 118 bytes of cache. The DataServer counts a LONG or LONG RAW column as being 256 bytes long. If a LONG or LONG RAW column is longer than 256 bytes, the DataServer refetches it.

In the case of joins, each record in the cache is a result of the fields selected in the join. In addition to the record, there is a row identifier field (4 or 18 bytes) for each table involved in the join. For example, a three-way join for tables that use the native ROWID as a row identifier, adds 54 bytes to the cache for each result row.

You can affect the performance of a query by controlling the size of the cache. As queries generate different results, they benefit from different cache sizes. Generally, the larger the cache, the faster the performance. However, you must balance cache size against other memory requirements for your system. Consider also that continually adjusting cache size in an application might decrease performance as each adjustment requires the DataServer to make several calls to the OCI.

To determine the optimal cache size for a query, experiment with different values for CACHE-SIZE and use DEBUG EXTENDED to generate cursor statistics in the `dataserv.lg` file that you can examine. Aim for minimal cursor activity. You might also want to lower the cache size for queries that typically fetch only a row or two. This makes memory available for other, more productive uses.

The following statement is an example of setting an optimal cache size for a particular query against the Sports database:

```
FOR EACH customer, EACH order OF customer WHERE order-num > 20
  QUERY-TUNING(CACHE-SIZE 20 ROW DEBUG EXTENDED):
```

2.13.5 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 Progress. 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.
- 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 Progress Data Dictionary.

If you create your ORACLE database using the Progress-to-ORACLE migration utility and choose the Create Extended 4GL Objects 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 Progress ROWID function

2.13.6 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 Progress 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 times and by reducing network traffic as you can send only those fields you want to update.

2.13.7 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 Progress. To get results that are

consistent with Progress, 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. It uses the following criteria to determine whether a join by SQL DB is possible:

- All tables in the join are in the same logical Progress 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 order-num:
```

- 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

[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 2–14 describes how these controls interact and affect the behavior.

Table 2–14: Controlling Join by SQLDB

QUERY-TUNING	Startup Parameter	Behavior
JOIN-BY-SQLDB	-nojoinbysqldb	ORACLE performs the join if possible.
JOIN-BY-SQLDB	None	ORACLE performs the join if possible.
NO-JOIN-BY-SQLDB	-nojoinbysqldb	The client performs the join.
NO-JOIN-BY-SQLDB	None	The client performs the join.
None	-nojoinbysqldb	The client performs the join.
None	None	ORACLE performs the join if possible and if the join is a true join.

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

```
FOR EACH customer, EACH order:
```

You receive a warning if you specify JOIN-BY-SQLDB when it is impossible to have ORACLE perform the join, and the DataServer performs the join instead. You receive a warning at compile time if you specify JOIN-BY-SQLDB when it is not optimal to have ORACLE perform the join.

Improving Join Performance

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.

2.13.8 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. For example, there might not be maximum size information if you had a Version 8 schema holder that you dumped and loaded into a Version 9 schema holder. In Version 8, the DataServer did not store information in the schema holder on maximum sizes of CHARACTER and RAW columns. To make sure that the schema holder has the required maximum size information, use the Progress Data Admin Update/Add ORACLE Table Definitions utility 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.

2.13.9 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 these and use the DEBUG diagnostic options to gather statistics on how your application runs. See the "Analyzing Performance" section in Chapter 4, "Connecting the DataServer" for information on collecting statistics:

- 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 a Progress SQL statement. If you use a Progress 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.

3

Configuring the DataServer

Building the DataServer for ORACLE involves creating executables for several processes. This chapter provides step-by-step instructions for initially setting up the DataServer. Specifically, it describes how to:

- Configure DataServer modules on NT
- Configure DataServer modules on UNIX
- Configure DataServer modules on Windows
- Configure an international environment
- Configure the DataServer in the Progress Explorer administration framework
- Create, maintain, and deploy a schema holder

3.1 DataServer Modules

The 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 3–1 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 3–1: Installing DataServer Modules *(1 of 2)*

Client	DataServer	Installing and Configuring
NT	Local NT	Use the default Progress client executable (`prowin32`). This executable also allow you to use SQL*Net or Net 8 to access ORACLE remotely.
NT or Windows	Remote NT	On the client machine, use the default Progress client executable (`prowin32`). On the NT host machine, use the default broker executable (`_probrkr`) and configure it using ProControl. Unless you set ORASRV to point to another DataServer executable, the broker starts the default DataServer executable `_orasrv`.
Windows	Remote UNIX	On the client machine, use the default Progress client executable (`prowin32`). On the UNIX host machine, use the default broker executable (`_probrkr`) and set ORASRV to point to the `_orasrv` DataServer executable to access ORACLE7 or ORACLE8. This executable supports Progress networking.

Table 3–1: Installing DataServer Modules *(2 of 2)*

Client	DataServer	Installing and Configuring
UNIX	Local UNIX	To access ORACLE7 or ORACLE8, use the `orarx` client executable. This executable also allow you to use SQL*Net or Net 8 to access ORACLE remotely.
UNIX	Remote UNIX	On the UNIX client machine, use the default client executable (`_progres`). On the UNIX host machine, use the default broker executable (`_probrkr`) and set ORASRV to point to the `_orasrv` DataServer executable to access ORACLE7 or ORACLE8. These executables support Progress networking.
UNIX	Remote NT	On the UNIX client machine, use the default client executable (`_progres`). On the NT host machine, use the default broker executable (`_probrkr`). Unless you set ORASRV to point to another DataServer executable, the broker starts the default DataServer executable `_orasrv`. These executables support Progress networking.
Windows	Local Windows	Use the default Progress Windows client executable (`prowin32`). This executable also allow you to use SQL*Net or Net 8 to access ORACLE remotely.

NOTE: If you are using a DataServer on UNIX to access ORACLE on a platform where ORACLE does not support shared objects, you must build DataServer executables instead of using the default client, broker, and DataServer executables. See Appendix E, "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 Progress modules run on the same UNIX machine, see the "Configuring UNIX Modules" section. If you are building a remote DataServer configuration

with a UNIX client accessing an NT host, see the "Configuring UNIX Modules" section for instructions on setting up the client and see the "Configuring NT Modules" section for instructions on setting up the host.

3.2 Configuring NT Modules

A DataServer configuration that includes NT 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, an NT client can access a DataServer running remotely on a UNIX host. See Table 3–1 for a list of the possible configurations for NT. It describes what you must do to set up the client and server modules.

In a remote configuration, 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.

Optionally, you can build customized broker and DataServer executables using the PROBUILD utility. Using PROBUILD on NT is similar to using it on UNIX. See Appendix E, "Building DataServer Executables," for instructions.

The following sections explain how to configure and run the local DataServer and how to configure and run the DataServer modules on the NT host machine. For instructions on setting up the client side of remote configurations on:

- **UNIX** — See the "Configuring UNIX Modules" section for information on the UNIX client.
- **Windows** — See the "Configuring Windows Modules" section for information on the Windows client.

3.2.1 Configuring the Local DataServer on NT

To configure the local DataServer on NT client module, follow these steps:

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 an `.ini` file. See Table 3–2 for information on these variables.

2 ♦ If you plan to use SQL*Net or Net 8 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.

3.2.2 Configuring the DataServer on the NT Host

Before you can run the DataServer modules on the NT host, you must set environment variables. The Progress ProControl utility provides a graphical interface that allows you to manage and configure the DataServer host environment. If you do not choose to use ProControl and want to set environment variables manually, see Table 3–2.

If you have not used ProControl before, see the *Progress Database Administration Guide and Reference* or the ProControl online help for security issues. For example, depending on whether a drive is configured as FAT or not, you must specify in the NT Services utility which account users will use to log on.

Follow these steps on the NT host to use ProControl to configure the DataServer modules:

1 ♦ Access ProControl by double-clicking its icon in the NT Control Panel. The ProControl dialog box opens.

2 ♦ Choose Detail. Tabbed folders appear.

3 ♦ Choose the Options tab and activate the DataServer Page toggle.

4 ♦ Choose OK and choose Detail on the ProControl main window. The DataServers tab appears.

5 ♦ Choose the DataServers tab and choose Insert to add an entry for the broker. The General and Environment tabs appear.

6 ♦ Enter the appropriate information in the following fields in the General folder:

- **Identifiers** — Provide a unique name to identify the broker. The name must be unique to the host. The identifier can be up to 30 characters long. For example, you might name it orabroker.

- **Executable Name** — Specify the broker executable. By default, the `_probrkr` executable displays. If you want to use a customized executable that you built using the PROBUILD utility, specify it here.
- **Working Directory** — Specify the directory that the broker uses, for example, to create the `dataserv.lg` file.

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.

- **Service Name** — Specify the value for the Service Name (-S) startup parameter. Look in your *`install-path`*`\system32\drivers\etc\services` file for an available service that the broker can use.
- **Network Protocol** — Specify TCP for the Network (-N) startup parameter.

7 ♦ Activate the Auto Start toggle if you want the broker to start automatically when you boot the host machine. To start the broker automatically, you must also use the NT Services utility to set the Progress ProService utility to start automatically.

8 ♦ Choose the Environment tab in the DataServers folder and set the required variables as described in Table 3–2.

Table 3–2: DataServer for ORACLE Environment Variables

Variable	Value
ORACLE_HOME	The pathname of the directory where you installed ORACLE.
ORACLE_SID	The identifier for the ORACLE instance you are accessing.
ORASRV	The name of the executable (including the path) of the ORACLE DataServer. If you are not using the default DataServer executable, `_orasrv`, set ORASRV to point to the appropriate executable.

9 ♦ Choose OK.

10 ♦ Start the broker by choosing Start on the DataServers folder.

Once the broker is running, you can build the schema holder for the ORACLE database. See the "Creating a Schema Holder" section for instructions.

3.3 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 on an NT host.

In a client-server configuration, 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 3–1 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.

3.3.1 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 3–3 describes the environment variables and how to set them for the local DataServer.

Table 3–3: Environment Variables for the Local DataServer

Variable	How to Set It
DSLOGDIR	The pathname of the log file that Progress uses to keep track of DataServer processes and error messages. By default, Progress writes to `$DSLOGDIR/dataserv.lg` in the current directory. (Optional)
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	Identifies ORACLE shared libraries. Set this variable to include `$ORACLE_HOME/lib`. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. (Required when running executables, whether supplied or built with PROBUILD, that access ORACLE7 or ORACLE8)
ORACLE_HOME	The pathname of the directory where you installed ORACLE. (Required)
ORACLE_SID	The identifier for the ORACLE instance you are accessing. (Required)
PROEXE	The pathname of the Progress client executable you created with PROBUILD. After setting this variable, you can run your customized executable entering **pro** at the system prompt. Make sure that you have set DLC to the directory where Progress is installed. (Optional)

If you plan to use SQL*Net or Net 8 to access the ORACLE database, make sure that you have configured the `tnsnames.ora` file as your ORACLE documentation instructs. The Progress 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 a Schema Holder" section for instructions.

3.3.2 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 3–4. 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 3–4 that you set on the client is PROEXE.

Table 3–4: Environment Variables for the Remote DataServer *(1 of 2)*

Variable	How to Set It
DSLOGDIR	The pathname of the log file that Progress uses to keep track of DataServer processes and error messages. By default, Progress writes to `$DSLOGDIR/dataserv.lg` in the current directory. (Optional)
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	Identifies ORACLE shared libraries. Set this variable on the host machine to include `$ORACLE_HOME/lib`. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. (Required when running executables, whether supplied or built with PROBUILD, that access ORACLE7 or ORACLE8)
ORACLE_HOME	The pathname of the directory where you installed ORACLE. (Required)
ORACLE_SID	The identifier for the ORACLE instance you are accessing. (Required)
ORASRV	The name of the executable (including the path) of the ORACLE DataServer. Set ORASRV to point to the appropriate DataServer executable: (Required) To access ORACLE7 or ORACLE8, point to the `_orasrv` DataServer executable.

Table 3–4: Environment Variables for the Remote DataServer *(2 of 2)*

Variable	How to Set It
PROBRKR	The pathname to the executable of the broker. (Optional)
PROEXE	The pathname of the Progress client executable you created with PROBUILD. After setting this variable, you can run your customized executable (by default, `_progres`) by entering pro at the system prompt. Make sure that you have set DLC to the directory where Progress is installed. Set this variable on the client machine. (Optional)

Once you have set up your environment, you are ready to create a schema holder. See the "Creating a Schema Holder" section for instructions.

3.4 Configuring Windows Modules

A DataServer configuration that includes a Windows 95 or 98 client can be local or client-server, that is, remote. If all DataServer modules are running on the client PC, the Progress client accesses the remote ORACLE database through SQL*Net or Net 8. See Table 3–1 for a list of the possible configurations. A Windows machine cannot serve as a host machine in a remote DataServer configuration.

To configure the Windows client module, follow these steps:

1 ♦ Make sure the ORACLE_HOME and ORACLE_SID variable is set correctly. You can set variables in the registry (ORACLE sets ORACLE_SID at installation time), at the system level or in an `.ini` file.

2 ♦ If you plan to use SQL*Net or Net 8 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.

Optionally, you can build a customized client executable using the PROBUILD utility. Using PROBUILD on Windows is similar to using it on UNIX. See Appendix E, "Building DataServer Executables," for instructions on building DataServer executables.

3.5 Configuring an International Environment

If you plan to access an ORACLE database that uses a double-byte character set or that relies on non-Western European languages, you must configure your DataServer so that it runs with the appropriate locale settings.

NOTE: By default, the DataServer for ORACLE is double-byte enabled (DBE). Use or build the standard executables.

You can use the DataServer to access double-byte data in the following configurations:

- On NT, a graphical client accessing a local or remote DataServer and a local or remote ORACLE instance through Progress networking or SQL*Net or Net 8.
- On NT, a batch client accessing a local DataServer and a local ORACLE instance or accessing an ORACLE instance on another host through SQL*Net or Net 8.
- On UNIX, a character-mode client using VT100 or a VT100 terminal emulator accessing a local or remote DataServer and a local or remote ORACLE instance through Progress networking or SQL*Net or Net 8.
- On UNIX, a batch client accessing a local DataServer and a local ORACLE instance or accessing an ORACLE instance on another machine through Progress networking or SQL*Net or Net 8.

Follow these steps to configure your international environment. The examples describe a double-byte environment, but the instructions apply to other locales as well:

1 ♦ Make sure that your installation of ORACLE includes the multilingual option with ALL LANGUAGES.

2 ♦ On UNIX, make sure that the ORACLE environment variable, NLS_LANG, is set correctly. You must set the values for language, territory, and character set correctly (`NLS_LANG=`*`language_territory.charset`*). For example, to access a Japanese database, set NLS_LANG to **JAPANESE_JAPAN.JA16EUC.** See your ORACLE documentation for instructions. An incorrect NLS_LANG setting can result in corrupted data.

On NT, make sure that your `oracle.ini` file includes the correct NLS_LANG setting.

3 ♦ 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 Progress databases you create are appropriate for the locale. For example, the following setting ensures that empty databases will 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.

4 ♦ Create a schema holder. See the "Creating a Schema Holder" section for instructions. Make sure that the code page of the empty database matches the code page that the Progress client uses. Setting DLCDB addresses this if you are working in a single-language environment.

5 ♦ 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 double-byte enabled applications or for applications running in many non-Western European locales.) Specify the Progress name for the equivalent ORACLE code page in the Code Page field. Table 3–5 lists the supported double-byte languages and the Progress names for their code pages.

Table 3–5: Supported DBE Code Pages

Language	Progress Code Page
Japanese	Shift-JIS (NT) EUCJIS (UNIX)
Korean	CP949 CP1361 KSC5601
Simplified Chinese	CP950 GB2312
Traditional Chinese	BIG-5 CP950

6 ♦ Run your Progress application.

NOTE: You must specify the Internal Code Page (-cpinternal) and Stream Code Page (-cpstream) parameters when you start the Progress client. The values that you specify for these parameters much match the code page that the ORACLE database uses. For example, if your ORACLE database uses EUCJIS, specify EUCJIS for -cpinternal and -cpstream.

On NT, run this command to run a procedure against a local instance of ORACLE:

```
prowin32 -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -U userid -P password
  -p procedure-name
```

On NT, run this command to connect to and run a procedure in batch mode against a remote instance of ORACLE through SQL*Net or Net 8:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -U userid@service-name
  -P password -b -p procedure-name
```

On UNIX, run this command to connect to and run a procedure against a local instance of ORACLE:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -1 -db oracle-dbname
  -ld oracle-logical-dbname -U userid -P password
  -p procedure-name
```

On UNIX, run this command to connect to and run a procedure against a remote instance of ORACLE through SQL*Net or Net 8:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder -1 -db oracle-dbname
  -ld oracle-logical-dbname -U userid@service-name
  -P password -p procedure-name
```

On UNIX, run this command to connect to and run a procedure in batch mode against a remote instance of ORACLE through Progress networking after starting the broker on the host machine:

```
_progres -cpinternal code-page -cpstream code-page
  schema-holder-name -db oracle-dbname
  -ld oracle-logical-dbname -S service-name -H host
  -N network -U userid -P password -b -p procedure-name
```

3.6 Configuring the DataServer in the Explorer Administration Framework

Integrating a DataServer into the Progress Explorer administration framework mainly affects only how you configure the remote DataServer broker. However, 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 the *Building Distributed Applications Using the Progress AppServer* manual and the *WebSpeed Installation and Configuration Guide*.

3.6.1 Configuring the Explorer Administration Framework on NT

On NT, you use the Progress Explorer tool to configure the DataServer for ORACLE in the Explorer administration framework. The Explorer has a default broker already defined, orabroker1. Follow these steps to define and configure:

1 ♦ Make sure that the Progress AdminServer is running.

2 ♦ Start the Progress Explorer.

From the Windows Desktop choose Start→Programs→Progress→Progress Explorer Tool. The Progress Explorer appears in the MMC framework.

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 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. See the Progress Explorer online help for more information.

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, application service properties, logging settings, environment variables, and others. For details on these settings, see the Progress Explorer online help.

Two important considerations when configuring the ORACLE DataServer:

- In the Property Editor Server category, the default pathname of the DataServer executable that the broker runs is `@{Startup\DLC\}bin\_orasrv`.
- In the Property Editor Environment Variables category, you must at least set the ORACLE_HOME and ORACLE-SID environment variables. See Table 3–4 for descriptions of these variables.

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

3.6.2 Configuring the 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 the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

When you configure the 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, and that the broker is spawning the appropriate DataServer process (`_orasrv` or a custom executable that you built with PROBUILD) and that it is running in the correct environment.

Table 3–6 describes the sections in the `ubroker.properties` file that apply to the 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 3–6: DataServer for ORACLE Sections

Section	Description
[UBroker.OR]	Parent entity of DataServer for ORACLE brokers. It defines default property settings for all DataServer for ORACLE Broker instances.
[UBroker.OR.orabroker1]	Sample property entity of a DataServer for ORACLE Broker instance. It defines property settings for the 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. Add any environment variables that you want to apply to the DataServer's environment to this section.

Here are the DataServer for ORACLE sections in the ubroker.properties file:

```
#  Default properties for broker instances serving Oracle DataServers
 #
[UBroker.OR]
     operatingMode=State-aware
     classMain=com.progress.ubroker.broker.ubroker
     initialSrvrInstance=0
     minSrvrInstance=0
     maxSrvrInstance=256
     loggingLevel=3
     srvrStartupParam=
     description=Oracle DataServer Broker
 #
 # Sample Oracle DataServer Broker definition
 #
 [UBroker.OR.orabroker1]
     srvrExecFile=$DLC/bin/_orasrv
     srvrStartupParam=-svub -S orabroker1 -N TCP -U X -P X -hs 0 -s 40
-cpinternal iso8859-1 -cpstream ibm850 -cpterm ibm850 -cpcase basic -cpcoll
basic
     srvrLogFile=$WRKDIR/orabroker1.server.log
     brokerLogFile=$WRKDIR/orabroker1.broker.log
     portNumber=4445
     description=A sample Oracle DataServer Broker #
 #
 # Environment for Oracle DataServer Broker: orabroker1
 #
 [Environment.orabroker1]
     ORACLE_HOME=/usr/orant
     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. 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 F, "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.

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.

3.7 Creating 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. Creating a schema holder involves the following steps:

1 ♦ Start an ORACLE instance for the supporting database.

2 ♦ Start the Progress client. If you are using a remote DataServer, you must start both the broker and client processes.

3 ♦ Create and connect an empty Progress database.

4 ♦ Create a schema holder.

NOTE: If you are accessing ORACLE7 and ORACLE8 databases, you must create a schema holder for each. The schema holders can exist in the same Progress database, however.

The following sections describe these steps in more detail.

3.7.1 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

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 Progress 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 3–7 describes the permissions that you need for creating a schema holder.

Table 3–7: Required ORACLE Permissions

Permission	Object
CREATE SESSION	Database
SELECT	System objects: sys.argument$ sys.col$ sys.com$ sys.con$ sys.dual sys.icol$ sys.ind$ sys.link$ sys.obj$ sys.procedure$ sys.seq$ sys.syn$ sys.tab$ sys.user$ sys.view$

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 or /USER) parameter to provide the user ID and the Password (-P or /PASSWORD) 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 a Progress 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) has to establish all user ID and password combinations within ORACLE with an ORACLE product, such as SQL*DBA.

3.7.2 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. Follow these steps to run ORACLE:

1 ♦ Set the ORACLE_HOME variable to the main ORACLE directory.

2 ♦ Set the ORACLE_SID variable to the ORACLE system identifier.

A process can have the ORACLE_SID set to only one value at a time. If you want to access multiple ORACLE databases on the same machine, set an ORACLE_SID variable to a different value for each database. Then create a separate broker for each database with the ORACLE_SID variable set to the matching value for each database. For more information, see the ORACLE documentation for your machine type.

3 ♦ Start an ORACLE instance for the target ORACLE database.

To start the ORACLE RDBMS, start the ORACLE SQL*DBA utility by entering sqldba at the system prompt. Then start the ORACLE RDBMS with the CONNECT and STARTUP commands as documented in the *ORACLE RDBMS Database Administrator's Guide* or *ORACLE7, The Utilities User's Guide*. If you use the graphical interface for the SQL*DBA utility, select Session→ Connect and Instance→ Start Up from the main menu.

3.7.3 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 4, "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 SQL*Net or Net 8.

Remote DataServer on NT — UNIX Client

On the host machine, use ProControl to start the broker you configured for the ORACLE DataServer.

On the client machine, run the client executable (`_progres`).

Remote DataServer on NT — Windows Client

On the host machine, use ProControl to start the broker you configured for the ORACLE DataServer.

On the client machine, run the client executable (`prowin32.exe`).

Local DataServer on UNIX

Run the client executable (`_progres`).

Remote DataServer on UNIX — UNIX Client

On the host machine, enter this 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 -N network
```

On the client machine, run the client executable (`_progres`).

Remote DataServer on UNIX — Windows Client

On the host machine, enter this 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 -N network
```

On the client machine, run the client executable (`prowin32.exe`).

Local DataServer on Windows

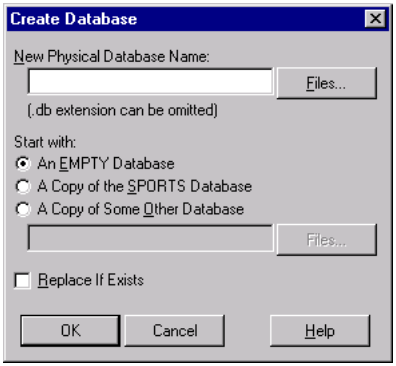
Run the client executable, `prowin32.exe`.

3.7.4 Creating an Empty Progress 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 Dictionary. See the *Progress Database Administration Guide and Reference* for other techniques. Follow these steps to create and connect an empty Progress database on your client machine. This example uses Windows, but you can follow the same steps if you are using Progress with a character-based interface, since the screens prompt for the same information:

1 ♦ Start Progress with no database connected and access the Data Dictionary. The Dictionary Startup dialog box appears.

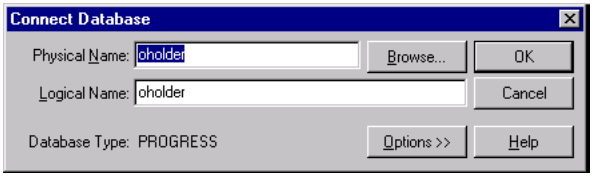
2 ♦ Select the Create a New Database option, then choose OK. The Create Database dialog box appears:



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 Database Connect 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. You can add connection parameters when you create the database or edit connection information later. See the *Progress Database Administration Guide and Reference* for a complete description of the Database Connect dialog box.

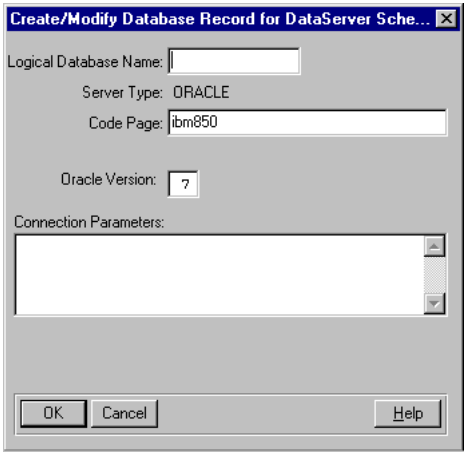
6 ♦ Choose OK to connect the empty Progress database and return to the Data Dictionary main window.

3.7.5 Using the DataServer Utility to Create a Schema Holder

Once you have started ORACLE, the DataServer processes, and created and connected to a local empty Progress database, you can create the schema holder. This example uses Windows; the character-based interfaces have the same sequence of screens. Follow these steps to create the schema holder:

1 ♦ From the Data Administration main menu, select DataServer→ORACLE Utilities→Create DataServer Schema. The following dialog box appears.

If you are using character-based Progress, select DataServer→ORACLE Utilities→Create DataServer Schema from the Data Dictionary main menu.



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 the *Progress Programming Handbook*.

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-Progress 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 Progress name for the code page that the ORACLE Call Interface (OCI) uses.

If your ORACLE environment uses a code page that Progress does not support, you must supply a conversion table that translates between the Progress client code page and the code page that ORACLE uses.

See the "Code Pages" section in Chapter 2, "Programming Considerations," for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see the *Progress Internationalization Guide*.

4 ♦ In the Oracle Version field, type the version number of ORACLE you are using. Possible values are 7 (the default) or 8.

5 ♦ Type the required connection parameters in the Connection Parameters field.

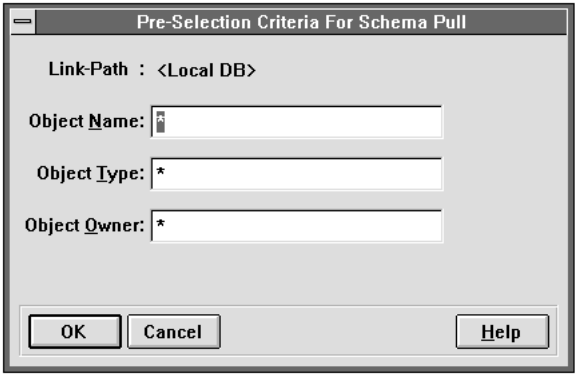
At a minimum, you need the User ID (-U) and Password (-P) parameters. Remote connections also require networking parameters. See the "Connecting a Schema Holder at Startup" section in Chapter 4, "Connecting the DataServer," for a description of the required and optional connection parameters. This is how you identify the ORACLE user in a SQL*Net or Net 8 connection:

```
-U userid@service-name -P password
```

6 ♦ Choose OK. The utility prompts you to verify the ORACLE user ID and password.

7 ♦ 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. Table 3–7 lists these permissions.

8 ♦ Choose OK. The following dialog box appears:



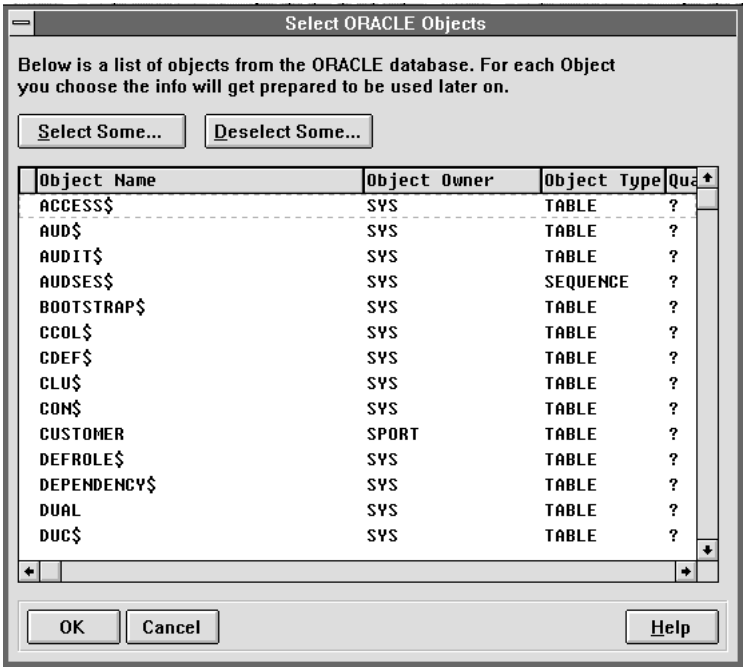
9 ♦ Preselect the objects that the schema holder should contain. You can select these objects by object name, object type, or owner. By default, the wildcard symbol (*) appears and specifies that the utility selects all objects. Typically, you should not include system-owned objects.

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 you database has two INVOICE tables, the schema holder 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 Progress 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 Progress Data Dictionary. For example, you can choose to use an alternate integer index or the native ROWID. See the “Defining the

ROWID" section in Chapter 5, "The DataServer Tutorial," for instructions. See the "ROWID Function" section in Chapter 2, "Programming Considerations," 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.

3.7.6 Maintaining a Schema Holder

The Progress Data Administration tool provides a set of ORACLE utilities that you can use to maintain a schema holder. Chapter 5, "The DataServer Tutorial," describes these utilities.

If you are using non-Progress 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 3–8 lists common database operations and the required techniques.

Table 3–8: Database Operations and the Schema Holder *(1 of 2)*

Database Operation	Technique
Modified a column in an ORACLE table	Update the schema image by using the DataServer for ORACLE utilities.
Added a column to an ORACLE table	Update the schema image by using the DataServer for ORACLE utilities.
Inserted a row in an ORACLE table	Use the ORACLE sequence generator to generate a value. Assign that value to the Progress_RECID column for that row. If you do not use the sequence generator, Progress might fail when it tries to access that table.

Table 3–8: Database Operations and the Schema Holder *(2 of 2)*

Database Operation	Technique
Added an index to an ORACLE table	If the index you add is for a field that is case insensitive, be sure to add the appropriate value in the shadow column named U##*column*. For example, if the indexed column contains the value uk, assign the value UK to the shadow column to ensure a case-insensitive index.
Modified data definitions in the schema image	Update the ORACLE database to reflect your changes by using the Incremental Schema Migration utility.

3.7.7 Deploying a Schema Holder

The guidelines and techniques that apply to deploying a Progress 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 4GL 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 and Add Table Definitions utility.
- Send a new data definitions file for the schema holder. Your users can load the .df file into an empty Progress database. Consider any CRC issues.

Follow these guidelines to maintain a deployed DataServer application:

1 ♦ Make changes to the ORACLE database.

2 ♦ Run the Update/Add ORACLE 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 in Chapter 2, "Programming Considerations," for more information.

3.7.8 Setting Up a Schema Holder As an NT Service

The schema holder can reside on the NT host or on the client machine. A schema holder on a client machine (a local schema holder) performs better. However, a schema holder on the NT 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 ProService (that is, a native NT 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 on an NT host and make it accessible to multiple users, follow these steps:

1 ♦ Access the Data Administration tool by double-clicking its icon in the Progress program group.

2 ♦ Create an entry for the schema holder using ProControl. Creating an entry in ProControl for a Progress database allows it to run as a ProService (a native NT service).

3 ♦ Access ProControl by double-clicking its icon in the NT Control Panel.

4 ♦ Choose Detail. Tabbed folders appear.

5 ♦ Choose the Databases tab and choose Insert. The General, DB-Writers, and Environment tabbed folders appear.

6 ♦ Enter the appropriate information in the following fields in the General folder:

- **Identifiers** — Provide a unique name to identify the service. The name must be unique to the host. The identifier can be up to 30 characters long. It does not have to be the same name as the schema holder.
- **Executable Name** — By default, the `_mprosrv.exe` executable displays. If you want to use a customized Progress server executable that you built using the PROBUILD utility, specify it here.
- **Working Directory** — Specify the directory that the service uses, for example, to create the before-image file.
- **Physical DB-Name** — Specify the schema holder's physical name.
- **Startup Parameter** — Specify any optional parameters, such as -RO.

7 ♦ Entering information in the DB-Writers and Environment folders is optional.

4

Connecting the DataServer

This chapter provides instructions for running the DataServer after you have configured its modules and created a schema holder. Specifically, it describes how to:

- Start the DataServer processes
- Connect to a schema holder
- Troubleshoot connection failure

4.1 Starting the Local DataServer

Starting the Progress client starts the local DataServer. You include information about the ORACLE database, user ID, and password in the startup command.

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. Table 4–1 describes how to set them.

Table 4–1: Environment Variables for the Local DataServer

Variable	How to Set It
DSLOGDIR	The pathname to the log file that Progress will use to keep track of DataServer processes and error messages.
ORACLE_HOME	The pathname of the directory where you installed ORACLE.
ORACLE_SID	The identifier for the ORACLE instance you are accessing.
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	The identifier of the shared ORACLE libraries. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. You must set this environment variable to include `$ORACLE_HOME/lib` when you run executables that you built with PROBUILD to access ORACLE7 or ORACLE8.

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:

Local DataServer — UNIX

```
pro schema-holder-name -db fill-char -dt ORACLE
  -ld oracle-logical-db-name -U userid -P password
```

Local DataServer — NT or Windows

```
prowin32 schema-holder-name -db fill-char -dt ORACLE
  -ld oracle-logical-db-name -U userid -P password
```

For example, the following command starts Progress with the DataServer on a UNIX machine, connects a local schema holder oholder in single-user mode, and connects an ORACLE database orademo with a user scott and the password tiger:

```
pro oholder -db oradb -dt ORACLE -ld orademo -U scott -P tiger
```

For SQL*Net or Net 8 startup information, see the "Connecting Through SQL*Net or Net 8" section.

4.2 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 a Progress 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.

4.2.1 Starting the DataServer in the Explorer Administration Framework

The Progress Explorer administration framework administers only the server module (_orasrv on NT and UNIX) of the DataServer. After starting the server, start your Progress client as you would in any remote DataServer configuration. The following sections describe how to run the DataServer in the Explorer administration framework on NT and on UNIX. The Explorer administration framework is not supported on Windows.

On the NT Host

Use the Progress Explorer tool to run DataServer processes on the NT host. Optionally you can use the command-line technique described in the "On the UNIX Host" section. Follow these steps to start and shut down the DataServer:

1 ♦ In the Progress Explorer tool, configure the DataServer for ORACLE broker as described in the "Configuring the DataServer in the Explorer Administration Framework" section in Chapter 3, "Configuring the DataServer."

2 ♦ 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.

3 ♦ Select the DataServer broker you want to start and right-click. A pop-up menu appears.

4 ♦ Choose the Start option from the pop-up menu. The Status in the right pane changes to Running for that DataServer broker.

5 ♦ To edit the ORACLE DataServer broker configuration, select the DataServer broker and right-click. Choose the Properties option from the pop-up menu. See the Progress Explorer online help for more information on editing an ORACLE broker configuration.

6 ♦ To check the status of an ORACLE DataServer broker, select the DataServer broker and right-click. Choose the Status option from the pop-up menu. See the Progress Explorer online help for more information the various Status tabs.

7 ♦ To stop the broker, select the DataServer broker and right-click. 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. Follow these steps to start and shut down the DataServer:

1 ♦ Configure the DataServer for ORACLE broker as described in the "Configuring the DataServer in the Explorer Administration Framework" section in Chapter 3, "Configuring the DataServer."

2 ♦ 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:

```
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.

3 ♦ 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.

You can also use the `oraman` utility to check the status of a broker. See Appendix F, "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 ODBC data source 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.
- Set the -S parameter to a service name whose associated port matches the port number assigned to the broker instance that you started in the Explorer. On NT, the Explorer Property Editor, General in the Broker category lists the broker instance's port number.

On UNIX, see the `ubroker.properties` file. Note that this -S setting differs from a standard remote connection where you set the -S parameter to the service that you used when starting `probrkr`.

- Set the -H parameter to the name of the machine where the broker instance is running.

4.2.2 Starting and Stopping the NT Broker Process

On an NT host machine, you can use the ProControl utility to start and stop DataServer modules, if you used ProControl to configure a broker, or you can issue commands at the MS-DOS prompt.

Follow these steps to run the DataServer modules on NT using ProControl:

1 ♦ Start an instance for the supporting ORACLE database.

2 ♦ Start the broker by selecting it from the selection list and choosing Start on the DataServers folder. The broker spawns the DataServer process when a client requests a connection.

3 ♦ Stop the broker by choosing Stop on the DataServers folder.

Follow these steps to run the DataServer modules from the MS-DOS prompt:

1 ♦ Set the environment variables described in Table 4–2.

2 ♦ Start the broker by entering this command at the MS-DOS prompt:

```
_probrkr -H host-name -S service-name -N TCP
```

3 ♦ Stop the broker by entering this PROSHUT command:

```
proshut -Gw -H host-name -S service-name -N TCP
```

4.2.3 Starting and Stopping the UNIX Broker Process

Follow these steps to start the broker process on the UNIX host machine:

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 4–2 describes how to set them.

Table 4–2: Environment Variables for the Remote DataServer

Variable	How to Set It
DSLOGDIR	The pathname to the log file that Progress uses to keep track of DataServer processes and error messages.
ORACLE_HOME	The pathname of the directory where you installed ORACLE.
ORACLE_SID	The identifier for the ORACLE instance you are accessing.
ORASRV	The name of the executable (including the path) of the ORACLE DataServer.
PROBRKR	The pathname to the executable of the broker.
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	The identifier of the shared ORACLE libraries. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. You must set this environment variable to include `$ORACLE_HOME/lib` when you run executables that you built with PROBUILD to access ORACLE7 or ORACLE8.

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 on your UNIX host machine. Select the *service-name* from the list of available services in the `/etc/services` file for your host:

```
brokername -H host-name -N network-type -S service-name
```

For example, the following command uses the default broker executable. The service name demosv is a service listed in the /etc/services file:

```
_probrkr  -H paris -N TCP -S demosv
```

4 ♦ 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 -N TCP
```

4.2.4 Starting the UNIX Client Process

After starting the remote broker, you start the Progress 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 Progress client and connects to a read-only schema holder named oholder for an ORACLE database named oradem o:

```
pro oholder -RO -db oradb -dt ORACLE -ld oradem o -H host1 -S oserviceA
  -N TCP -U scott -P tiger
```

See the "Connecting a Schema Holder at Startup" and "Optional Connection and Startup Parameters" sections for descriptions of the required command line.

4.2.5 Starting the NT or Windows Client Process

Start the Progress client process on your Windows machine by running the prowin32.exe executable. The Windows executable includes support for the DataServer. You can create a program icon for the Progress 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.

This is a sample command line:

```
prowin32 oholder -RO -db oradb -dt ORACLE -H host1 -S oserviceA
  -N TCP -U scott -P tiger
```

See the "Connecting a Schema Holder at Startup" and "Optional Connection and Startup Parameters" sections for descriptions of the required command line.

4.3 Connecting a Schema Holder

In addition to the automatic connection that can take place when you create a schema holder, Progress provides these techniques for connecting a schema holder:

- Use the Progress 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. Once you have supplied the connection parameters, you can use Auto-Connect to connect to the schema holder. You cannot use Auto-Connect to connect to the ORACLE database. 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 the *Progress Language Reference*). For example, this command connects a local read-only schema holder named oholder.

```
CONNECT oholder -RO -db oradb -dt ORACLE -ld oradem0 -S oserviceA
  -H doc4 -N TCP -U scott -P tiger.
```

- Use connection parameters when you start Progress. You can include these parameters in a parameter file that you specify when you start Progress. A parameter file is portable and easy to maintain. For information on how to create a parameter file, see the *Progress 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 the Progress CONNECT statement. Any combination of connection techniques works, as long as you follow these steps:

1 ♦ Connect the schema holder.

2 ♦ Connect the ORACLE database. 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 Progress.

4.3.1 Connecting a Schema Holder at Startup

Progress supports connection parameters that you can use to connect the Progress schema holder and the ORACLE database. These parameters control how your system connects to a database. When the DataServer for ORACLE runs in a remote configuration, your startup command or parameter file must include parameters that control networking options.

Table 4–3 describes the connection parameters that relate to the remote DataServer and networking options.

Table 4–3: Remote DataServer Connection Parameters

Parameter	Description
Host Name (-H)	Indicates the name of the host machine in the network.
Network Type (-N)	Indicates the network type. TCP is the only possible value.
Service Name (-S)	Indicates the name of the service you are calling. The service you are calling is the broker on the host machine. Use the same name you used when you started the broker.

The following command starts Progress for NT or Windows:

- In single-user mode
- In a remote-DataServer configuration
- With a local schema holder connected
- With an ORACLE database connected

```
prowin32 schemaholdername -db fill-char -dt ORACLE
  -ld oracle-logical-db-name -H hostname -S service-name
  -N TCP -U userID -P password
```

For example, the following command starts up Progress for Windows in a remote DataServer configuration, with a read-only schema holder and an ORACLE database connected.

Remote DataServer — NT or Windows Client

```
prowin32 oholder -RO -db oradb -dt ORACLE -ld orademo -H host1
  -S oserviceA -N TCP -U scott -P tiger
```

In this example, the schema holder's physical name is oholder; it is read-only; the database type is ORACLE; the ORACLE database's logical name is orademo; the host name is host1; the service name is oserviceA; the network type is TCP/IP; the user ID is scott; the password is tiger.

Local DataServer — NT or Windows

```
prowin32 oholder -RO -db oradb -dt ORACLE -ld orademo -U scott -P tiger
```

Remote DataServer — UNIX Client

```
_progres oholder -RO -db oradb -dt ORACLE -ld orademo -H host1
  -S oserviceA -N TCP -U scott -P tiger
```

Local DataServer — UNIX

```
_progres oholder -RO -db oradb -dt ORACLE -ld orademo -U scott -P tiger
```

Table 4–4 describes the parameters required for connecting to a schema holder and an ORACLE database.

Table 4–4: Required Connection Parameters *(1 of 2)*

Parameter	Description
Internal Code Page (-cpinternal)	Specifies the name of the code page that Progress uses in memory. This code page must match the code page that the ORACLE database uses. Required when using a code page other than ibm850 or iso-1, such as in a DBE DataServer configuration.
Stream Code Page (-cpstream)	Specifies the name of the code page that Progress uses for stream I/O. This code page must match the code page that the ORACLE database uses. Required when using a code page other than ibm850 or iso-1, such as in a DBE DataServer configuration.
Password (-P)	Supplies the password that the DataServer for ORACLE uses to log into the ORACLE RDBMS.
Physical Database Name (-db)	For Progress databases, the string that follows this parameter is the physical name of the database you want to connect. However, for ORACLE databases, this physical database name can be any fill characters. For example, use the logical database name after the -db parameter or any other designation, such as oradb.

Table 4–4: Required Connection Parameters *(2 of 2)*

Parameter	Description
Single-user Mode (-1) or Read-only (-RO)	Specifies that a database is used in single-user mode. When you connect the schema holder, you must connect either in single-user mode or as read-only. Specifies that a schema holder is read-only. Connecting a schema holder as read-only increases processing speed at compile time. It also allows multiple client processes on the same machine to access the schema holder without starting additional server processes.
User ID (-U)	Supplies the user ID that the DataServer for ORACLE uses to log into the ORACLE RDBMS. SQL*Net and Net 8 allow you to pass the user ID and password as a single string to the -U parameter. If you specify `-U userid/password@service-name`, the USERID function returns the password information in addition to the user ID and service.

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 the *Progress Startup Command and Parameter Reference*.

4.3.2 Connecting Through SQL*Net or Net 8

SQL*Net and Net 8 are ORACLE network interfaces that allow you to access an ORACLE RDBMS instance running on a remote machine. You do not need any Progress-supplied software on the machine where ORACLE is running. However, you need SQL*Net or Net 8 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 SQL*Net or Net 8, 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.

Follow these steps to connect to an ORACLE instance through SQL*Net or Net 8:

1 ♦ Make sure that the remote ORACLE instance is running.

2 ♦ Make sure that SQL*Net or Net 8 is running on the client machine.

3 ♦ Supply the following connection parameters in your Progress startup command or parameter file. The SQL*Net and Net 8 connection parameters require that you specify the identifier you configured in the `tnsnames.ora` file for the entry that points to the ORACLE instance you are accessing. SQL*Net and Net 8 also require that the client process have read access to the `tnsnames.ora` file in the `$TNS_ADMIN` directory, which is indicated by the TNS_ADMIN variable in the `oracle.ini` file. See the ORACLE documentation for information on configuring `tnsnames.ora`.

NT or Windows Client — SQL*Net or Net 8

```
prowin32 schema-holder-name -db oracle-dbname
  -U userid@service-name -P password
```

UNIX Client — SQL*Net or Net 8

```
pro schema-holder-name -db oracle-dbname
  -U userid@service-name -P password
```

For example, the following command starts Progress on a UNIX machine, connects a local schema holder oholder in read-only mode and connects the remote ORACLE instance orademo with the ORACLE_SID X and uses the srvr1 service. The username is scott and the password is tiger. The -dt and -ld parameters are optional:

```
pro oholder -RO -db oradb -dt ORACLE -ld orademo
  -U scott@oserviceA -P tiger
```

If you are connected through SQL*Net or Net 8, the Progress USERID function returns `scott@oserviceA`, instead of just `scott`.

If you use a sqlnet connection string, it must be part of the username. You can use either of these forms:

```
CONNECT <logical name> -U <user@sqlnetstring> -P <password>.

CONNECT <logical name> -U <user/password@sqlnetstring>
```

You should not use the sqlnet connection string as part of the password, as in the following example:

```
CONNECT <logical name> -U <user> -P <password@sqlnetstring>
```

4.3.3 Unsupported Connection Parameters

You cannot use the following Progress connection parameters when connecting to an ORACLE database through the DataServer for ORACLE:

- Blocks in Database Buffers (-B)
- Before-image File Name (-g)
- Lock Table Entries (-L)
- Number of Users (-n)
- Buffered I/O - UNIX only (-r)
- Version 6 Query (-v6q)

4.3.4 Optional Connection and Startup Parameters

You can use other Progress startup and connection parameters with the DataServer. Table 4–5 describes parameters that are especially relevant to using the DataServer for ORACLE.

Table 4–5: Optional Connection and Startup Parameters *(1 of 2)*

Parameter	Description
Database Type (-dt ORACLE)	Specifies that the non-Progress type of the target database is ORACLE.
DataServer (-Dsrv)	Specifies parameters that allow you to control how the DataServer processes queries. -Dsrv is a connection parameter. See the "Query Tuning with Connection and Startup Parameters" section for more information.
Logical Database Name (-ld)	Specifies the logical name for the ORACLE database. This is the name by which you refer to the database in your applications.

Table 4–5: Optional Connection and Startup Parameters *(2 of 2)*

Parameter	Description
Index Cursors (-c)	Specifies the maximum number of ORACLE cursors per user that Progress 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 for more information.
Schema Cache File (-cache)	Specifies the name of a binary file that contains the schema for a database. Reading schema from this file is more efficient than reading from a remote schema holder. See the "Local Schema Caching" section for more information.

4.3.5 Query Tuning with Connection and Startup Parameters

In addition to controlling aspects of how the DataServer handles queries programmatically within 4GL 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 NO-DEBUG in a query, specifying qt_debug, SQL at connection overrides the NO-DEBUG default, but does not override the NO-DEBUG option that you specified for the query. The DataServer writes 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 in Chapter 2, "Programming Considerations," for more information.

You override query-tuning defaults with the DataServer (-Dsrv) connection parameter when you connect to the ORACLE database. This is the syntax:

SYNTAX

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

Table 4–6 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 4–6: Connection Query-tuning Options *(1 of 2)*

Option	Description
qt_bind_where qt_no_bind_where	Specifies whether the DataServer uses ORACLE bind variables for values in WHERE clauses. Specify qt_no_bind_where to use literals. Use at run time only. Default: qt_bind_where
qt_cache_size *integer* QT_BYTE qt_cache_size *integer* QT_ROW	Specifies the size of the cache (in bytes or records) for information used by lookahead or standard cursors. Byte maximum: 65535 bytes Byte minimum: Specify the number of bytes contained in a single record. For joins, specify the number of bytes contained in two joined records. If a join returns a 500-byte record, you need a cache of 1000 bytes. Row maximum: The number of records that can fit in 65535 bytes. See the "Caching Records" section in Chapter 2, "Programming Considerations" for more information. Row minimum: 1. If the server performs the join, the minimum is 2. Default: 1024 bytes with standard cursors; 8192 with lookahead cursors

Table 4–6: Connection Query-tuning Options *(2 of 2)*

Option	Description
qt_debug,SQL qt_debug,EXTENDED qt_no_debug	Specifies whether the DataServer should print debugging information that it generates for the query to the `dataserv.lg` file. Specify qt_debug,SQL to print the SQL that the DataServer executes against the ORACLE DBMS. Specify qt_debug,EXTENDED to print information in addition to the SQL statements executed by the DataServer, such as cursor statistics. Specify qt_debug,*option* to override the NO-DEBUG default. In addition to SQL and EXTENDED, there are other options that can assist you in analyzing performance. See Table 4–8. Default: qt_no_debug
qt_lookahead qt_no_lookahead	Specifies whether the DataServer uses lookahead or standard cursors. Specify qt_no_lookahead for behavior that is consistent with Progress. Default: qt_lookahead, except with statements that use an EXCLUSIVE lock

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:

```
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 4–7 describes these startup parameters.

Table 4–7: Query-tuning Startup Parameters

Startup Parameter	Description
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 (-nojoinbysqldb)	Specifies that the client evaluates and performs queries that have joins. This might slow performance, but provides results that are consistent with Progress behavior. Use -nojoinbysqldb at compile time. It has no effect at run time.

4.3.6 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.lg` log file. The qt_debug option provides extended diagnostic capabilities that you can use to determine which parts of your application might be causing additional network traffic or processing by ORACLE. Table 4–8 lists the diagnostic capabilities of qt_debug. Note that the log file uses the numeric values (in parentheses) to identify information that a specific option generates.

Table 4–8: Diagnostic Options

Option	Description
CURSOR (18835)	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 Progress and ORACLE maximum cursor settings or cursor "leaks" that your application might have.
DATA-BIND (1029)	Information about the data types, buffer sizes, and addresses that the DataServer and ORACLE use when binding variables.
PERFORMANCE (55775)	Information on the time certain operations take. These statistics are available only for some platforms. Note that any time differences between what the DataServer reports and what the ORACLE tkprof utility reports might be due to network performance issues, rather than to DataServer or ORACLE behavior.
SUMMARY (1073797455)	Information on cursors and timing in summary form as a Progress data (.d) file. Call Progress Software Corporation Consulting for assistance with this file. This option is not available as a QUERY-TUNING phrase option.
VERBOSE (122879)	Information on all of the above. This option generates a very large log file. Remember to clear your log file before using this option to test a procedure.

NOTE: Turning on debugging options decreases DataServer performance. Be sure to turn off debugging options when running DataServer applications in production mode.

This connection statement causes the DataServer to report on the time that certain operations take:

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

4.3.7 Index Cursors

The Progress 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. Progress 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. If your `init.ora` file does not contain an OPEN_CURSORS parameter, then the ORACLE default is 50. The valid range for numbers of cursors varies depending on the version of ORACLE and system configuration.

The Progress default maximum number of ORACLE open cursors for the DataServer is 50 also. When you use the -c parameter to set the maximum number of cursors, you cannot exceed the number that your `init.ora` file specifies. For example, if the ORACLE OPEN_CURSORS parameter is set to 250, then you can set the upper limit for maximum open cursors open to 250 with the -c parameter.

Determining the optimal number of cursors for your application involves balancing memory consumption, performance, and possible application failures. Use the -Dsrv qt_debug,EXTENDED parameter to log information on how many cursors your application uses. The following excerpt from the `dataserv.lg` file shows the number of cursors the DataServer uses as the cc value after the open calls:

```
OCI call oopen  <42> cc = 0
CI call osql3  <42>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <43> cc = 1
 OCI call osql3  <43>
     SELECT * FROM SPORTS.ORDER_ WHERE progress_recid = :rid
 OCI call oopen  <44> cc = 2
```

Avoid setting the -c parameter too low or too high:

- **Too low** — Your application fails because it opens too many queries, nests FOR EACH loops too deeply, or uses too many FIND statements that reference different indexes. The DataServer returns an error message suggesting that you set -c. A low setting can also cause unnecessary recompiles of SQL which hurts performance.
- **Too high** — You allocate all available cursors, including the cursor that the ORACLE DBMS needs for internal purposes. The DataServer returns a recursive SQL error. See how many cursors your application uses and set -c to a slightly lower number.

4.3.8 Local Schema Caching

The ability to store schema definitions in a local file allows you to access them more quickly. Once you create a local schema cache, you have to connect to the schema holder only when you compile applications. You no longer rely on having a local schema holder for maximum performance. Running DataServer applications with a local schema cache instead of a schema holder also results in better performance.

The Progress 4GL SAVE CACHE COMPLETE statement creates a binary file that contains the entire schema for a Progress database. Use this syntax to create a cache file for a connected schema holder:

```
SAVE CACHE COMPLETE schema-holder-name TO filename.
```

For example, the following statement creates a cache file named ocache for the oholder schema holder:

```
SAVE CACHE COMPLETE oholder TO ocache.
```

To use the cache file for a schema holder, specify the -cache parameter and the cache filename when you connect to the schema holder. For example, the following CONNECT statement connects the ORACLE database with the schema holder and tells Progress to use the cache file:

```
CONNECT oholder -RO -db oradb -dt ORACLE -ld orademo -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 the *Progress Programming Handbook* and the SAVE CACHE Statement reference entry in the *Progress Language Reference*.

4.4 Failures and Progress Responses

Table 4–9 lists circumstances under which a connection might fail and describes how Progress responds.

Table 4–9: Failure Responses *(1 of 2)*

Failure Circumstances	Results
During startup.	The system displays an error message and returns to the operating system prompt.
During a CONNECT statement.	The system aborts the remainder of the CONNECT statement (as though the remainder never existed) and a run-time error condition occurs. Any connections you made previous to the failed connection remain in effect. You can use the NO-ERROR modifier with the CONNECT statement to trap run-time errors. If you use the NO-ERROR modifier and it fails, you see the same failure behavior as you do with an unsuccessful CONNECT statement. However, run-time error conditions do not occur.
During an attempted auto-connect.	The system aborts the remainder of the connect (as though the remainder never existed) and a run-time error condition occurs. Any connections you made previous to the failed connection remain in effect. There is no way to trap auto-connect run-time error conditions.
During an attempt to connect using the Progress Data Dictionary.	The Data Dictionary displays an error message and returns to the main window.
During an attempt to connect a connected database with a different logical name.	The system responds with a run-time error condition and you can continue. You can use the NO-ERROR modifier to suppress the error.
During an attempt to connect a connected database with the same logical name.	The system responds with a warning and you can continue. You can use the NO-ERROR option to suppress the warning.
During an attempt to connect an unconnected database, when the logical name is already in use by a connected database.	The system responds with a run-time error condition and you cannot connect to the second database.

Table 4–9: Failure Responses *(2 of 2)*

Failure Circumstances	Results
During an attempt to create a schema holder for an ORACLE database.	The system responds with an error that an ORACLE system table does not match the schema holder.
During an attempt to use ORACLE utilities on an ORACLE database.	The system responds with an error that an ORACLE system table does not match the schema holder.
During an attempt to connect to an ORACLE database that uses a double-byte character set.	The system responds with an error that the code page of the database and the -cpinternal value differ. You also receive messages about missing conversion tables.

4.4.1 Connection Troubleshooting

Some reasons that a connection attempt to an ORACLE database might fail are:

- The Progress 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. If you have three unsuccessful connection attempts in a row, ORACLE does not allow any more connection attempts during the remainder of the Progress session. If this happens, quit your Progress session and restart it to make another connection attempt.
- 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.

4.4.2 Accessing the DataServer Log

Progress 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 Windows clients, it has information about the `ORACLE DLL`. For each process, the log provides the process identification number, physical database name, database type, user ID, and a message about the process' status. Here is an excerpt from a sample DataServer log file:

```
    Fri May 15 11:40:33 1997
11:40:33  (pid 2361) ORACLE SERVER db: t user: odemo
      Remote ORACLE Server begin. (message number)
11:40:33  (pid 2361) ORACLE SERVER db: t user: odemo
      Login to db as user odemo. (message number)
```

The query-tuning DEBUG option causes the DataServer to write information about the SQL that it generates to the `dataserv.lg` file as well.

To have access to the DataServer log file, follow these steps 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 don't set the environment variable, Progress writes the information to the `dataserv.lg` file in the process's current directory. If Progress cannot open this file, it writes the information to the `ds_<process-id>.lg` file in the process' current directory, and you will have one file per process.

2 ♦ Open the `dataserv.lg` file to read the DataServer log.

5

The DataServer Tutorial

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

- Create the ORACLE demonstration database
- Prepare to use the DataServer utilities
- Update a schema image
- Verify a schema image
- Change logical name and connection information in a schema image
- Modify a schema image to use Progress features
- Change the code page in a schema image
- Delete a schema image
- Migrate a Progress database to ORACLE
- Migrate a schema incrementally
- Adjust schema image

5.1 ORACLE Demonstration Database

The demonstration database allows you to run the sample procedures from this chapter, or your own procedures, against an ORACLE7 or ORACLE8 database. This demonstration database is not part of the Progress installation media. To create and initialize the ORACLE database, you use the Progress-to-ORACLE utility to migrate the Progress Sports database to the version of ORACLE you are using. Connect to the schema holder that the utility creates, then run the tutorial exercises or your own Progress applications against the ORACLE Sports database.

Follow these steps to migrate the Progress Sports database to ORACLE:

1 ♦ Make sure the ORACLE background processes are running for your ORACLE database.

2 ♦ Start the DataServer processes that your configuration requires:

- **Local configuration** — Start the Progress client.
- **Remote configuration** — Start the DataServer broker on the host machine, and the Progress client on the client machine.

See Chapter 4, "Connecting the DataServer," for the startup commands.

3 ♦ Connect to the Progress database that is to be migrated to ORACLE.

4 ♦ On Windows, from the Data Admin tool choose DataServer→ ORACLE Utilities→ Schema Migration Tools→ PROGRESS DB to ORACLE.

If you are using character-based Progress, access the Data Dictionary to access the DataServer utilities.

NOTE: The examples demonstrate the ORACLE utilities for Windows. If you are using Progress on UNIX, you will see slightly different interfaces that have the same components as these examples.

5 ♦ Provide the information noted in Table 5–1.

Table 5–1: Progress-to-ORACLE Utility *(1 of 2)*

Interface Element	Description
Original Progress Database	Accept the default value.
Connect parameters for Progress Db	Accept the default value.

Table 5–1: Progress-to-ORACLE Utility *(2 of 2)*

Interface Element	Description
Name of Schema holder Database	Enter **myholder**.
Logical name for ORACLE Database	Enter **mysports.**
What version of ORACLE	Enter the version of ORACLE you are using. Possible values are 7 or 8. The default is 7.
ORACLE Owner's Username	Enter the ORACLE database owner's name.
ORACLE User's Password	Enter the owner's password.
ORACLE connect parameters	Leave this field blank.
Code page for Schema Image	Accept the default code page.
Tables	Leave this field blank. The Sports database would not benefit from ORACLE tablespaces.
Indexes	Leave this field blank. The Sports database would not benefit from storing indexes in separate ORACLE tablespaces.
Progress 4GL Compatible Objects	Check this toggle box to create an ORACLE database that supports arrays, case-insensitive indexes, Find PREV/LAST, and the Progress record identifier.
Load SQL	Check this toggle box. In order to create the schema in ORACLE you must select the Load SQL option. If you leave the Load SQL option unselected only the SQL file is created.
Move Data	Check this toggle box so that the migrated Sports database contains data.

6 ♦ Choose OK. The utility creates the myholder database, adds ORACLE schema information to it, and connects you to the ORACLE database.

7 ♦ Choose DataServer→ ORACLE Utilities for a menu that allows you to access the DataServer utilities described in Table 5–2.

Table 5–2: DataServer for ORACLE Utilities Menu

ORACLE Utilities	
Create DataServer Schema...	Use this utility to create a schema image for an ORACLE database.
Update/Add Table Definitions...	Use this utility to update the schema image to reflect any changes you made to the ORACLE data definitions.
Verify Table Definition...	Use this utility to make sure that the data definitions in the schema image match your ORACLE data definitions.
Edit Connection Information...	Use this utility to change connection information or the logical database name in a schema image.
Change DataServer Schema Code Page...	Change the code page in a schema image.
Delete DataServer Schema...	Use this utility to delete a schema image.
Run ORACLE SQL *Plus...	Use this utility to modify your ORACLE database with SQL*PLUS commands.
Schema Migration Tools	Access utilities for migrating a Progress database to an ORACLE database and for incrementally migrating a schema to ORACLE and adjusting schema image.

8 ♦ 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 in Chapter 3, "Configuring the DataServer," for 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.

5.2 Updating a Schema Image

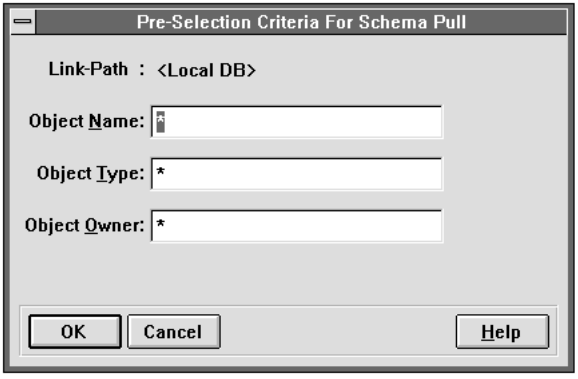
In this exercise, you update a schema image to reflect any changes you might have made to the data definitions for your ORACLE database.

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.

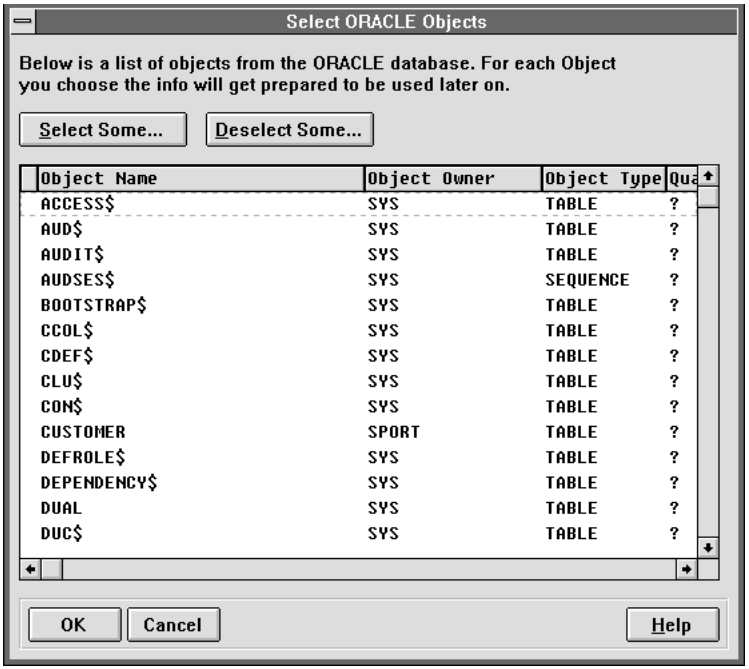
Follow these steps to update a schema image:

1 ♦ Choose DataServer→ ORACLE Utilities→ Update/Add Table Definitions. The following dialog box appears:



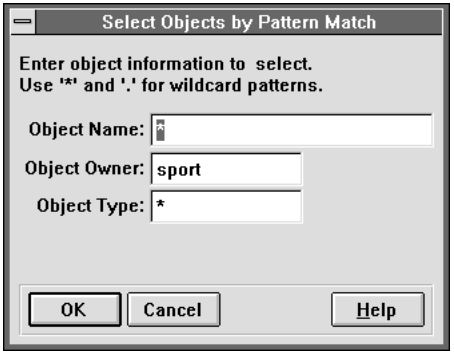
2 ♦ Preselect the ORACLE objects that the utility uses to update the schema image. By default, the wildcard symbol (*) appears in the fill-in fields. It specifies that the utility uses all objects in the ORACLE database, including system catalog information. You can change the criteria by typing new information in the fill-in fields.

3 ♦ Choose OK. A dialog box lists the objects and table information that you preselected:



4 ♦ Select the objects you want to update, then choose OK. Typically, you should not include system-owned objects.

5 ♦ To select tables by matching a pattern, choose the Select Some button. The following dialog box appears:



6 ♦ Type the pattern that you want to match, then choose OK. Progress returns to the main window.

When you update a definition, Progress overwrites the old definition with the new one based on the current ORACLE object. It also preserves the Progress-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 Progress-specific information for the previously existing columns (fields) in the definition.

5.3 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 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 solve 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 5–3 lists the differences that the utility detects.

Table 5–3: Verify Utility Report *(1 of 3)*

Database Object	Difference	Category
Table	Description	Retained
Table	Foreign type	Severe

Table 5–3: Verify Utility Report *(2 of 3)*

Database Object	Difference	Category
Table	Name in Progress	Retained
Table	Package name	Severe
Table	ROWID index	Retained
Index	Active	Minor
Index	Description	Retained
Index	Name in Progress	Retained
Index	Unique	Retained[1]
Index field	Abbreviated	Minor
Index field	Ascending	Severe
Index field	Order	Severe
Field	Case-sensitivity	Retained
Field	Decimals	Retained[2]
Field	Description	Retained
Field	Extent	Severe
Field	Initial value	Retained[2]
Field	Mandatory	Retained
Field	Name in Progress	Retained
Field	Order	Retained
Field	Progress data type	Retained[2]
Field	Progress format	Retained[2]

Table 5–3: Verify Utility Report *(3 of 3)*

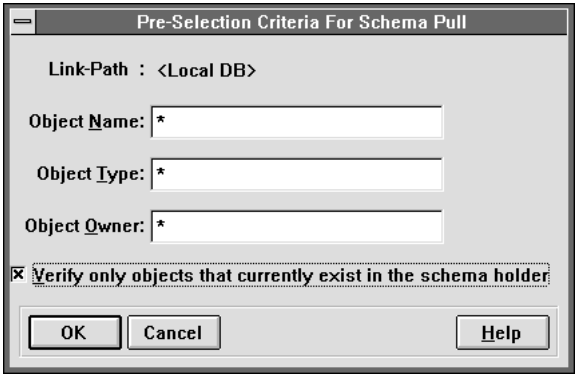
Database Object	Difference	Category
Field	Shadow-column Name	Severe

[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 Progress data type is CHARACTER, the data type information is not updated.

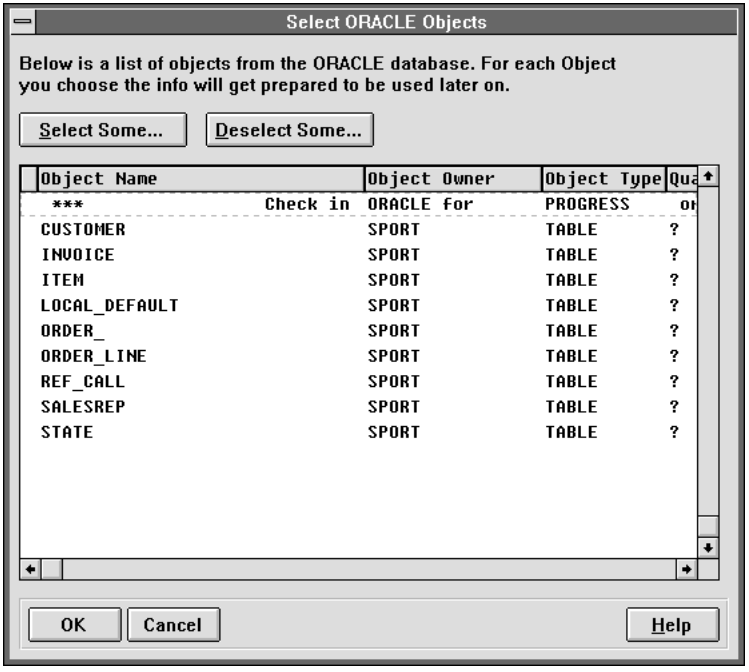
Follow these steps to verify a table.

1 ♦ Choose DataServer→ ORACLE Utilities→ Verify Table Definition. The following window appears:



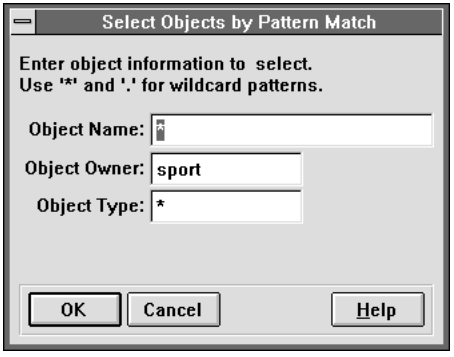
2 ♦ Preselect the ORACLE objects that the utility uses to update the schema image. By default, the wildcard symbol (*) appears in the fill-in fields. It specifies that the utility uses all objects in the ORACLE database, including system catalog information. You can change the criteria by typing new information in the fill-in fields.

3 ♦ Choose OK. A dialog box lists the objects and table information that you preselected:

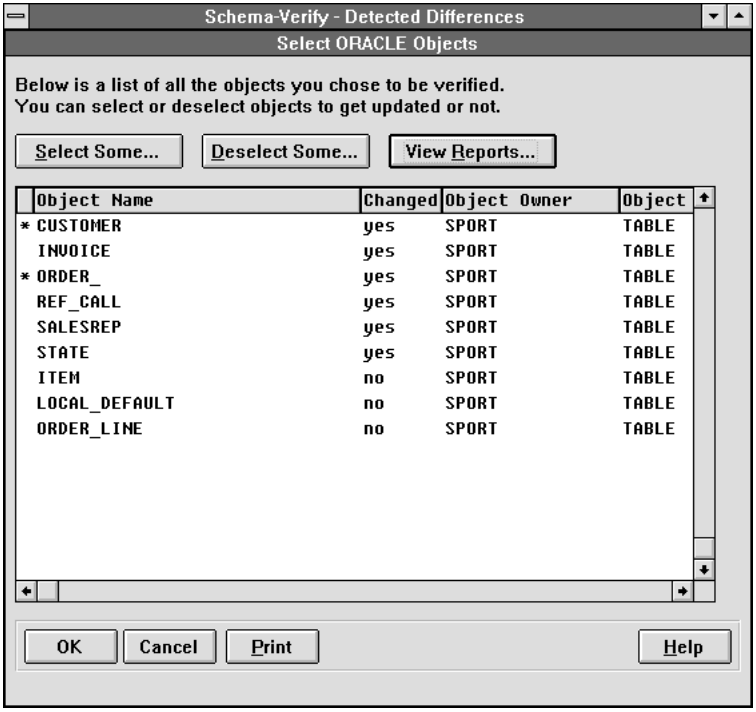


4 ♦ Select the objects you want to update, then choose OK.

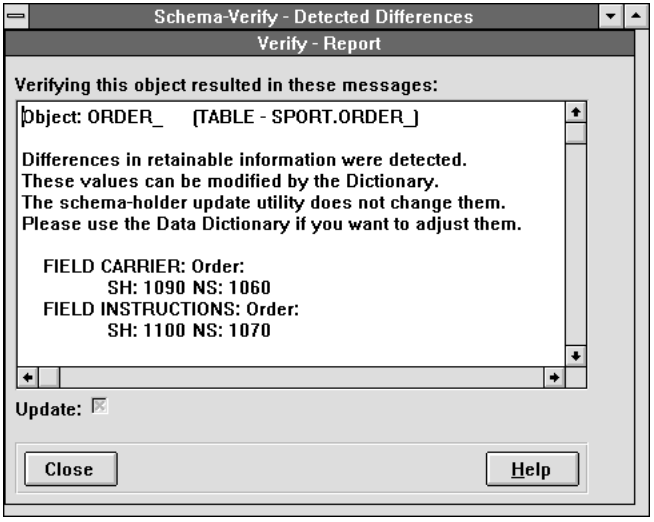
5 ♦ To select tables by matching a pattern, choose the Select Some button. The following dialog box appears:



6 ♦ Type the pattern that you want to match, then choose OK to start the verification. The following dialog box appears listing the objects and the results of the verification:



7 ♦ Choose the View Reports button to view a description of the differences found. SH indicates the value in the schema image; NS indicates the value in the ORACLE database:



8 ♦ Chose Close to return to the Schema Verify dialog box.

9 ♦ The utility automatically selects objects with severe differences for updating. You can select or deselect all other objects as you wish.

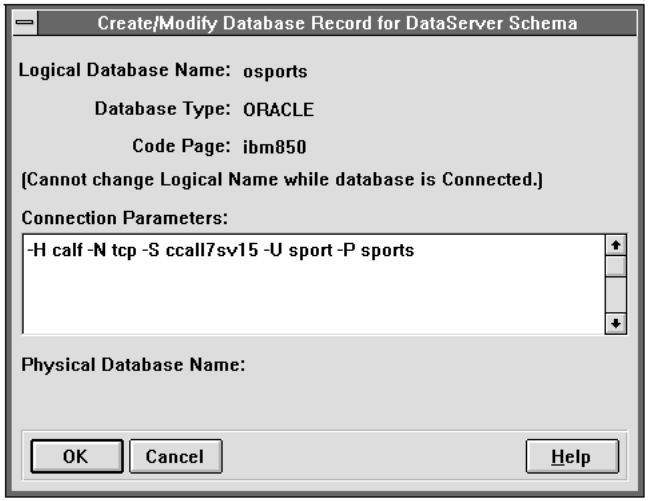
You must resolve retained differences manually. Retained differences appear in reports until you resolve them.

10 ♦ Choose OK to start the update or Cancel to quit the utility without updating the schema image.

5.4 Changing Logical Name and Connection Information

To change connection information for a schema image and the ORACLE database, follow these steps.

1 ♦ Choose DataServer→ ORACLE Utilities→ Edit Connection Information. The following window appears:



2 ♦ Make changes to the Connection Parameters fields. When you are done, choose OK to return to the main window.

The changes do not take effect until you disconnect and reconnect the schema holder. When you reconnect, Progress uses the new connection parameters.

For details on connection parameters, see Chapter 4, "Connecting the DataServer," and the *Progress Startup Command and Parameter Reference*.

This utility also allows you to change the logical name of the ORACLE database. Because you started the tool when you were connected to the database under another name, changing the name causes the tool to close. You can restart the tool to continue working with the utilities.

NOTE: If you rename a schema image, you must recompile the Progress procedures that you compiled under the old schema-image name.

5.5 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 Progress 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 Progress information at both the table and field levels in the schema image. The following sections describe how to enter Progress information at both levels.

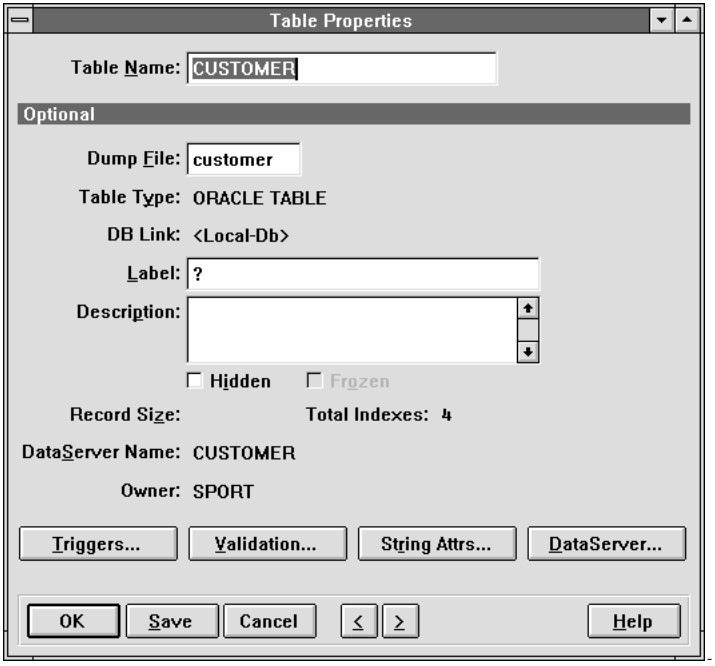
5.5.1 Modifying Table-level Information

Follow these steps to modify information in the schema image at the table level:

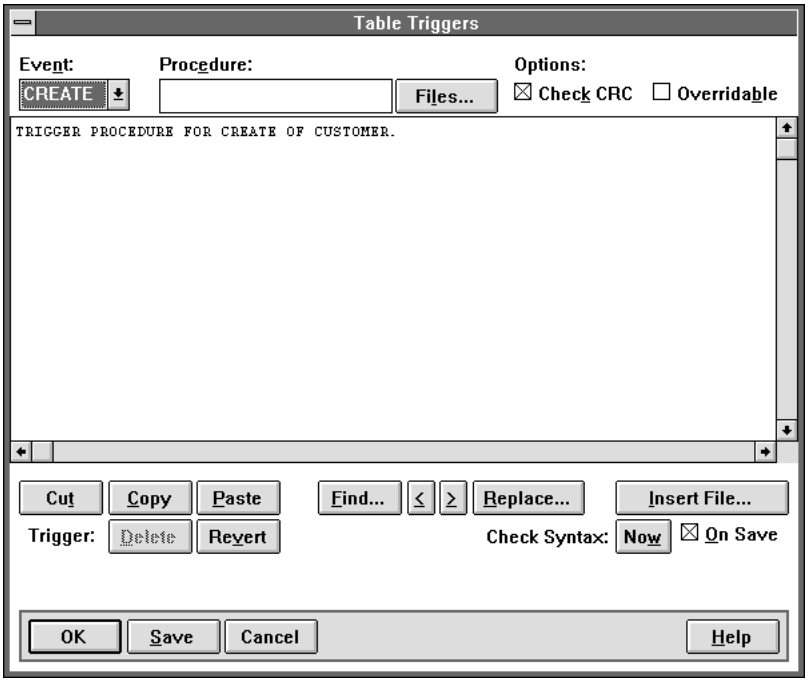
1 ♦ Access the Data Dictionary.

2 ♦ Select a table from the Tables list.

3 ♦ Choose the Table Properties button. The following dialog box appears:



4 ♦ Choose the Triggers button. The Table Triggers dialog box appears:



You can create or include a trigger procedure.

5 ♦ Choose OK to return to the Table Properties dialog box.

6 ♦ Choose OK to return to the Data Dictionary.

5.5.2 Modifying Field-level Information

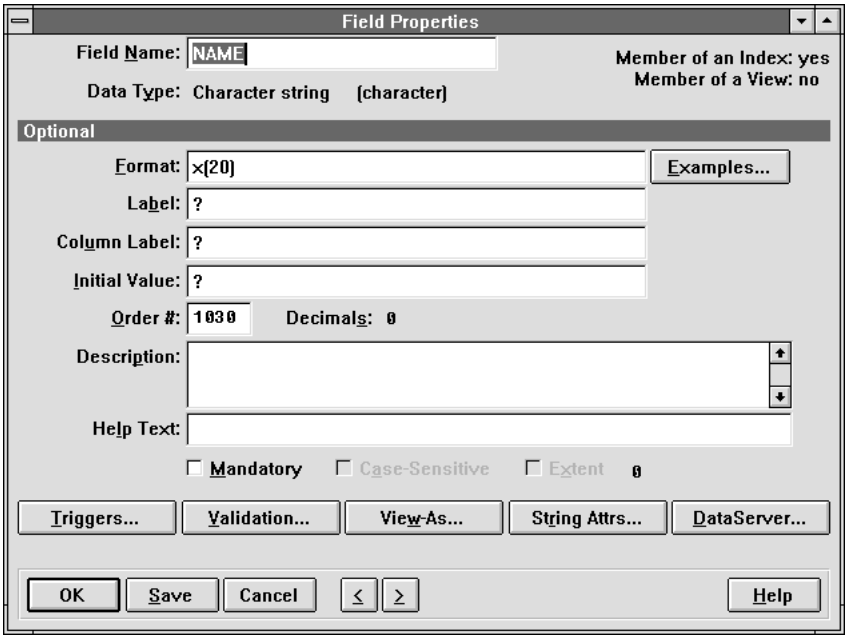
Follow these steps to modify information in the schema image at the field level:

1 ♦ From the Data Dictionary main window, choose the Fields mode button.

2 ♦ Select a table from the Tables list.

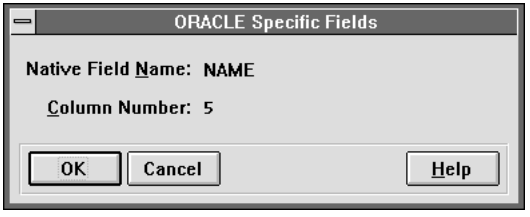
3 ♦ Select a field from the Fields list.

4 ♦ Choose the Field Properties button. The following dialog box appears:



You can enter Progress 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 Progress displays the data. For CHARACTER fields that are not indexed, you can change the case sensitivity. You cannot change a field definition to be an extent through the Data Dictionary.

5 ♦ Choose the DataServer button. The following dialog box provides information about the corresponding column in the ORACLE database:



6 ♦ Choose OK to return to the Field Properties dialog box.

7 ♦ When you are done making changes, choose OK to return to the Data Dictionary.

NOTE: You can override field-level validation expressions in your application by including the appropriate Progress 4GL statement.

5.5.3 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 scale £ 0 or undefined.
- If none of the above is available, the DataServer uses the native ROWID.

The Progress 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 must be defined to meet at least these criteria:

- The index consists of a single field.
- The indexed column is NUMBER data type.

Your application must address these additional criteria if the index definition does not meet them:

- The index must be treated as unique by your application.
- The indexed column must be mandatory in your application.
- The indexed column is NUMBER data type and allows only values with less than ten digits.

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

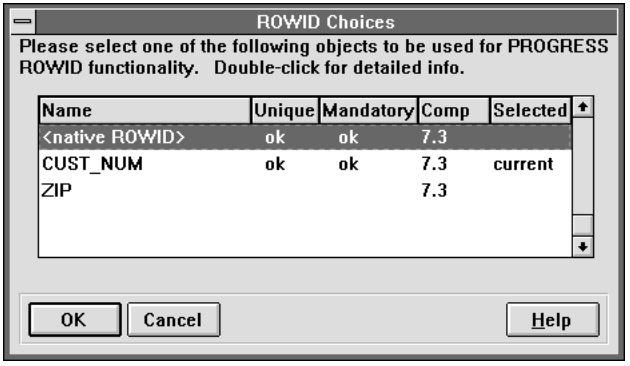
that the indexed column meets the other two criteria as well. If you do not meet all three criteria, you risk corrupting your database.

To select an index to support the Progress ROWID function, follow these steps in the Data Dictionary:

1 ♦ Select the table from the Tables list.

2 ♦ Choose the Table Properties button.

3 ♦ Choose the DataServer button. The ROWID Choices dialog box appears:



4 ♦ Choose native ROWID or an index to support the ROWID function. If the ORACLE table contains a PROGRESS_RECID column, the following message appears:



5 ♦ Choose OK to return to the Table Properties dialog box.

6 ♦ Choose OK to return to the Data Dictionary.

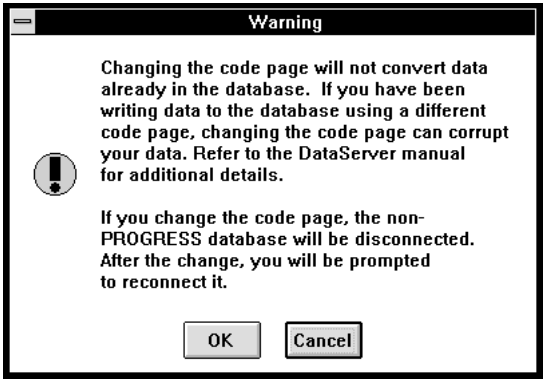
7 ♦ If you referenced the table during the current session, you must disconnect from ORACLE, then reconnect, for the ROWID selection to take effect.

5.6 Changing a Code Page

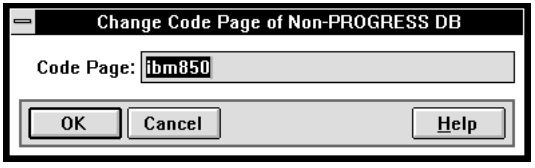
You can change the code page in a schema image at any time. The Create DataServer Schema utility allows you to create a schema image even if you do not have the correct code-page information. You can add or correct the code-page information at a later date. However, if you have been writing 8-bit character 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.

Follow these steps to change the 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 Progress name for the code page that the ORACLE Call Interface (OCI) uses.

See the*Progress Internationalization Guide* for a complete list of code pages that Progress supports. If you are using an unsupported code page, Progress allows you to create your own conversion tables.

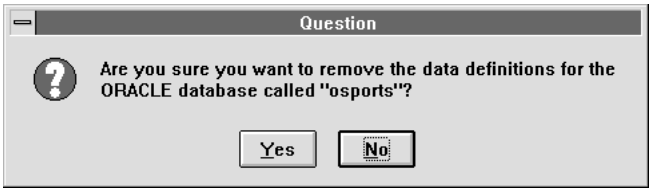
4 ♦ Choose OK to change the code page and return to the Data Administration main window.

If you were connected to the schema holder and ORACLE database when you chose to change the code page, Progress disconnected you to make the change. The Connect Database dialog box appears to allow you to reconnect.

5.7 Deleting a Schema Image

Follow these steps to delete a schema image from a schema holder:

1 ♦ Choose DataServer→ ORACLE Utilities→ Delete DataServer Schema. The following dialog box appears:



2 ♦ Choose Yes to verify your selection. After Progress deletes the schema image, it returns to the main window.

5.8 Migrating a Progress Database to ORACLE with PRO/SQL

The PRO/SQL utility creates a SQL script (.sql file) that contains all the data definition statements needed to re-create a Progress database as an ORACLE database. The script contains statements to define only those features that are supported by ORACLE. For example, if your Progress 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 Progress functionality that would not otherwise be available in an ORACLE database. The script creates a PROGRESS_RECID column to support Progress record identifiers, a shadow column for each case-insensitive indexed column, and a column for each extent of an array.

PRO/SQL is not integrated into the DataServer architecture. Unlike the Progress-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 the *Progress Basic Development Tools* manual or online help for instructions on using PRO/SQL.

5.9 The Progress-to-ORACLE Utility

The DataServer for ORACLE supports the Progress-to-ORACLE utility that allows you to migrate a Progress database to an ORACLE database. While the DataServer typically makes an ORACLE database conform to a Progress database, this utility provides compatibility in the opposite direction. It copies an existing Progress 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 Progress 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 Progress-to-ORACLE utility has the following requirements:

- A local Progress database.
- ORACLE permissions, as Table 5–4 describes.

Table 5–4: Permissions for the Progress-to-ORACLE Utility *(1 of 2)*

Permission	Object
CREATE SESSION	Database
CREATE TABLE	Database
CREATE SEQUENCE	Database

Table 5–4: Permissions for the Progress-to-ORACLE Utility *(2 of 2)*

Permission	Object
SELECT	System objects: sys.argument$ sys.col$ sys.com$ sys.con$ sys.dual sys.icol$ sys.ind$ sys.link$ sys.obj$ sys.procedure$ sys.seq$ sys.syn$ sys.tab$ sys.user$ sys.view$
Enough quota	Default table space

5.9.1 Preparing a Database for the Utility

The Progress-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 Progress and ORACLE systems. Table 5–5 describes the changes to a database that result from making a Progress database compatible with ORACLE by selecting the Create Extended 4GL Objects option when using the Progress-to-ORACLE utility.

Table 5–5: A Progress-to-ORACLE Database Conversion *(1 of 2)*

Progress Database	ORACLE Database
Record ID	An additional NUMBER column named PROGRESS_RECID
Case-insensitive index	An additional column named U##field

Table 5–5: A Progress-to-ORACLE Database Conversion *(2 of 2)*

Progress Database	ORACLE Database
Array	A column for each array element named field##1, field##2, etc.
Sequence	A sequence with the same characteristics (incrementation, beginning value, etc.)

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.

5.9.2 ORACLE Size Limitations

The limitations that ORACLE and Progress place on the number of columns or fields per table are important issues to consider when you plan a migration:

- **ORACLE7** — 254 columns per table
- **ORACLE8** — 1,000 columns per table
- **Progress** — 32,000 fields per table

When you use the Progress-to-ORACLE utility to create an ORACLE database, it adds columns to an ORACLE table to accommodate Progress functionality (see Table 5–5). Because ORACLE7 limits the size of a database table to 254 columns and ORACLE8 to 1,000 columns, you must plan for the additional columns generated by the utility when you create the database schema.

When you design the Progress database tables that you want to make compatible with ORACLE, leave enough columns free for the migration utility to use. Table 5–6 lists how many columns on an ORACLE table each Progress object requires.

Table 5–6: Columns Required by Progress Objects *(1 of 2)*

Database Object	Number of Columns
Record ID	One per table
Case-insensitive indexes	One per indexed case-insensitive field
Array	One per array element

Table 5–6: Columns Required by Progress Objects *(2 of 2)*

Database Object	Number of Columns
Sequence	One per sequence[1]

[1] A sequence does not require an additional column. For each sequence in the Progress database, the utility creates an ORACLE sequence object.

Use the following formula to calculate how many columns an ORACLE7 table has available for data:

```
columns available for fields = 254 - 1 - case-insensitive indexed fields -
number of extent elements
```

For example, a table that contains 253 fields will convert correctly only if none of the fields are extents and you have marked none of the indexed fields as case insensitive. If your table includes 2 field extents, and each extent has 5 elements, the utility needs 10 columns to unroll the field extents. The ORACLE7 table then has only 243 columns available for other fields:

```
243 = 254 - 1 - 0 - 10
```

Column Names

ORACLE allows column names to be only 30 characters long. The Progress-to-ORACLE utility truncates the names of the Progress fields so that they meet this limitation. After running the utility, you can use the Progress Data Dictionary to access the schema holder and modify the field name.

The utility names objects it creates in the ORACLE database by adding symbols to ORACLE table or column names. It can generate a name that is longer than 30 characters, which causes ORACLE to return error 972 when you access that column. For example, to create a unique record identifier in the State table, the utility creates a column named STATE##PROGRESS_RECID.

Column Width

The Progress-to-ORACLE utility uses a field's format information when it defines the field as an ORACLE column. Since Progress allows a field to hold more data than the field's format will display, the utility might instruct ORACLE to create a column that is wider than the format indicates. For those fields with a display format of x(8), the utility automatically generates a

VARCHAR(30) column. Before running the utility, adjust the display format of fields to accommodate all the data the fields contain.

If a column generated by the utility is not wide enough to hold the data, Progress does not load the remainder of the data for that table into the ORACLE database.

Follow these steps to change the width of the column:

1 ♦ Use an editor to open the file with the .sql extension. The migration utility creates the .sql file, which contains information for the entire database.

2 ♦ Find the section of the file that describes the column.

3 ♦ Assign the column a larger value to accommodate the data.

4 ♦ Extract the section of the file that describes the column and place it in another file named *filename*.sql. Be sure to include any index information for the column.

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.

5.9.3 Naming Conventions

The Progress-to-ORACLE utility generates names for the columns it adds to an ORACLE table. These names follow the conventions described in Table 5–7.

Table 5–7: Progress-to-ORACLE Naming Conventions *(1 of 2)*

Progress Field	ORACLE Column Name
Record ID	PROGRESS_RECID
Case-insensitive index (for example, cust-num)	U##*column-name* (U##cust_num)

Table 5–7: Progress-to-ORACLE Naming Conventions *(2 of 2)*

Progress Field	ORACLE Column Name
Array with three elements (for example, MONTH)	*field*##1, *field*##2, *field*##3 (MONTH##1, MONTH##2, MONTH##3)
Sequence	*sequence-name* (NEXT_CUST_NUM)

For example, if your source Progress table includes a field extent named MONTH with 12 elements, the Progress-to-ORACLE utility creates 12 columns of the same data type named MONTH##1, MONTH##2, MONTH##3, etc. A Progress 4GL reference to MONTH[3] translates into a reference to the ORACLE column MONTH##3.

In addition, the Progress-to-ORACLE utility modifies the names of Progress 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 Progress 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 Progress field name Order-line changes to Order_line to account for the fact that the hyphen (–) is an unacceptable character in ORACLE object names.

5.9.4 ORACLE Tablespaces for Tables and Indexes

The Progress-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 Progress-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, the utility gives you the option of specifying a different online tablespace or using the SYSTEM tablespace. If no tablespace is defined in ORACLE, the utility uses the SYSTEM tablespace. See your ORACLE documentation for more information on tablespaces.

5.9.5 Unknown Value

Table 5–8 shows how the Progress-to-ORACLE migration utility maps Progress unknown values and zero-length character strings to ORACLE values.

Table 5–8: Unknown Value Mapping

Progress Value	ORACLE Value
? (Unknown value)	NULL
"" (Zero-length character string)	" " (One space)
" " (One space)	" " (One space)

Strings that contain one or more blank spaces in a unique index might cause an error when you attempt to load data. The blank spaces are stripped out, therefore the strings are no longer unique.

5.9.6 Running the Progress-to-ORACLE Utility

You can run the Progress-to-ORACLE utility interactively or in batch mode. Follow these steps to migrate a Progress database to ORACLE:

1 ♦ Create a target ORACLE7 or ORACLE8 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 Progress-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 ♦ 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 Progress client and connect to the Progress 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 Progress client. The values that you specify for these parameters much match the code page that the ORACLE database uses.

6 ♦ On Windows, from the Data Admin tool choose DataServer→ ORACLE Utilities→ Schema Migration Tools→ PROGRESS DB to ORACLE.

On UNIX, access the utility from the DataServer menu in the Data Dictionary. Or, you can start Progress and run the utility from the command line:

```
pro source_db -p prodict/ora/protoora.p
```

7 ♦ The following screen appears and prompts you for the information described in Table 5–9:

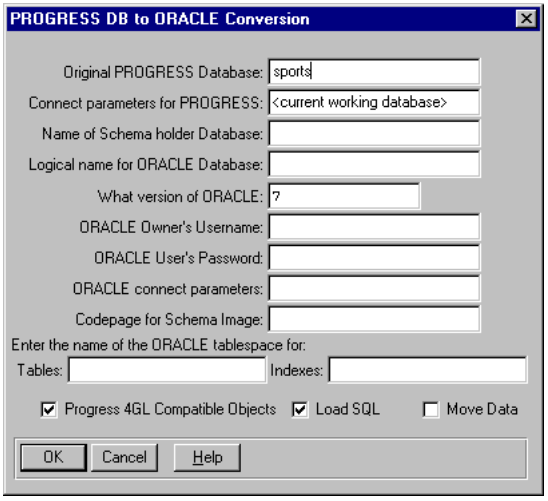


Table 5–9: Progress-to-ORACLE Utility *(1 of 3)*

Interface Element	Description
Original Progress Database	Enter the source database name.
Connect parameters for Progress	Enter parameters for the connection to the source Progress database, which is the current working database. If you started the utility at the command line, do not enter any parameters that you specified when you started the utility
Name of Schema holder Database	Enter the name for the schema holder. The utility creates the schema holder if it does not exist.

Table 5–9: Progress-to-ORACLE Utility *(2 of 3)*

Interface Element	Description
Logical name for ORACLE Database	Enter the ORACLE database logical name. The logical database name is the name of the schema image and the name you will use to refer to the ORACLE database in applications. The database's logical name must be different than the name you enter for the schema holder and different than the name of any other schema image existing in that schema holder.
What version of ORACLE	Enter the version of ORACLE you are using. Possible values are 7 or 8. The default is 7.
ORACLE Owner's Username	Enter the ORACLE database owner's name.
ORACLE User's password	Enter the owner's password.
ORACLE connect parameters	Enter additional connection parameters for the schema image. The utility provides the required -U and -P parameters, but you might want to specify others, such as any -Dsrv options.
Code page for Schema Image	Enter the Progress name for the code page that the ORACLE Call Interface (OCI) uses. By default, the code page for a schema image is ibm850. You can leave this field blank and use the Change Code Page utility to add the code page information for the schema image later.
Tables	Enter the name of the ORACLE tablespace where you want to store schema information. The default is the ORACLE user's default tablespace.
Indexes	Enter the name of the ORACLE tablespace where you want to store index information. The default is the ORACLE user's default tablespace.

Table 5–9: Progress-to-ORACLE Utility *(3 of 3)*

Interface Element	Description
Progress 4GL Compatible Objects	Check this toggle box to create an ORACLE database that supports arrays, case-insensitive indexes, backward and forward scrolling, and the Progress record identifier. These objects result in additional columns added to ORACLE tables. This is the default option. If you do not want Progress 4GL compatibility, deselect the toggle box.
Load SQL	Check this toggle box to load the .sql file that contains the data definitions for your Progress database into the ORACLE database. This is the default option.
Move Data	Check this toggle box to dump and load data from your Progress 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.

If you want a complete migration of your Progress database to ORACLE, you must enter information in all fields and check all toggle boxes.

The utility creates a schema holder, an ORACLE database that contains the objects from your Progress 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.

Follow these steps to run the Progress-to-ORACLE utility in batch mode on a UNIX client machine:

1 ♦ Create a target ORACLE database. You must use an empty ORACLE database when you run the Progress-to-ORACLE utility.

2 ♦ Start an instance of your target ORACLE database.

3 ♦ 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 listed in Table 5–10.

Table 5–10: Progress-to-ORACLE Utility Batch Parameters *(1 of 2)*

Environment Variable	Description
PRODBNAME	Specify the source Progress database name.
PROCONPARMS	Specify parameters for the connection to the source Progress database.
SHDBNAME	Specify the new schema-holder name.
ORADBNAME	Specify the ORACLE database logical name. The database's logical name can be the same as its physical name, but it must be different from the name you enter for the schema holder.
ORAVERSION	Specify the version of ORACLE you are using. Possible values are 7 or 8. The default is 7.
ORAUSERNAME	Specify the ORACLE user's name.
ORAPASSWORD	Specify the user's password.
ORACONPARMS	Specify additional connection parameters for the schema holder.
ORACODEPAGE	Specify the Progress name for the code page that the ORACLE Call Interface (OCI) uses. By default, the code page for a schema image is ibm850. You can leave this field blank and use the Change Code Page utility to add the code page information for the schema image later.
TABLEAREA	Enter the name of the ORACLE tablespace where you want to store schema information. The default is the SYSTEM tablespace.
INDEXAREA	Enter the name of the ORACLE tablespace where you want to store index information. The default is the SYSTEM tablespace.

Table 5–10: Progress-to-ORACLE Utility Batch Parameters *(2 of 2)*

Environment Variable	Description
COMPATIBLE	Specify YES to create an ORACLE database that supports arrays, case-insensitive indexes, backward and forward scrolling, and the Progress record identifier. These objects result in additional columns added to ORACLE tables. Enter NO if you do not want the Extended 4GL capability. The default is YES.
LOADSQL	Specify YES to load the .sql file that contains the data definitions for your Progress database into the ORACLE database. Specify NO if you do not want the utility to load the .sql file, for example if you want to edit the file before loading it. The default is YES.
MOVEDATA	Specify YES to dump and load data. Otherwise, if you do not want to populate the ORACLE database, specify NO. For example, you might specify NO if your database is large, and you want to dump and load data at a more convenient time. The default is NO.

6 ♦ Enter these commands to set and export environment variables at the system prompt before running `protoora.p`:

```
PRODBNAME=db-name; export PRODBNAME
PROCONPARMS="-1 -i"
SHDBNAME=schema-holder-name; export SHDBNAME
.
.
.
pro -b -p prodict/ora/protoora.r
```

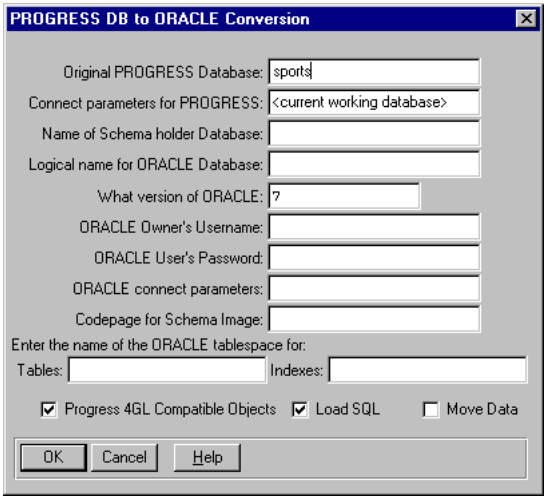
5.10 Progress-to-ORACLE Incremental Schema Migration Utility

The Incremental Schema Migration utility allows you to migrate schema changes from Progress to an ORACLE database. For example, in the process of developing an application in Progress that you will migrate to ORACLE, you might want to make and test schema changes in the Progress database that you want reflected in the ORACLE database. The utility reads a Progress `delta.df` file that has been created using the standard incremental dump procedure. and creates a `delta.sql` file that contains the SQL DDL for making the changes and a `delta.df` file. You can then load the `delta.df` file into the schema holder. You can then apply the `delta.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 what the ORACLE definitions are.

Follow these steps to run the utility:

1 ♦ From the Data Admin main menu, choose DataServers→ ORACLE Utilities→ Schema Migration Tools→ Generate Delta.sql Progress to ORACLE. The following dialog box appears:



2 ♦ Provide the information as described in Table 5–11.

Table 5–11: Generate Delta.sql Progress-to-ORACLE Utility *(1 of 2)*

Interface Element	Description
Delta DF File	The name of the delta.df file that was created when you ran the incremental dump routine against two Progress databases. You can browse for the file name by choosing the Files button.
Schema Holder Database	The name of the schema holder. The schema information must match the schema of the ORACLE database to be updated. Progress Software recommends that you are connected to this database before running the utility. If you are not connected to the database, you can connect to it at this point using the Connect Parameters.
Connect parameters for Schema	Specify these parameters to connect to the ORACLE database to be updated if you are not already connected to the schema holder.
Logical name for ORACLE database	Specify the ORACLE database logical name, that is, the name by which you refer to the ORACLE database in your application.
ORACLE tablespace for Tables	Enter the names of any tablespaces to be used here.
ORACLE tablespace for Indexes	Enter the names of any tablespaces to be used for indexes here.

Table 5–11: Generate Delta.sql Progress-to-ORACLE Utility *(2 of 2)*

Interface Element	Description
Progress 4GL Compatible Objects	Check this toggle box to create a target database that supports arrays, case-insensitive indexes, backward and forward scrolling, and the Progress record identifier. These objects result in additional columns added to the ORACLE database's tables.
Create schema holder delta.df	Check this toggle box if you want the utility to generate a .df file that includes the incremental schema information. You can then load this .df file into the schema holder. By default, this toggle box is checked.

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 that are compatible with Progress. It creates the same objects as the Progress-to-ORACLE Migration utility. For example, Progress indexes are case-insensitive. To create this equivalent functionality in the ORACLE database, for an index defined in the Progress database on a CHARACTER field, the utility generates SQL to create two columns in ORACLE: the second column is the uppercase shadow of the first. Table 5–12 describes the ORACLE equivalents of Progress object types.

Table 5–12: ORACLE Equivalents of Progress Objects

Progress Object	ORACLE Equivalents
Case-insensitive Index	Two indexed columns: the first column is the uppercase equivalent of the second column and is named U##*first-column-name*.
Array	One column for each extent of the Progress array. The columns are named *field-name*##*extent-number*. For example, a Progress field called monthly-amount with an extent of 12 will have 12 columns in ORACLE with names such as MONTHLY_AMOUNT##1 through MONTHLY_AMOUNT##12.
Table	A PROGRESS_RECID column. This indexed column provides a unique key on the ORACLE table. A sequence named *table-name*_SEQ. This sequence populates the PROGRESS_RECID column for each row in the ORACLE table.
Obsolete Field	A view with a name ending in _V##. When you delete a field from the Progress database, the utility identifies this obsolete field and generates SQL to create an ORACLE view of the table that excludes the obsolete column because ORACLE does not allow the explicit deletion of a column from a table. Whenever Progress programs reference the table, the DataServer references the ORACLE view that contains the correct columns.

Table 5–13 shows how the fields of a Progress table convert to ORACLE equivalents.

Table 5–13: Sample Object Equivalents

Progress State Table	ORACLE STATE Table
Case-insensitive index: State	U##STATE STATE
Character field: State-Name	STATE_NAME
Array with 3 Extents: State-Fact	STATE_FACT##1 STATE_FACT##2 STATE_FACT##3
Default record identifier object	STATE##PROGRESS_RECID STATE_SEQ

The utility ensures that the migrated objects have names that are unique to the ORACLE database. If you have given the object a name that is not unique, it drops characters from the end of the name and appends numbers until it creates a unique name. Since ORACLE requires that index names be unique to the database, the utility appends the table name to the indexed column name to create a unique name.

5.10.1 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.

5.11 Adjust ORACLE Schema Image to Progress Database

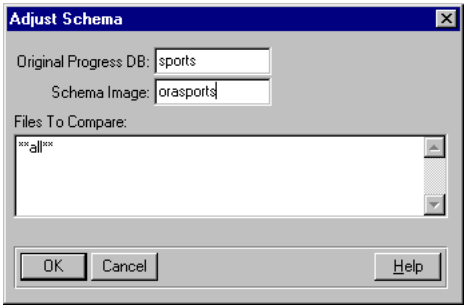
The Adjust Schema utility allows you to migrate schema changes from an ORACLE database and compare it to the same schema in a Progress database. If you make changes in both your Progress database and the same changes in ORACLE using ORACLE tools, pull those changes into the schema holder. The Adjust Schema utility will compare the Progress database to the

schema image and update the Progress attributes necessary for your 4GL code to run against both data sources.

Follow these steps to run the Adjust Schema Utility:

1 ♦ 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:



2 ♦ Provide the information as described in Table 5–14.

Table 5–14: Adjust Schema Utility Progress-to-ORACLE Utility

Interface Element	Description
Original Progress DB	Enter the name of the source Progress database
Schema Image	Enter the ORACLE database logical name. The logical database name is the name of the schema image.
Files to Compare	Either leave the default **all** for all objects to be compared or enter the names of tables, sequences, or Progress Views to be compared. If listing individual objects, the list must be a comma-separated list within a group of objects and semi-colons must separate the groups.

3 ♦ Connect to the source Progress database.

4 ♦ Connect to the Schema Image Schema Holder.

5 ♦ Select Adjust Schema option from the Oracle Utilities Schema Migration menu.

6 ♦ Verify that the proper names are displayed in the Original Progress DB and Schema Image fields.

7 ♦ Either leave the **all** to compare all objects or type the names of objects to be compared as follows:

```
<table>,<table>,....;<sequence>,<sequence>,....;<view>,<view>,....
```

8 ♦ Select OK to start the utility.

A

Upgrading DataServer Applications

This appendix addresses users who have been using Progress Version 7 or 8 and want to upgrade to take advantage of Version 9 features. The topics discussed here include:

- Upgrading a Progress Version 7 or 8 schema holder
- Upgrading a DataServer application from ORACLE7 to ORACLE8

A.1 Upgrading Schema Holders from Progress Version 7 or 8

If you have Version 7 or 8 schema holders and want to take advantage of Progress Version 9 features, make the following preparations before using the Version 9 DataServer:

1 ♦ Start Progress Version 7 or Version 8 with the schema holder connected.

2 ♦ Dump the data definitions (.df file) from the schema holder and move the .df file to the new client machine, if necessary. Dumping and loading a .df file is the only way to preserve any information you might have added to the schema, such as display formats, help strings, and validation expressions.

3 ♦ Start Progress Version 9.

4 ♦ Create and connect to an empty Version 9 Progress database.

5 ♦ From the Data Admin main menu, choose Admin→ Load Data and Definitions→ Load 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 ORACLE Table Definitions DataServer utility to verify the data definitions in the Version 9 schema holder.

8 ♦ In Version 8, the DataServer did not store information in the schema holder on maximum sizes of CHARACTER and RAW columns. 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.

A.1.1 Using the Manual Upgrade Technique

Follow these steps to modify ORACLE tables manually to take advantage of the FIND PREV/LAST statements, cursor repositioning, and arrays. There are two parts to this process: in part A you make changes to your ORACLE database; in Part B you make changes to the schema holder. Part B is required only if you created your schema holder with a DataServer prior to Release 6.3D.

Part A

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:

```
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:

```
ALTER TABLE table-name ADD (progress_recid number null);
```

4 ♦ Update the table and set the progress_recid using *table-name*_SEQ.nextval:

```
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:

```
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:

```
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:

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

This statement also verifies that the system table sys.dual exists. Your ORACLE database must have the sys.dual table to support the extended 4GL features. If the sys.dual table does not exist in your ORACLE database, see the "Creating the ORACLE sys.dual Table" section for instructions.

8 ♦ Connect to ORACLE and use the Progress Data Dictionary's ORACLE utilities to update the schema holder.

NOTE: If you created your schema holder with a DataServer prior to Release 6.3D, follow the steps in Part B before you connect to ORACLE and update the schema holder.

Here's an example of these SQL*Plus commands for the Customer table in the Progress demo database. The first command creates a sequence for the Customer table. The sequence starts with 1 and increments by 1:

```
CREATE SEQUENCE customer_SEQ START WITH 1 INCREMENT BY 1;
```

This line adds a column to the Customer table. It names the column progress_recid and declares the data type as NUMBER, where the number can be null:

```
ALTER TABLE customer ADD (progress_recid number null);
```

This command updates the Customer table and sets progress_recid using the sequence value:

```
UPDATE customer
   SET progress_recid = customer_SEQ.nextval;
```

This command creates a unique index for the Customer table that consists of progress_recid:

```
CREATE UNIQUE INDEX customer##progress_recid ON
   customer (progress_recid);
```

This line drops the name index for the Customer table:

```
DROP INDEX customer##name;
```

This command creates a new index for customer that uses the same components as the old index, with the addition of progress_recid:

```
CREATE INDEX customer##name ON customer (name,
   progress_recid);
```

These commands drop the zip index for the Customer table and recreate it with the addition of progress_recid:

```
DROP INDEX customer##zip;
```

```
CREATE INDEX customer##zip ON customer (zip,
   progress_recid);
```

Part B

These steps are necessary only if you created your schema holder with a DataServer prior to Release 6.3D.

1 ♦ Start the Progress Version 6 client and connect to the schema holder only. Do **not** connect to the ORACLE database.

2 ♦ Access the Data Dictionary and select Modify→ Schema→ Modify Existing File.

3 ♦ Type the filename **ORACLE6_COLUMNS** and press **RETURN**.

4 ♦ Tab to the Hidden attribute and change it to NO. Press **GO**.

5 ♦ Choose GoIndex, then add an index to ORACLE6_COLUMNS. Give the index any valid name and select any field. You are creating this index so that you can delete the file ORACLE6_COLUMNS. (The Data Dictionary does not let you delete a file that has no index.)

6 ♦ Delete the table ORACLE6_COLUMNS.

7 ♦ Apply the changes and exit the Data Dictionary.

8 ♦ Enter and run the following procedure:

```
FIND _Db WHERE _Db-name = "logical database name".
RUN prodict/ora/ora6_crc.p (RECID (_Db)).
```

9 ♦ Connect to ORACLE and use the Progress Data Dictionary's ORACLE utilities to update the schema holder.

Now you can run Progress procedures against the ORACLE database and take advantage of the DataServer's compatibility features, with the exception of case-insensitive indexed fields.

A.2 Upgrading from ORACLE7 to ORACLE8

Note that there is no path for upgrading an ORACLE7 database to ORACLE8 through the DataServer. If you want to upgrade a DataServer application from ORACLE7 to ORACLE8, follow these steps:

1 ♦ Upgrade your ORACLE7 database to ORACLE8. See your ORACLE documentation for instructions.

2 ♦ Using the Progress Version 9 Data Admin Dump utility, dump a data definition (.df) file from the schema holder you used to access the ORACLE7 database.

3 ♦ In the environment where you will run Progress, set the ORAVERSION environment variable to 8.

4 ♦ Create a new empty Version 9 Progress database.

5 ♦ Using the Progress Data Admin Load utility, load the .df file into the empty Version 9 Progress database which results in a new schema holder for the ORACLE8 database.

6 ♦ Recompile your Progress r-code against the new schema holder.

B

Stored Procedure Reference

This appendix contains reference entries for extensions to the Progress 4GL that support running ORACLE stored procedures from within Progress procedures. The appendix describes the following Progress 4GL statements and functions:

- CLOSE STORED-PROCEDURE Statement
- PROC-HANDLE Function
- PROC-STATUS Function
- RUN STORED-PROCEDURE Statement

CLOSE STORED-PROCEDURE Statement

Retrieves the output parameter values from an ORACLE stored procedure and indicates that the procedure is finished.

SYNTAX

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

`procedure`

The name of the stored procedure that you want to close.

`integer-field = PROC-STATUS`

The PROC-STATUS function returns the value of the return value from the ORACLE stored procedure.

`WHERE PROC-HANDLE = integer-field`

An integer whose value uniquely identifies the stored procedure that produces the results returning from the ORACLE database.

EXAMPLE

The PROC-STATUS clause of the CLOSE STORED-PROCEDURE statement allows the DataServer for ORACLE to retrieve the text of an ORACLE error message that was passed to raise_application_error. Use the ERROR-STATUS:GET-MESSAGE handle to retrieve the message as in the following example:

```
DEFINE VARIABLE st AS INTEGER INITIAL 0.
DEFINE VARIABLE h  AS INTEGER.
RUN STORED-PROC p1 h = PROC-HANDLE NO-ERROR.
CLOSE STORED-PROC p1 st = PROC-STATUS WHERE PROC-HANDLE = h.
DISPLAY st.
IF ERROR-STATUS:ERROR
     THEN
   MESSAGE ERROR-STATUS:GET-MESSAGE(1) ERROR-STATUS:GET-NUMBER(1)
          VIEW-AS ALERT-BOX.
```

NOTES

- If you specified a PROC-HANDLE when you ran a stored procedure, you must specify the PROC-HANDLE when you close the stored procedure.
- If you do not specify a PROC-HANDLE, the CLOSE STORED-PROC statement will close the procedure if there is only one stored procedure running. If there is more than one stored procedure running, an error will be returned.
- You can close all stored procedures at once with the following statement:

```
RUN STORED-PROC closeallprocs.
```

- You cannot close a send-sql-statement procedure until you have retrieved all row results.

EXAMPLE

The PROC-STATUS clause of the CLOSE STORED-PROCEDURE statement allows the DataServer to retrieve the text of an ORACLE error message that was passed to raise_application_error. Use the ERROR-STATUS:GET-MESSAGE handle to retrieve the message as in the following example:

```
DEFINE VARIABLE st AS INTEGER INITIAL 0.
DEFINE VARIABLE h  AS INTEGER.
RUN STORED-PROC p1 h = PROC-HANDLE NO-ERROR.
CLOSE STORED-PROC p1 st = PROC-STATUS WHERE PROC-HANDLE = h.
DISPLAY st.
IF ERROR-STATUS:ERROR
     THEN
   MESSAGE ERROR-STATUS:GET-MESSAGE(1) ERROR-STATUS:GET-NUMBER(1)
          VIEW-AS ALERT-BOX.
```

SEE ALSO

PROC-HANDLE Function, PROC-STATUS Function, RUN STORED-PROCEDURE Statement

PROC-HANDLE Function

Returns the appropriate return value data type (usually integer) that acts as a unique identifier for an ORACLE stored procedure:

SYNTAX

```
PROC-HANDLE
```

This procedure runs the ORACLE stored procedure pcust and writes the procedure handle to the variable handle. It writes the results of the stored procedure identified by this procedure handle into the Progress-supplied buffer, proc-text-buffer, and displays it:

EXAMPLE

```
DEFINE VAR h1 AS INTEGER.

RUN STORED-PROC pcust h1 = PROC-HANDLE
(10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = h1:
  DISPLAY proc-text.
END.
CLOSE STORED-PROC pcust WHERE PROC-HANDLE = h1.
```

NOTES

- Progress Software 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 Progress application. The DataServer passes SQL statements to the ORACLE RDBMS and uses the default system handle in the process.

SEE ALSO

CLOSE STORED-PROCEDURE Statement, PROC-STATUS Function, RUN STORED-PROCEDURE Statement

PROC-STATUS Function

Returns the return status from an ORACLE stored procedure. The return status is an integer value that indicates whether a stored procedure failed and why it failed. See the *ORACLE PL/SQL User's Guide and Reference* for descriptions of the possible values for the return status:

SYNTAX

```
PROC-STATUS
```

EXAMPLE

This procedure runs the ORACLE stored procedure pcust and writes the results of the stored procedure into the proc-text-buffer. The CLOSE STORED-PROC statement then retrieves the output parameters. The return status is written to the variable stat and is displayed:

```
DEFINE VAR stat AS INTEGER.

RUN STORED-PROC pcust (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
END.
CLOSE STORED-PROC pcust stat = PROC-STATUS.
  DISPLAY stat.
```

SEE ALSO

CLOSE STORED-PROCEDURE Statement, PROC-HANDLE Function, RUN STORED-PROCEDURE Statement

RUN STORED-PROCEDURE Statement

Runs an ORACLE stored procedure:

SYNTAX

```
RUN STORED-PROCEDURE procedure [ integer = PROC-HANDLE ]
[ ( [ output ] parameter , [ output ] parameter ) ]
[ NO-ERROR ]
```

`procedure`

The name of the stored procedure that you want to run or the built-in procedure name, send-sql-statement, which passes PL/SQL statements to ORACLE.

`integer = PROC-HANDLE`

An integer whose value uniquely identifies the stored procedure that produces the results returning from the ORACLE database.

`parameter`

A run-time parameter to be passed to the stored procedure. A *parameter* has the following syntax:

SYNTAX

```
{ [ INPUT | OUTPUT | INPUT-OUTPUT ]
  [ PARAM parameter-name = ] expression }
```

INPUT is the default. In ORACLE, OUTPUT and INPUT-OUTPUT work the same way. If you do not use the parameter name, you must supply all of the parameters in correct order. If you do use the parameter name, you must precede your assignment statement with the keyword PARAM. You must also name parameters to pass values that are different from the default values. If you do not supply a parameter, and no default is specified in the stored procedure, you receive a run-time error.

You can designate a parameter as an extent in the Progress Data Dictionary. You can also use a named ORACLE cursor as an OUTPUT parameter. If a stored procedure has multiple cursors, you must specify a cursor by name when fetching results.

NO-ERROR

Specifies that any ERROR conditions that the RUN STORED-PROC statement produces are suppressed. Before you close a stored procedure, check the ERROR-STATUS system handle for information about any errors that occurred. You receive an error when you attempt to close a stored procedure that did not start.

EXAMPLES

This procedure runs the ORACLE stored procedure pcust and writes the results of the stored procedure into the Progress-supplied buffer, proc-text-buffer. The same code works for accessing a stored procedure from an ODBC-compliant data source:

```
DEFINE VAR handle AS INTEGER.

RUN STORED-PROCEDURE pcust handle = PROC-HANDLE
(10, OUTPUT 0, OUTPUT 0) NO-ERROR.
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle:
  DISPLAY proc-text-buffer.
END.
IF ERROR-STATUS:ERROR
THEN DO:
  MESSAGE "Stored Procedure failed to run".
END.
ELSE CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = handle.
```

This procedure uses the send-sql-statement option of the RUN STORED-PROCEDURE statement to send SQL to ORACLE. It writes the results of the stored procedure into the Progress-supplied buffer, proc-text-buffer. The same code works for sending SQL to an ODBC-compliant data source:

```
DEFINE VAR handle1 AS INTEGER.

RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE
     ("SELECT name, cust_num FROM customer").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle1:
   display proc-text.
   END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle1.
END.
```

This code example shows how to trap errors from the non-Progress DBMS within a procedure:

```
DEFINE VAR h1 AS INTEGER.
DEFINE VAR j AS INTEGER.
RUN STORED-PROC send-sql-statement h1 = PROC-HANDLE NO-ERROR
  ("select count (*) from xxx.customer where name between 'A' and 'Z' ").

IF ERROR STATUS:ERROR THEN DO:
  DO j = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE "error" ERROR-STATUS:GET-NUMBER(j)
        ERROR-STATUS:GET-MESSAGE(j).
  END.
END.

CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = h1.
```

NOTE

- The RUN STORED-PROCEDURE statement starts a transaction with the same scope as transactions started with the UPDATE statement.

SEE ALSO

CLOSE STORED-PROCEDURE Statement, PROC-HANDLE Function, PROC-STATUS Function

C

Environment Variables

This appendix lists the ORACLE, Progress, and system environment variables that affect building and running the DataServer for ORACLE. See the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX* for more information on Progress environment variables.

Table C–1 lists the environment variables and describes how to set them.

Table C–1: Environment Variables *(1 of 2)*

Variable	How to Set It
DLC	The pathname of the directory where you installed Progress.
DLCLOAD	Before running PROBUILD, set DLCLOAD to `$DLC/probuild`.
DSLOGDIR	The pathname of the log file that Progress uses to keep track of DataServer processes and error messages.
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	The identifier of the shared ORACLE libraries. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. You must set this environment variable to include `$ORACLE_HOME/lib` when you run executables (whether supplied or built with PROBUILD) that you built with PROBUILD to access ORACLE 7.3 or ORACLE8. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information.
ORACLE_HOME	The pathname of the directory where you installed ORACLE.
ORACLE_SID	The identifier for the ORACLE instance you are accessing.
ORASRV	The pathname to the executable of the ORACLE DataServer. Set it to the DataServer executable you want to run: `dlc\bin\_orasrv` (on NT), `$DLC/bin/_orasrv` or one you created with PROBUILD.
ORAVERSION	The version of ORACLE you are accessing (7 or 8, 7 is the default). Set this environment variable when you are upgrading a DataServer application to ORACLE8.
PROBRKR	The pathname to the executable of the broker.

Table C–1: Environment Variables *(2 of 2)*

Variable	How to Set It
PROEXE	The pathname of the Progress client executable. After setting this variable, you can run a customized executable (by default, `_progres`) by entering **pro** at the system prompt.
TNS_ADMIN	The pathname of the directory where `listener.ora` and `tnsnames.ora` SQL*Net configuration files are found for the server. Or for the client, where the `tsnames.ora` file is found.

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.

D

Sample Queries

This appendix contains sample queries and the information that the DataServer provides when you specify the DEBUG SQL query-tuning option. In each case, notes explain the DataServer and cursor behavior. The numbers in angle brackets (<*n*>) indicate cursors.

Query 1

```
FIND customer 2.
DISPLAY name cust-num postal-code.

FIND NEXT customer.
DISPLAY name cust-num postal-code.
```

```
OCI call oopen  <1> cc = 0
 OCI call osql3  <1>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <2> cc = 1
 OCI call osql3  <2>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER T0 WHERE (CUST_NUM = :1)  ORDER BY CUST_NUM
 OCI call oexec  <2>
 OCI call oexec  <1>
 OCI call osql3  <2>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER T0 WHERE ((CUST_NUM > :p1)) ORDER BY CUST_NUM
 OCI call oexec  <2>
 OCI call oexec  <1>

      Cursor <1> Rows processed 1
 OCI call oclose <1> cc = 2

      Cursor <2> Rows processed 81
 OCI call oclose <2> cc = 1
```

<1> The DataServer uses the cursor to compare schema information and fetch column values.

<2> 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.

Query 2

```
FIND customer 2.
DISPLAY name cust-num postal-code.
FIND NEXT customer USE-INDEX Country-Post.

DISPLAY name cust-num postal-code WITH FRAME b.
```

```
OCI call oopen  <3> cc = 0
 OCI call osql3  <3>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <4> cc = 1
 OCI call osql3  <4>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER T0 WHERE (CUST_NUM = :1)  ORDER BY CUST_NUM
 OCI call oexec  <4>
 OCI call oexec  <3>
 OCI call oopen  <5> cc = 2
 OCI call osql3  <5>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##COUNTRY_POST) */ PROGRESS_RECID
      FROM SPORTS.CUSTOMER T0 WHERE ((U##COUNTRY = :p1 AND U##POSTAL_C
     ODE = :p2 AND PROGRESS_RECID > :p0) OR (U##COUNTRY = :p1 AND U##P
     OSTAL_CODE > :p2) OR (U##COUNTRY > :p1)) ORDER BY U##COUNTRY, U##
     POSTAL_CODE, PROGRESS_RECID
 OCI call oexec  <5>
 OCI call oexec  <3>

      Cursor <3> Rows processed 1
 OCI call oclose <3> cc = 3

      Cursor <5> Rows processed 62
 OCI call oclose <5> cc = 2

      Cursor <4> Rows processed 1
 OCI call oclose <4> cc = 1
```

<3> The DataServer uses the cursor to compare schema information and fetch column values.

<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.

Query 3

```
FIND customer 2.
DISPLAY name cust-num postal-code.
```

```
OCI call oopen  <6> cc = 0
 OCI call osql3  <6>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <7> cc = 1
 OCI call osql3  <7>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER T0 WHERE (CUST_NUM = :1)  ORDER BY CUST_NUM
 OCI call oexec  <7>
 OCI call oexec  <6>

      Cursor <6> Rows processed 1
 OCI call oclose <6> cc = 2

      Cursor <7> Rows processed 1
 OCI call oclose <7> cc = 1
```

<6> The DataServer uses the cursor to compare schema information and fetch column values.

<7> This cursor selects the progress_recid column for a particular row by CUST_NUM.

Query 4

```
FIND FIRST customer.
DISPLAY name cust-num postal-code.
FIND customer 2.
DISPLAY name cust-num postal-code.
```

```
OCI call oopen  <8> cc = 0
OCI call osql3  <8>
    SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
OCI call oopen  <9> cc = 1
OCI call osql3  <9>
    SELECT /*+ INDEX_ASC(TO CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
    M SPORTS.CUSTOMER TO ORDER BY CUST_NUM
OCI call oexec  <9>
OCI call oexec  <8>
OCI call oopen  <10> cc = 2
OCI call osql3  <10>
    SELECT /*+ INDEX_ASC(TO CUSTOMER##COUNTRY_POST) */ PROGRESS_RECID
     FROM SPORTS.CUSTOMER TO WHERE ((U##COUNTRY = :p1 AND U##POSTAL_C
    ODE = :p2 AND PROGRESS_RECID > :p0) OR (U##COUNTRY = :p1 AND U##P
    OSTAL_CODE > :p2) OR (U##COUNTRY > :p1)) ORDER BY U##COUNTRY, U##
    POSTAL_CODE, PROGRESS_RECID
OCI call oexec  <10>
OCI call oexec  <8>

     Cursor <8> Rows processed 1
OCI call oclose <8> cc = 3

     Cursor <10> Rows processed 31
OCI call oclose <10> cc = 2

     Cursor <9> Rows processed 83
OCI call oclose <9> cc = 1
```

<8> The DataServer uses the cursor to compare schema information and fetch column values.

<10> The WHERE clause generated by the DataServer positions the cursor after the row retrieved by the first use of cursor <8> to retrieve CUSTOMER 2. Unlike the WHERE clause with cursor <5> (Query 2), this non-unique index has two components.

Query 5

```
FOR EACH customer FIELDS (name cust-num postal-code):
    DISPLAY name cust-num postal-code.
END.
```

```
OCI call oopen  <21> cc = 0
 OCI call osql3  <21>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <22> cc = 1
 OCI call oparse <22>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID uni
     que_int_0,CUST_NUM,U##COUNTRY,COUNTRY,U##NAME,NAME,ADDRESS,ADDRES
     S2,CITY,STATE,U##POSTAL_CODE,POSTAL_CODE,CONTACT,PHONE,U##SALES_R
     EP,SALES_REP,CREDIT_LIMIT,BALANCE,TERMS,DISCOUNT,COMMENTS,PROGRES
     S_RECID FROM SPORTS.CUSTOMER T0
 OCI call oexfet <22>

      Cursor <21> Rows processed 0
 OCI call oclose <21> cc = 2

      Cursor <22> Rows processed 83
      Number of array fetches  6
      Number of rows fetched   83
      Number of array rows     15
      Number of array columns  22
      Number of tables         1
      Space for one row        525
      Requested cache size     8192
      Actual cache size used   7875
 OCI call oclose <22> cc = 1
```

<21> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<22> 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 performed an oexfet to fetch an array of rows. The DataServer used the default cache-size of 8192. Since 525 bytes are required for each row, it used only 7875 bytes of cache to fetch up to 15 rows each call. Processing the 83 rows in the CUSTOMER table required a total of 6 array fetches.

Query 6

```
FOR EACH customer NO-LOCK:
    DISPLAY name cust-num postal-code.
END.
```

```
OCI call oopen  <23> cc = 0
 OCI call osql3  <23>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <24> cc = 1
 OCI call oparse <24>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID uni
     que_int_0,CUST_NUM,U##COUNTRY,COUNTRY,U##NAME,NAME,ADDRESS,ADDRES
     S2,CITY,STATE,U##POSTAL_CODE,POSTAL_CODE,CONTACT,PHONE,U##SALES_R
     EP,SALES_REP,CREDIT_LIMIT,BALANCE,TERMS,DISCOUNT,COMMENTS,PROGRES
     S_RECID FROM SPORTS.CUSTOMER T0
 OCI call oexfet <24>

      Cursor <23> Rows processed 0
 OCI call oclose <23> cc = 2

      Cursor <24> Rows processed 83
      Number of array fetches  6
      Number of rows fetched   83
      Number of array rows     15
      Number of array columns  22
      Number of tables         1
      Space for one row        525
      Requested cache size     8192
      Actual cache size used   7875
 OCI call oclose <24> cc = 1
```

<23> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<24> The single lookahead cursor selects columns directly. It selects all columns because the query does not contain a field list.

Query 7

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK:
    DISPLAY name cust-num postal-code.
END.
```

```
OCI call oopen  <25> cc = 0
 OCI call osql3  <25>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <26> cc = 1
 OCI call oparse <26>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID uni
     que_int_0,CUST_NUM,NAME,POSTAL_CODE FROM SPORTS.CUSTOMER T0
 OCI call oexfet <26>

      Cursor <25> Rows processed 0
 OCI call oclose <25> cc = 2

      Cursor <26> Rows processed 83
      Number of array fetches  1
      Number of rows fetched   83
      Number of array rows     106
      Number of array columns  4
      Number of tables         1
      Space for one row        77
      Requested cache size     8192
      Actual cache size used   8162
 OCI call oclose <26> cc = 1
```

<25> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<26> 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.

Query 8

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK
    QUERY-TUNING (NO-LOOKAHEAD):
    DISPLAY name cust-num postal-code.
END.
```

```
OCI call oopen  <27> cc = 0
 OCI call osql3  <27>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <28> cc = 1
 OCI call oparse <28>
     SELECT /*+ INDEX_ASC(TO CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER TO
 OCI call oexfet <28>
 OCI call oexec  <27>   83 times

      Cursor <27> Rows processed 1
 OCI call oclose <27> cc = 2

      Cursor <28> Rows processed 83
      Number of array fetches  1
      Number of rows fetched   83
      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 oclose <28> cc = 1
```

<27> The DataServer uses the cursor to compare schema information and fetch column values.

<28> 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 83 progress_recid values in the CUSTOMER table.

Query 9

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK
    QUERY-TUNING (NO-LOOKAHEAD):
    DISPLAY name cust-num postal-code.
END.
```

```
OCI call oopen  <29> cc = 0
 OCI call osql3  <29>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <30> cc = 1
 OCI call oparse <30>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID FRO
     M SPORTS.CUSTOMER T0
 OCI call oexfet <30>   83 times

      Cursor <29> Rows processed 1
 OCI call oclose <29> cc = 2

      Cursor <30> Rows processed 83
      Number of array fetches  1
      Number of rows fetched   83
      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 oclose <30> cc = 1
```

<29> The DataServer uses the cursor to compare schema information and fetch column values.

<30> This is a standard cursor. Note that the advantage of using a field list is lost by not using a standard 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.

Query 10

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK,
    EACH order FIELDS (order-date sales-rep) OF customer NO-LOCK:
    DISPLAY customer.name customer.cust-num customer.postal-code
        order.order-date order.sales-rep.
END.
```

```
OCI call oopen  <31> cc = 0
 OCI call osql3  <31>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <32> cc = 1
 OCI call osql3  <32>
     SELECT * FROM SPORTS.ORDER_ WHERE progress_recid = :rid
 OCI call oopen  <33> cc = 2
 OCI call oparse <33>
     SELECT T0.PROGRESS_RECID unique_int_0,T0.CUST_NUM,T0.NAME,T0.POST
     AL_CODE,T1.PROGRESS_RECID unique_int_1,T1.CUST_NUM,TO_CHAR(T1.ORD
     ER_DATE,'YYYYMMDDHH24MISS'),T1.SALES_REP FROM SPORTS.CUSTOMER T0,
     SPORTS.ORDER_ T1 WHERE (T1.CUST_NUM = T0.CUST_NUM)
 OCI call oexfet <33>
          omru   <33>
          omru   <33>
     Cursor <31> Rows processed 0
 OCI call oclose <31> cc = 3
       Cursor <33> Rows processed 207
      Number of array fetches  4
      Number of rows fetched   207
      Number of array rows     57
      Number of array columns  8
      Number of tables         2
      Space for one row        143
      Requested cache size     8192
      Actual cache size used   8151
 OCI call oclose <33> cc = 2
      Cursor <32> Rows processed 0
 OCI call oclose <32> cc = 1
```

<31> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<32> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<33> 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 4 array fetches.

Query 11

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK,
    EACH order FIELDS (order-date sales-rep) OF customer SHARE-LOCK:
    DISPLAY customer.name customer.cust-num customer.postal-code
        order.order-date order.sales-rep.
END.
```

```
OCI call oopen  <34> cc = 0
 OCI call osql3  <34>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <35> cc = 1
 OCI call osql3  <35>
     SELECT * FROM SPORTS.ORDER_ WHERE progress_recid = :rid
 OCI call oopen  <36> cc = 2
 OCI call oparse <36>
     SELECT T0.PROGRESS_RECID unique_int_0,T0.CUST_NUM,T0.NAME,T0.POST
     AL_CODE,T1.PROGRESS_RECID unique_int_1,T1.CUST_NUM,TO_CHAR(T1.ORD
     ER_DATE,'YYYYMMDDHH24MISS'),T1.SALES_REP FROM SPORTS.CUSTOMER T0,
     SPORTS.ORDER_ T1 WHERE (T1.CUST_NUM = T0.CUST_NUM)
 OCI call oexfet <36>
 OCI call oopen  <37> cc = 3
 OCI call osql3  <37>
     SELECT ORDER_NUM,CUST_NUM,TO_CHAR(ORDER_DATE,'YYYYMMDDHH24MISS'),
     TO_CHAR(SHIP_DATE,'YYYYMMDDHH24MISS'),TO_CHAR(PROMISE_DATE,'YYYYM
     MDDHH24MISS'),CARRIER,INSTRUCTIONS,PO,TERMS,U##SALES_REP,SALES_RE
     P,PROGRESS_RECID  FROM SPORTS.ORDER_ WHERE PROGRESS_RECID = :rid
 OCI call oexec  <37>   206 times

      Cursor <34> Rows processed 0
 OCI call oclose <34> cc = 4

      Cursor <36> Rows processed 207
      Number of array fetches  4
      Number of rows fetched   207
      Number of array rows     57
      Number of array columns  8
      Number of tables         2
      Space for one row        143
      Requested cache size     8192
      Actual cache size used   8151
 OCI call oclose <36> cc = 3

      Cursor <37> Rows processed 1
 OCI call oclose <37> cc = 2

      Cursor <35> Rows processed 0
 OCI call oclose <35> cc = 1
```

<34> The DataServer uses the cursor to compare schema information. It does not fetch any column values.

<35> 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.

<36> The cursor is used to perform the join by the SQLDB. The join still requires a lookahead cursor.

<37> Since the query requests the ORDER_ row with a SHARE-LOCK, the DataServer must refetch each ORDER_ row to get all 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 ORDER_ row.

Query 12

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK,
    EACH order FIELDS (order-date sales-rep) OF customer NO-LOCK
    QUERY-TUNING (NO-LOOKAHEAD):
    DISPLAY customer.name customer.cust-num customer.postal-code
        order.order-date order.sales-rep.
END.
```

```
OCI call oopen  <38> cc = 0
 OCI call osql3  <38>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <39> cc = 1
 OCI call osql3  <39>
     SELECT * FROM SPORTS.ORDER_ WHERE progress_recid = :rid
 OCI call oopen  <40> cc = 2
 OCI call oparse <40>
     SELECT T0.PROGRESS_RECID,T1.PROGRESS_RECID FROM SPORTS.CUSTOMER T
     0,SPORTS.ORDER_ T1 WHERE (T1.CUST_NUM = T0.CUST_NUM)
 OCI call oexfet <40>
 OCI call oexec  <38>
 OCI call oopen  <41> cc = 3
 OCI call osql3  <41>
     SELECT ORDER_NUM,CUST_NUM,TO_CHAR(ORDER_DATE,'YYYYMMDDHH24MISS'),
     TO_CHAR(SHIP_DATE,'YYYYMMDDHH24MISS'),TO_CHAR(PROMISE_DATE,'YYYYM
     MDDHH24MISS'),CARRIER,INSTRUCTIONS,PO,TERMS,U##SALES_REP,SALES_RE
     P,PROGRESS_RECID  FROM SPORTS.ORDER_ WHERE PROGRESS_RECID = :rid
 OCI call oexec  <41>   204 times
 OCI call oexec  <38>   204 times
      Cursor <38> Rows processed 1
 OCI call oclose <38> cc = 4
      Cursor <40> Rows processed 207
      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 oclose <40> cc = 3
      Cursor <41> Rows processed 1
 OCI call oclose <41> cc = 2
      Cursor <39> Rows processed 0
 OCI call oclose <39> cc = 1
```

<38> The DataServer uses the cursor to compare schema information and fetch CUSTOMER rows.

<39> 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.

<40> 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.

<41> 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.

Query 13

```
FOR EACH customer FIELDS (name cust-num postal-code) NO-LOCK,
    EACH order FIELDS (order-date sales-rep) OF customer NO-LOCK
    QUERY-TUNING (NO-JOIN-BY-SQLDB):
    DISPLAY customer.name customer.cust-num customer.postal-code
        order.order-date order.sales-rep.
END.
```

```
OCI call oopen  <42> cc = 0
CI call osql3  <42>
     SELECT * FROM SPORTS.CUSTOMER WHERE progress_recid = :rid
 OCI call oopen  <43> cc = 1
 OCI call osql3  <43>
     SELECT * FROM SPORTS.ORDER_ WHERE progress_recid = :rid
 OCI call oopen  <44> cc = 2
 OCI call oparse <44>
     SELECT /*+ INDEX_ASC(T0 CUSTOMER##CUST_NUM) */ PROGRESS_RECID uni
     que_int_0,CUST_NUM,NAME,POSTAL_CODE FROM SPORTS.CUSTOMER T0
 OCI call oexfet <44>
 OCI call oopen  <45> cc = 3
 OCI call oparse <45>
     SELECT /*+ INDEX_ASC(T0 ORDER_##CUST_ORDER) */ PROGRESS_RECID uni
     que_int_0,CUST_NUM,TO_CHAR(ORDER_DATE,'YYYYMMDDHH24MISS'),SALES_R
     EP FROM SPORTS.ORDER_ T0 WHERE CUST_NUM = :1
 OCI call oexfet <45>   82 times
      Cursor <42> Rows processed 0
 OCI call oclose <42> cc = 4
      Cursor <44> Rows processed 83
      Number of array fetches  1
      Number of rows fetched   83
      Number of array rows     106
      Number of array columns  4
      Number of tables         1
      Space for one row        77
      Requested cache size     8192
      Actual cache size used   8162
 OCI call oclose <44> cc = 3
      Cursor <45> Rows processed 4
      Number of array fetches  1
      Number of rows fetched   4
      Number of array rows     124
      Number of array columns  4
      Number of tables         1
      Space for one row        66
      Requested cache size     8192
      Actual cache size used   8184
 OCI call oclose <45> cc = 2
      Cursor <43> Rows processed 0
 OCI call oclose <43> cc = 1
```

<42> The DataServer uses the cursor to compare schema information and fetch CUSTOMER rows.

<43> 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.

<44> The DataServer uses a lookahead cursor to select fields in the field list in addition to those required by the client.

<45> The lookahead cursor selects fields from the ORDER_ table that correspond to a particular CUSTOMER row (WHERE CUST_NUM = :1).

E

Building DataServer Executables

This appendix describes how to build DataServer executables for all possible configurations. Specifically it includes information on the following:

- Overview of the building process
- PROBUILD utility
- Building executables on the UNIX client
- Building executables on the UNIX host
- Setting environment variables
- Linking DataServer executables

E.1 Overview

As of version 9.1, Progress provides a single set of executables (both client and server) that will access any supported version of ORACLE (7.3 or later). These executables dynamically locate and load ORACLE shared libraries on those UNIX platforms where ORACLE supports shared objects. For information on whether of not your version of ORACLE supports shared objects, please 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 LD_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.

Whether or not you can use the single executable depends on your version of UNIX and your version of ORACLE. There are three possible scenarios.

1. ORACLE (Version 7.3 or later) supports shared objects on your version of UNIX and Progress 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 PROBUILD any executables. You do not need to set the LD_LIBRARY_PATH environment variable. You may use `_progres` as the client executable and `_orasrv` as the server executable.

2. ORACLE (Version 7.3 or later) supports shared objects on your version of UNIX, however, Progress is unable to dynamically load the ORACLE shared object when the DataServer connects to your ORACLE database. In this case you do not need to PROBUILD any executables. However, you may use `_progres` to access a remote DataServer only. You may use `orarx` as a self-service client and `_orasrv` as the server executable. In the environment where you want to run `orarx` and__`orasrv` you must set the LD_LIBRARY_PATH environment variable. If either `orarx` or `_orasrv` has the SETUID bit on you must also include a soft link pointing to the ORACLE shared libraries in `/usr/lib`.

3. ORACLE does not provide a shared object. You must PROBUILD your server executable. You may also PROBUILD a self-service client executable (`orarx`). Refer to the following sections of this appendix for instructions. Once you have built your server executable (`_orasrv`), you may use `_progres` to access a remote DataServer only. Because you statically link the necessary ORACLE code when you PROBUILD your server, you do not need to set the LD_LIBRARY_PATH environment variable.

The following sections provide detailed instructions on using the Progress PROBUILD utility to build executables. These instructions apply if you are building custom executables by choice on UNIX, NT, or Windows.

Table E–1 lists the possible configurations that include UNIX and describes the process for setting up the client and server modules.

Table E–1: Building DataServer Processes

Client	Server	Building and Configuring
UNIX	Local UNIX	Use PROBUILD to build a single executable for a UNIX client process that includes the ORACLE DataServer as a configurable element. Build this configuration for local access or if you want the DataServer to access the remote ORACLE instance through ORACLE communications (SQL*Net or Net 8).
UNIX	Remote UNIX	On the UNIX client machine, build a client executable that includes the remote ORACLE DataServer as a configurable element using PROBUILD. Building the client is optional-by default, the UNIX client executable supports access to the remote DataServer. On the UNIX host machine, build two executables: 1) the remote DataServer broker, and 2) the remote ORACLE DataServer.
UNIX	Remote NT	On the UNIX client machine, build a client executable that includes the remote ORACLE DataServer as a configurable element using PROBUILD. Building the client is optional-by default, the UNIX client executable supports access to the remote DataServer. On the NT host machine, configure the default broker executable using ProControl on NT.
Windows	Remote UNIX	On Windows, use the default Progress client executable (`prowin32.exe`). On the UNIX host machine, build two executables: 1) the remote DataServer broker, and 2) the remote ORACLE DataServer.

NOTE: If you are building executables to access ORACLE7 (7.2 and earlier) and ORACLE8, you must build separate executables for each version. When you build an executable, you link to ORACLE libraries that vary across versions.

E.2 PROBUILD and the DataServer

Building a customized executable with PROBUILD involves these general steps:

1. Run the PROBUILD utility on UNIX to build a link script that includes the DataServer objects in your Progress modules. You must build executables on the platform where you'll run them and link to the version of ORACLE which you will access.

2. Set environment variables required during the linking process.

3. Link the objects together to create an executable Progress module. If there are problems running the executables and you are not using SQL*Net or Net 8, make sure that the ORALIB environment variable is not set and relink the executables. Progress sets the ORALIB environment variable properly if it is not set when you run the link scripts.

The PROBUILD utility allows you to customize Progress by letting you include various software modules or products. For each product you select, the PROBUILD utility prompts you to enter a link script name and an executable name. PROBUILD then displays a list of configurable elements for that product. You select the elements you want to include.

E.2.1 PROBUILD Requirements

The PROBUILD utility has software and environment requirements. Follow these steps to set up your environment to run PROBUILD:

1 ♦ Run PROBUILD to create executables on the machine that you plan to run them. For example, if you are building a remote DataServer configuration, to build the broker and the DataServer, run PROBUILD on the host. To build the client, run PROBUILD on the client machine. If you are building executables for deployment, be sure to build the executables on the same platform on which your users will run them.

2 ♦ Make sure that you have a linker installed on your system.

3 ♦ Verify settings for Progress environment variables. Set DLC to the directory where you have installed Progress.

4 ♦ Verify that the `buildenv` script (located in the `$DLC/probuild/eucapp` directory) includes the appropriate environment-variable settings for your system.

The link script that PROBUILD creates calls the `buildenv` script. The `buildenv` 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.

PROBUILD also generates a log file, `buildenv.log`, which 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:

```
Progress V9 Probuild Environment: Wed Nov 4 17:35:47 EDT 1998
------------------------------------------------------------
User:                           uid=850(user1) gid=110(rdl)
Working Directory:              /users/user1/83a/sc
Machine OS:                     HP-UX
Machine OS Version:             B.10.20
Machine Type:                   2012828620
DLC Setting:                    /users/user1/dlc
PROLOAD Setting:                /users/user1/dlcload
User Defined ORALIB:            No, buildenv script defined
ORALIB Setting:
ORACLE_HOME Setting:
LD_LIBRARY_PATH Setting:
LD_RUN_PATH Setting:
PATH Setting:
PROPATH Setting:
```

For more information on the PROBUILD utility, see the *Progress Client Deployment Guide*. For more information on environment variables, see the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

After setting up the PROBUILD environment, you can run PROBUILD to generate link 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.

E.3 Building Executables on the UNIX Client

This section contains instructions for building client executables. To run the PROBUILD utility and generate the link script for the client executable, follow these steps:

1 ♦ Set up the PROBUILD environment as described in the "PROBUILD and the DataServer" section.

2 ♦ Enter **$DLC/probuild/eucapp/probuild** at the system prompt. The Progress End User Configuration screen appears.

3 ♦ Enter the directory where you want to put the link script (and ultimately, the executable for the client). The default is your current directory. If you leave the Install Link Script Field empty, the utility places the link script in your current directory.

4 ♦ Choose Continue. The PRODUCT LIST appears.

5 ♦ Select PROGRESS CLIENT, and choose Continue. The LINK SCRIPT/EXECUTABLE NAMES screen appears.

6 ♦ Type names for the link script and the executable.

The default link script name for the UNIX client is `ldpro`. The default executable name is `_progres`. Progress Software recommends that you do not use the default names for custom executables.

7 ♦ Choose Continue. The CONFIGURABLE ELEMENTS screen appears.

8 ♦ If you are building a client that includes the local DataServer, select ORACLE DATASERVER.

If you are building a client to access a remote DataServer, select REMOTE ORACLE DATASERVER.

You can include other elements from this list to further customize your Progress client module.

9 ♦ Choose Continue to return to the product list.

10 ♦ Choose Return, then choose Exit to exit PROBUILD.

PROBUILD builds a link script that you then run to create a customized Progress executable that references the ORACLE libraries for the DataServer.

Next, if you are building a local DataServer, set the DataServer environment variables and run the link script to create the Progress client executable. See the "Setting Environment Variables" and "Linking DataServer Executables" sections.

If you are building a client to access a remote DataServer, set client DataServer environment variables and run the link script to create the client executable, then set up the host modules for your configuration. See the "Building Executables on the UNIX Host" section.

E.4 Building Executables on the UNIX Host

This section contains instructions for building executables for the broker and the DataServer for a remote DataServer configuration with a UNIX host. You must build these executables on the same type of UNIX machine where you intend to run them.

To run the PROBUILD utility and generate the link script for the broker and DataServer executables, follow these steps on your UNIX host machine. See the "Building Executables on the UNIX Client" section for illustrations of the PROBUILD interface:

1 ♦ Set up the PROBUILD environment as described in the "PROBUILD and the DataServer" section.

2 ♦ Enter **$DLC/probuild/eucapp/probuild** at the system prompt.

3 ♦ Enter the directory where you want to put the link script (and ultimately, the executable for the broker). The default is your current directory. If you leave the Install Link Script Field empty, the utility places the link script in your current directory.

4 ♦ Choose Continue. A list of products appears.

5 ♦ Browse through the list and select REMOTE DATASERVER BROKER, then choose Continue.

6 ♦ Type names for the link script and the executable.

The default link script name for the broker is `ldprobrkr`. The default executable name is `_probrkr`. Progress Software recommends that you do not use the default names for custom executables.

7 ♦ Choose Continue. A list of configurable elements appears.

8 ♦ Select the appropriate network protocol, then choose Continue. The list of products appears again.

9 ♦ Select REMOTE ORACLE DATASERVER from the product list.

10 ♦ Type names for the link script and the executable, or accept the default.

The default link script name for the DataServer is `ldorasrv`. The default executable name is `_orasrv`.

11 ♦ Choose Continue. The list of configurable elements appears.

12 ♦ Select the appropriate network protocol, then choose Continue. PROBUILD generates the link scripts or command procedures for the DataServer.

13 ♦ Choose Return, then choose Exit to exit PROBUILD.

PROBUILD builds a link script that you then run to create a customized Progress executable that references the ORACLE libraries for the DataServer.

Next, you must set environment variables before running the link script to create the Progress executables. See the "Setting Environment Variables" and "Linking DataServer Executables" sections.

E.5 Setting Environment Variables

After you run the PROBUILD utility to create your link scripts or command procedures, you must set some of the DataServer environment variables or logicals on the host machine before you run the scripts to create the executables. The others you can set after linking the executables. Be sure to set the environment variables in the same environment (UNIX shell) from which you plan to link and run the executables. Table E–2 describes how to set them for the local DataServer and Table E–3 describes how to set them for the remote DataServer.

Table E–2: Environment Variables for the Local DataServer *(1 of 2)*

Variable	How to Set It
DSLOGDIR	The pathname of the log file that Progress uses to keep track of DataServer processes and error messages. By default, Progress writes to `$DSLOGDIR/dataserv.lg` in the current directory. (Optional)
ORACLE_HOME	The pathname of the directory where you installed ORACLE. (Required before running link scripts)
ORACLE_SID	The identifier for the ORACLE instance you are accessing. (Required before running link scripts)
ORALIB	The identifier for the ORACLE libraries which must be linked into the executable you created with PROBUILD. The `buildenv` script automatically sets this variable for you if it is not already set. If there are problems running the executables, see the "Setting ORALIB" section for troubleshooting instructions. (Required before running link scripts for some configurations)

Table E–2: Environment Variables for the Local DataServer *(2 of 2)*

Variable	How to Set It
PROEXE	The pathname of the Progress client executable you created with PROBUILD. After setting this variable, you can run your customized executable by entering **pro** at the system prompt. Make sure that you have set DLC to the directory where Progress is installed. (Optional)
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	Identifies ORACLE shared libraries. Set this variable to include `$ORACLE_HOME/lib`. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. (Required when running executables that access ORACLE7 or ORACLE8)

Table E–3: Environment Variables for the Remote DataServer *(1 of 2)*

Variable	How to Set It
DSLOGDIR	The pathname of the log file that Progress uses to keep track of DataServer processes and error messages. By default, Progress writes to `$DSLOGDIR/dataserv.lg` in the current directory. Set this variable on the host machine. (Optional)
ORACLE_HOME	The pathname of the directory where you installed ORACLE. Set this variable on the host machine. (Required before running link scripts)
ORACLE_SID	The identifier for the ORACLE instance you are accessing. Set this variable on the host machine. (Required before running link scripts)
ORALIB	The identifier for the ORACLE libraries which must be linked into the executable you created with PROBUILD on the host machine. The `buildenv` script automatically sets this variable for you. If there are problems running the executables, see the "Setting ORALIB" section for troubleshooting instructions. Set this variable on the host machine. (Required before running link scripts for some configurations)

Table E–3: Environment Variables for the Remote DataServer *(2 of 2)*

Variable	How to Set It
ORASRV	The name of the executable (including the path) of the ORACLE DataServer. Set it to the DataServer executable you created with PROBUILD. Set this variable on the host machine. (Optional)
PROBRKR	The pathname to the executable of the broker. Set it to the broker executable for the remote DataServer you created with PROBUILD. Set this variable on the host machine. (Optional)
PROEXE	The pathname of the Progress client executable you created with PROBUILD. After setting this variable, you can run your customized executable by entering **pro** at the system prompt. Make sure that you have set DLC to the directory where Progress is installed. Set this variable on the client machine. (Optional)
LD_LIBRARY_PATH (Sun Solaris) or LIBPATH (AIX) or SHLIB_PATH (HP-UX)	Identifies ORACLE shared libraries. Set this variable on the host machine to include `$ORACLE_HOME/lib`. On your UNIX system, if the SETUID bit is on, make sure there is a soft link from `/usr/lib` pointing to the ORACLE shared libraries. Note that ORACLE does not support shared objects on all UNIX platforms. See your ORACLE documentation for more information. (Required when running executables that access ORACLE7 or ORACLE8)

Once you set these variables, you can run the link scripts you created with PROBUILD to generate executables. See the "Linking DataServer Executables" section for instructions.

E.5.1 Linking DataServer Executables

After using PROBUILD to generate the link scripts for the DataServer executables and setting environment variables as required, you must run the link scripts to create the executables.

Run the link scripts that you generated with PROBUILD by entering the link script name at the system prompt.

Now you can use the executables to start the DataServer and build a schema holder. See the "Creating a Schema Holder" section in Chapter 3, "Configuring the DataServer," for instructions.

E.5.2 Setting ORALIB

Before you run the link scripts to generate executables that include the DataServer module, the ORALIB environment variable must point to the appropriate ORACLE libraries. The `buildenv` script, which the link scripts call, sets ORALIB for you. Try to run the link script or scripts that you created with PROBUILD before setting ORALIB yourself.

NOTE: ORALIB settings vary depending on your installation of ORACLE and the ORACLE version you are accessing. You must reset ORALIB if you want to access a different version of ORACLE.

Do not set ORALIB for executables that access ORACLE 7.3 or ORACLE8. Setting LD_LIBRARY_PATH, LIBPATH, or SHLIB_PATH when you run those executables provides the required library information.

The automatic setting of ORALIB might fail on some platforms. Indications that ORALIB is set incorrectly are:

- Error messages about undefined symbols when you run the link script. The directory associated with these undefined symbols can help you isolate the problem:
  - A message about the `/usr` directory indicates a UNIX configuration problem.
  - A message about an ORACLE directory indicates an ORACLE configuration problem.
  - A message about the `/dlc` directory indicates a Progress DataServer configuration problem.
- Error messages about files not being opened.
- Executable is created, but it does not have the ability to execute.

These are alternate techniques for setting ORALIB:

- Use a shell script to set ORALIB
- Set ORALIB manually

If you need to probuild the ORACLE executables on HP-UX with ORACLE 7.3.3 and 7.3.4, you must set the ORALIB environment variable manually to include an additional library. This additional library is cma or /usr/lib/libcma.sl which is a subset of dce. The automatic script does not include this library. If you use ORACLE 7.3.3, or 7.3.4, you need to get a patch from ORACLE and regenerate the the libclntsh.sl.

Set the environment variable as follows:

```
ORALIB="-L/usr/lib -lclntsh -lcma"; export ORALIB
```

When PROBUILD generates ldpro and ldorasrv, they pick up this definition and build the orasrv and _progres executables with the shared library.

Using a Shell Script to Set ORALIB

If you know the appropriate value for ORALIB given your installation of ORACLE, write a shell script named `oralib.sh` to set ORALIB to that value as follows:

```
ORALIB="value" ; export ORALIB
```

Setting ORALIB Manually

Follow these steps to determine possible values for ORALIB using the ORACLE PRO*C sample program.

1 ♦ On your host machine, set PATH to `$PATH:$ORACLE_HOME/bin`.

2 ♦ Make sure that you have installed the PRO*C product for ORACLE on your host machine. PRO*C contains the libraries that allow you to access the ORACLE database, the sample make file, and the sample C program.

For Version 7.3 and later, these files typically reside in `$ORACLE_HOME/rdbms/demo/`. Earlier versions resided in `$ORACLE_HOME/rdbms/lib/`

3 ♦ Copy the files from `demo/` to another directory.

4 ♦ Delete the `sample.o` file, if it exists.

5 ♦ Enter one of the following commands to use the make file to link the ORACLE sample C program and direct the output to a temporary file. On some platforms, the sample C program is called `samplec`:

ORACLE 7.3

```
make -f $ORACLE_HOME/rdbms/demo/oracle.mk > tmpfile
```

ORACLE8

```
make -f $ORACLE_HOME/rdbms/demo/demo_rdbms.mk > tmpfile
```

You must have read, write, and execute permissions to run the make file.

If you link the sample C program successfully, the make command displays a list of ORACLE libraries in the order it uses them. It displays some libraries more than once. Some of the library names are preceded by a -l. The -l indicates that an alias for a library name is being used.

If you still cannot link the sample C program successfully, see the ORACLE documentation for information on linking the program.

6 ♦ Edit the `tmpfile` as follows:

- Delete the beginning of the file up to and including `-o sample sample.o`.
- For ORACLE 7.2, 7.3, and ORACLE8 add `$ORACLE_HOME/lib/libclient.a` as the first entry in the file.
- Delete all entries of -l*lib-name* for which there is not a corresponding entry of lib*lib-name* in `$ORACLE_HOME/lib`.
- Delete the -lm at the end of the file, if applicable.
- For the libraries that exist in your `ORACLE_HOME/lib` directory, replace all the -l prefixes with `$ORACLE_HOME/lib/lib` and add a .a extension. For example, change `-lsqlnet` to `$ORACLE_HOME/lib/libsqlnet.a` if this library exists in your `$ORACLE_HOME/` directory.

This is a sample make file. Strike-through lines indicate the information that you must delete:

```
~~/usr/5bin/cc -target sun4 -c sample.o~~
~~/usr/5bin/cc -Bdynamic L/gateways/sun4/oracle/produc/7.0.15/lib -o~~
~~sample sample.0~~ /gateways/sun4/oracle/produc/7.0.15/lib/libocic.a
/gateways/sun4/oracle/produc/7.0.15/lib/osntab.o -lsqlnet -lora
/gateways/sun4/oracle/produc/7.0.15/lib/libpls.a -lsqlnet
-lnlsrtl -lcv6 -lcore -lnlsrtl -lcv6 -lcore ~~-lm~~
```

7 ♦ Add the appropriate commands and symbols to the `tmpfile` so that the remaining text becomes a setting for the ORALIB environment variable. For example, add the `setenv` command to the remaining text.

```
setenv ORALIB "..."
```

If you are using a Bourne shell, add the following command to the remaining text.

```
ORALIB="..."
```

8 ♦ Export the environment variable.

9 ♦ Run the `tmpfile` from the system prompt within the same shell to set ORALIB.

If the following message appears, you can ignore it:

```
xlc: cannot write to directory
```

E.5.3 Rebuilding Libclnt.so

On Solaris, libclnt.so has an unintended dependency on the liboo library; therefore, you must rebuild the libclnt.so as described on ORACLE's web site. In $ORACLE_HOME/bin, the script genclntsh needs to be modified by commenting out the lines:

```
ar d $LIBCOMMON sorapt.o
```

Execute this script to regenerate libclnt.so.

F

DataServer Command Line Utilities and Startup Parameters

This appendix describes the following utilities and parameters that you use to configure, manage, start, and stop the DataServer host and client.

- Command line utilities allow you to start, stop, and configure installed ORACLE components. See the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX* for additional information about the utilities and their role and relationship to other system administration facilities.
- Startup parameters for UNIX and Windows allow you to start and manage DataServer clients. See the *Progress Startup Command and Parameter Reference* for additional information about syntax and usage.

F.1 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. It discusses the purpose, syntax, and primary parameters for each operating system.

F.1.1 NSCONFIG Utility

Use the NSCONFIG utility to help you debug existing NameServer configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with a NameServer configuration, and checks that the syntax and values are valid.

The NSCONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network:

SYNTAX

Operating System	Syntax
UNIX Windows	`nsconfig [[[-name name-server] [-propfile path-to-properties-file] [-validate]]	-help]`

PARAMETERS

`-name name-server`

Specifies which existing NameServer configuration to examine. The name must match the name of an existing NameServer configuration in the specified properties file. If you do not specify a NameServer, the NSCONFIG utility analyzes all NameServer configurations defined in the properties file specified by the `-propfile` parameter.

`-propfile` *`path-to-properties-file`*

Specifies a filename or pathname to a file that contains the property settings to be validated, for example `test.properties`. If a filename or pathname is not specified, it defaults to the installation version of the `ubroker.properties` file, such as:

- `%DLC%\properties\ubroker.properties` on Windows NT
- `$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 NT, 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 Progress installation directory. For example, `$DLC/properties/ubroker.properties` on UNIX, or `%DLC%\properties\ubroker.properties` on Windows NT.

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` may include:

Configuration Entity Name	Configuration Entity Function
`[UBroker]`	Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.
`[NameServer]`	Defines default property settings for all instances of a NameServer.
`[NameServer.`*`product-instance-name`*`]`	Defines property settings for this instance of an NameServer. The `ubroker.properties` file may contain several of these entities each with a unique *product-instance-name*.

Parent entities provide default values for all of their child entities. For example, the parent `[UBroker]` contains a set of definitions that can be inherited by its child `[NameServer]`, and then again by its child `[NameServer.`*`product-instance-name`*`]`. However, at any child level, a redefinition of any value supercedes 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 NT 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.

F.1.2 NSMAN Utility

Use the NSMAN utility to control the operation of a configured NameServer. The utility allows you to start a NameServer, query its status, and shut down a NameServer:

SYNTAX

Operating System	Syntax

UNIX Windows	nsman
	{
	{ -name *name-server*
	{ -kill | -start | -stop | -query }
	[
	-host *host-name* -user *user-name* | -user *user-name*
	]
	[-port *port-number*]
	}
	| -help
	}

PARAMETERS

`-name` *name-server*

This parameter is required. It specifies the name of the NameServer.

`-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 NT 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 NT 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 NT 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 NT Server, including the one where the AdminServer is running.

F.1.3 ORACONFIG Utility

Use the ORACONFIG utility to help you debug existing DataServer for ORACLE configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with a DataServer for ORACLE configuration, and checks that the syntax and values are valid.

The ORACONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network:

SYNTAX

Operating System	Syntax
UNIX Windows	`oraconfig [ [ [ -name DataServer-name ] [ -propfile path-to-properties-file ] [ -validate ] ] | -help ]`

PARAMETERS

`-name` *DataServer-name*

Specifies which existing DataServer for ORACLE configuration to examine. The name must match the name of an existing DataServer for ORACLE configuration defined in the specified properties file. If you do not specify a DataServer by name, the ORACONFIG utility analyzes all 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` on Windows NT

- `$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 NT, 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 Progress installation directory. For example, `$DLC/properties/ubroker.properties` on UNIX, or `%DLC%\properties\ubroker.properties` on Windows NT.

  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 DataServer for ORACLE configurations in `ubroker.properties` may include:

Configuration Entity Name	Configuration Entity Function
`[UBroker]`	Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.
`[UBroker.OR]`	Defines default property settings for all instances of a DataServer for ORACLE.

Configuration Entity Name	Configuration Entity Function
[UBroker.OR.*product-instance-name*]	Defines property settings for this instance of a DataServer for ORACLE. The ubroker.properties file may contain several of these entities each with a unique *product-instance-name*.

Parent entities provide default values for all of their child entities. For example, the parent [UBroker] contains a set of definitions that can be inherited by its child [UBroker.OR], and then again by its child [UBroker.OR.*product-instance-name*]. However, at any child level, a redefinition of any value supercedes 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 these products within a UNIX or Windows NT 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.

F.1.4 ORAMAN Utility

Use the ORAMAN utility to control the operation of a configured DataServer for ORACLE. The utility allows you to start a broker, query its status, and start and stop additional DataServer servers, and shut down the broker:

SYNTAX

Operating System	Syntax

UNIX Windows	oraman
	{
	{ -name *DataServer-name*
	{ -kill | -start | -stop | -query }
	[
	-host *host-name* -user *user-name* | -user *user-name*
	]
	[-port *port-number*]
	}
	| -help
	}

PARAMETERS

-name *DataServer-name*

This parameter is required. It specifies the name of a DataServer.

-kill

Stops and removes the DataServer from memory, no matter what it is doing.

-start

Starts the DataServer.

-stop

Tells the DataServer to stop itself.

`-query`

Queries the DataServer for its status.

`-host` *`host-name`*

Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.

`-user` *`user-name`*

Specifies a user name and prompts for a password. A user name and password are required only when you use the `-host` parameter and specify a remote host name. If you specify a remote host name with the `-host` parameter, but do not specify a user name with the `-user` parameter, you receive a prompt for a user name and password.

`-port` *`port-number`*

Specifies the port number of the machine on which the AdminServer is running. If a port number is not specified, it defaults to 20931.

`-help`

Displays command-line help.

NOTES

- When you specify a user name with the `-user` parameter, Windows NT 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 NT 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 NT 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 NT Server, including the one where the AdminServer is running.

F.1.5 PROBRKR Command

Starts the DataServer broker. To use the DataServer from a remote client, you must first start the broker. Once you start the broker, it can receive the client requests and spawn the appropriate DataServer:

SYNTAX

Operating System	Syntax
UNIX Windows	`probrkr` *dbname* `-S` *service-name* [`-H` *host-name*] [`-N` *network-type*]

PARAMETERS

`dbname`

Specifies the name of the database where you are connecting to.

`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.

`network-type`

Specifies the network protocol for which the broker will run.

Progress supports the TCP protocol.

NOTES

- See the*Progress Startup Command and Parameter Reference*for more details on the Server Name (-S), Host Name (-H), and Network Type (-N) startup parameters.
- You can use any of the startup parameters with the PROBRKR command. See the *Progress Startup Command and Parameter Reference* for details.

- You must start the remote broker in the same environment in which your ORACLE data source names (DSNs) are defined because the servers spawned by the broker inherit the setup of the environment from the broker. For example, set the environment variable ORASRV to the name of the executable (including the path) of the DataServer for ORACLE. Be sure to set this variable on the host machine. Also, in the same environment, make sure you have set all ORACLE environment variables required to connect to the data source. See Chapter 3, "Configuring the DataServer," for examples of required variables.
- Start the broker on a node that is locally connected to the disk containing the data source.

F.1.6 PROSHUT Command

Shuts down the Progress database server and individual Progress processes. Before you shut down the server, have all application users quit their Progress sessions. If necessary, you can disconnect users by using the PROSHUT command's Disconnect a User or Unconditional Shutdown parameters:

SYNTAX

Operating System	Syntax
UNIX NT	proshut *db-name* [-b | -by | -bn | -H *host-name* | -S *service-name* | -N *network-type* | -F | -Gw | -C list | -C disconnect usernum] ...

PARAMETERS

db-name

Specifies the database the server is running against.

`-b`

Directs Progress to perform a batch shutdown. When no client is connected, the database automatically shuts down. When one or more clients are connected, PROSHUT prompts the user to enter "yes" to perform an unconditional batch shutdown and to disconnect all active users; or "no" to perform a batch shutdown only if there are no active users. The -b parameter combines the functionality of the -by or -bn parameters.

`-by`

Directs Progress to perform an unconditional batch shutdown and to disconnect all active users.

Using -by in conjunction with -F causes an emergency shutdown.

`-bn`

Directs Progress to perform a batch shutdown only if there are no active users.

`-H` *`host-name`*

Specifies the machine where the database server runs. You must specify the host name if you issue the shutdown command from a machine other than the host.

`-N` *`network-type`*

Specifies the networking protocol used by the requesting application to connect to the Progress DataServer. This optional parameter is always TCP.

`-S` *`service-name`*

Specifies the database server or broker process. You must specify the service name if you issue the shutdown command from a machine other than the host.

`-F`

Starts an emergency shutdown. To use this parameter, you must run PROSHUT on the machine where the server resides, and on UNIX systems. This parameter is not applicable for remote shutdowns or DataServer shutdowns.

`-Gw`

For DataServers, specifies the DataServer broker to shut down.

When you enter the PROSHUT command without the -by, -bn, or -F parameters, the following menu appears:

```
1 Disconnect
2 Unconditional Shutdown
3 Emergency Shutdown (Kill All)
x Exit
```

The following table lists the menu options and their actions:

Option	Action
1	Prompts you for the number of the user you want to disconnect.
2	Disconnects all users and shuts down the database.
3	Prompts you to confirm your choice. If you cancel the choice, you cancel the shutdown. If you confirm the choice, Progress displays the following message: `Emergency shutdown initiated...` PROSHUT marks the database for abnormal shutdown, kills all remaining processes connected to the database, and deletes shared-memory segments and semaphores. The database is in a crashed state. Progress performs normal crash recovery when you restart the database and backs out any active transactions.
x	Cancels the shutdown without taking any action.

`-C list`

Lists all of the users connected to the database. The list is printed out to the screen without any page breaks.

`-C disconnect` *`usernum`*

Allows you to initiate a disconnect for the specified user. This is similar to option 1 of the PROSHUT menu.

NOTES

- The PROSHUT command runs the following executable:

```
_mprshut
```

- You can access PROSHUT from the PROMON utility by using the Shut Down Database qualifier.
- The user who shuts down the server must have started it, or be root (on UNIX).
- When you initiate PROSHUT over a network, the amount of time that it takes to actually shut down all of the Progress processes and to free any ports varies depending on the number of clients, brokers, and servers that must be shut down. The PROSHUT utility might return control to the terminal before all of the processes are stopped and resources are freed.

F.2 DataServer Startup Parameters for UNIX and Windows

Table F–1 lists the UNIX and Windows parameters that you use with the command line utilities to start a DataServer for ORACLE.

Table F–1: DataServer Parameters for UNIX and Windows

Parameter	Syntax
DataServer	-Dsrv *keyword, value*
Database Type	-dt *db-type*
Host Name	-H *host-name*
Service name	-S *service-name*
Network protocol	-N *protocol-type*
Data Source User Name	-U *user-name*
Data Source User Name Password	-P *password*
Server Join	-nojoinbysqldb

Index

Numbers

-1 parameter 4–13

A

Abbreviated index 2–35

Aggregates
  views 2–10, 2–11

Analyzing performance 4–19

ARRAY-MESSAGE option 2–51

Arrays 2–18, 5–23
  joins 2–59

Audience xi

Auto-connect
  failure 4–23

B

-b startup parameter
  PROSHUT command F–14

BEGINS operator 2–35

BFILE data type 2–18

Bind variables 2–49
  queries 2–51

BIND-WHERE option 2–51

Blank padding
  CHAR data type 2–16

Blanks
  leading and trailing 2–7

-bn startup parameter
  PROSHUT command F–14

Bold typeface
  as typographical convention xiii

Broker
  running multiple 3–4, 3–6, 3–7, 4–3
  starting on NT 4–6
  starting on UNIX 4–6

Brokers
  DataServer F–13
  shutting down
    permissions F–16

BY option
  joins 2–59

-by startup parameter
  PROSHUT command F–14

C

-C disconnect usernum qualifier
PROSHUT command F–15

-C list qualifier
PROSHUT command F–15

-c startup parameter 4–16

-cache startup parameter 4–16

CACHE-SIZE option 2–52

Caching
schema 1–6

Case-insensitive indexes 2–8, 5–22

CHAR data type 2–16
blank padding 2–16

CHARACTER data type 2–16

Character sets 2–4
changing in schema holders 5–19

CLOB data type 2–18

CLOSE QUERY statement 2–29

CLOSE STORED-PROCEDURE statement 2–40, 2–41, B–2

Code pages 2–4
changing in schema holders 5–19
collation table 2–5
conversion table 2–5
double-byte 3–12

Column name 5–24

Column width 5–24

COMPILE statement 2–33

CONNECT statement 4–9, 4–23

Connecting schema holders 4–9

Connection failures
with ORACLE 4–23

Connection parameters 4–10
network options 4–10
unsupported 4–15

CONTAINS operator 2–35

COUNT-OF function 2–35

-cpinternal startup parameter 2–5, 3–13, 4–12, 5–27

-cpstream startup parameter 3–13, 4–12, 5–27

CREATE statement 2–22, 2–36

Creating schema holders 3–18

CURRENT-VALUE function 2–36

CURRENT-VALUE statement 2–36

Cursor repositioning 2–28, 2–33

Cursors 2–27, 2–52, 2–55, 4–21
stored procedures 2–42

D

Data definitions 1–6

Data Dictionary 5–14

Data types 2–15
BFILE 2–18
changing 5–16
CHAR 2–16
CHARACTER 2–16
CLOB 2–18
DATE 2–17, 2–21
DECIMAL 2–16
display format 5–16
INTEGER 2–17
LOB 2–18
LOGICAL 2–17
LONG 2–11, 2–16, 2–32
LONG RAW 2–16
NUMBER 2–16
RAW 2–18
VARCHAR2 2–16

Database conversion
  Progress-to-ORACLE 5–22, 5–27

Database Type (-dt) startup parameter 4–15

Databases
  naming conventions 5–25
  objects 3–24
  Progress objects 2–2
  schema 1–6
  triggers 2–12

DataServer (-Dsrv) startup parameter 4–16
  qt_cache_size 2–32

DataServers
  architecture 1–3
  broker 3–7, 4–24
  building executables E–4
  compiling applications 2–33
  cursor repositioning 2–28
  internal logic 1–5
  log file 3–8, 3–9, 4–2, 4–7, C–2, E–8, E–9
  networks 1–13
  record creation 2–22
  starting a broker F–12
  starting MS-Windows client 4–8
  starting NT client 4–8
  starting UNIX client 4–8
  stopping UNIX processes 4–8
  transactions 2–25

DATE data type 2–17, 2–21

-db startup parameter 4–12

DBA privileges 3–18, 3–19

DBE DataServer
  startup parameters 3–13, 4–12, 5–27

db-name startup parameter
  PROSHUT command F–13

DBRESTRICTIONS function 2–34

DEBUG EXTENDED option 2–53

DEBUG SQL option 2–53

DECIMAL data type 2–16

DEFINE BROWSE statement 2–31

Deleting schema holders 5–20

Demonstration databases
  ORACLE 5–2

Deploying schema holders 3–27

Disconnect usernum qualifier
  PROSHUT command F–15

Display formats 5–16, 5–25

Distributed databases 3–24
  connecting 4–13
  synonyms 2–15
  two-phase commit 2–26

DLC environment variable C–2

DLCLOAD environment variable C–2

DSLOGDIR environment variable 3–8, 3–9, 4–2, 4–7, C–2, E–8, E–9

-Dsrv startup parameter 2–32, 4–16

-dt startup parameter 4–15

E

Environment variables
  DLC C–2
  DLCLOAD C–2
  DSLOGDIR 3–8, 3–9, 4–2, 4–7, C–2, E–8, E–9
  LD_LIBRARY_PATH 4–2, 4–7, C–2, E–9, E–10
  LIBPATH (AIX) 3–8, 3–9, 4–2, 4–7, C–2, E–9, E–10
  ORACLE_HOME 3–8, 3–9, 3–20, 4–2, 4–7, C–2, E–8, E–9
  ORACLE_SID 3–8, 3–9, 3–20, 4–2, 4–7, 4–24, C–2, E–8, E–9
  ORALIB E–8, E–9, E–13
  ORASRV 3–9, 4–7, C–2, E–10
  ORAVERSION C–2

PROBRKR 3–10, 4–7, C–2, E–10
PROEXE 3–8, 3–10, C–3, E–9, E–10
SHLIB_PATH (HP-UX) 3–8, 3–9, 4–2, 4–7, C–2, E–9, E–10
TNS_ADMIN C–3
UNIX E–4

Error handling 2–26

Error messages
displaying descriptions xvii
incomplete 3–26
ORACLE 3–26
troubleshooting 4–23

EXCLUSIVE-LOCK option 2–24, 2–55, 2–59

Executables
UNIX E–10

External data types 2–15

F

-F startup parameter
PROSHUT command F–14

Field extents. *See* also Arrays

Field lists 2–31
updating records 2–58

Field properties
modifying 5–16

FIELDS option 2–31
SHARE-LOCK 2–37

FIND FIRST statement 2–33

FIND LAST statement 2–33

FIND PREV statement 2–33

FIND statement 2–6, 2–23
QUERY option 2–36

FOR EACH statement 2–6

Force Access (-F) startup parameter
PROSHUT command F–14

Formats 5–16, 5–25

Functions
CURRENT-VALUE 2–36
DBRESTRICTIONS 2–34
MATCHES 2–35
double-strings 2–36
NEXT-VALUE 2–36
PROC-HANDLE 2–47
PROC-STATUS 2–41
RECID 2–23, 2–29
ROWID 2–29, 3–25
ROWID data type 2–15
SETUSERID 2–37
USERID 2–37, 4–14

G

-Gw startup parameter
PROSHUT command F–14

H

-H startup parameter 4–10
PROSHUT command F–14

handles
SESSION:TIME-SOURCE 2–37

Help
Progress messages xvii

HINT option 2–53

Hints 2–7, 2–54, 2–57

Host Name (-H) startup parameter 4–10
PROSHUT command F–14

I

Index Cursors (-c) startup parameter 4–16

Index Hint (-noindexhint) startup parameter

2–7, 2–54, 4–19

Index hints 2–7

Indexes 2–6
  abbreviated 2–35
  case-insensitive 2–8, 5–22
  repositioning 2–9
  word 1–1

INDEX-HINT option 2–54

INPUT-OUTPUT option 2–40, 2–41, B–6

Instances 1–6

INTEGER data type 2–17

Integrity constraint 2–2

Internal Code Page (-cpinternal) startup parameter 2–5

Italic typeface
  as typographical convention xiii

J

JOIN-BY-SQLDB option 2–54, 2–58

Joins 2–10, 2–11

K

Keystrokes xiii

Kill users (-by) startup parameter
  PROSHUT command F–14

L

-ld startup parameter 4–15

LD_LIBRARY_PATH environment variable 3–8, 3–9, 3–8, 3–9, 4–2, 4–7, C–2, E–9, E–10

Libclnt.so E–14
  rebuilding on Solaris E–14

LIBPATH (AIX) environment variable 3–8, 3–9, 4–2, 4–7, C–2, E–9, E–10

Libraries
  additional on HP-UX E–11
  linking E–13

List qualifier
  PROSHUT command F–15

Literals
  queries 2–51

LOB data type 2–18

Log files 3–8, 3–9, 4–2, 4–7, C–2, E–8, E–9

LOGICAL data type 2–17

Logical Database Name (-ld) startup parameter 4–15

Logical database names 3–23, 4–15
  schema holders 5–13

LONG column 2–11, 2–32

LONG data type 2–16

LONG RAW data type 2–16

Lookahead cursors 2–52, 2–55

LOOKAHEAD option 2–55

M

Make file E–12

Manual
  organization of xi
  syntax notation xiv

MATCHES function 2–35
  double-byte strings 2–36

Messages
  displaying descriptions xvii
  ORACLE 3–26

Modifying schema holders 5–14

Monospaced typeface
as typographical convention xiii

Multi-table views 2–11

N

-N startup parameter 4–10
PROSHUT command F–14

Naming conventions 2–3, 5–25

National Language Support (NLS)
DataServer environment 3–11
stored procedure 2–45

Net 8 4–13

Network Type (-N) startup parameter 4–10
PROSHUT command F–14

NEXT-VALUE function 2–36

No kill users (-bn) startup parameter
PROSHUT command F–14

NO-ARRAY-MESSAGE option 2–51

NO-BIND-WHERE option 2–51

NO-DEBUG option 2–53

NO-ERROR option 4–23
stored procedures B–7

NO-INDEX-HINT option 2–7, 2–54

-noindexhint startup parameter 2–7, 2–54, 4–19

NO-JOIN-BY-SQLDB option 2–54, 2–58

-nojoinbysqldb startup parameter 2–54, 2–58, 4–19

NO-LOOKAHEAD option 2–55

Non-unique table names 2–3

NOT NULL 2–20

NSCONFIG F–2

NSMAN F–5

Null 2–19

NUMBER data type 2–16

Number of Databases (-h) startup parameter 3–26, 5–4

O

OCI. *See* also ORACLE Call Interface

OF option 2–6

OPEN QUERY statement 2–23, 2–37

ORACLE
CHAR data type 2–16
data types
external 2–15
internal 2–15
DATE data type 2–17
DBA privileges 3–18
demonstration database 5–2
distributed database 3–24, 4–13
integrity constraint 2–2
linking libraries E–13
LONG data type 2–16
LONG RAW data type 2–16
naming conventions 2–3, 5–25
Net 8 4–13
non-unique table names 2–3
NOT NULL 2–20
NUMBER data type 2–16
password 2–37, 3–19, 4–12, 4–24
permissions 3–18
RAW data type 2–18
record locking 2–24
ROWID 2–15
ROWID data type 2–15
running 3–20
schema objects 2–2
size limitations 5–23
SQL*Net 4–13

stored procedures 2–37
Transaction Processing option 2–23
triggers 2–12
user ID 2–37, 3–19, 4–13, 4–24
VARCHAR2 data type 2–16
views 2–10
WHERE clause 2–37

ORACLE 8 1–16
creating schema holder 3–24
upgrading to A–6

ORACLE Call Interface 1–6

ORACLE utilities 3–22

ORACLE_HOME environment variable 3–8, 3–9, 3–20, 4–2, 4–7, C–2, E–8, E–9

ORACLE_SID environment variable 3–8, 3–9, 3–20, 4–2, 4–7, 4–24, C–2, E–8, E–9

ORACLE8
upgrading from ORACLE7 1–19

ORACONFIG F–7

ORALIB environment variable E–8, E–9, E–13

ORAMAN F–10

ORASRV
setting in ProControl 3–6

ORASRV environment variable 3–9, 4–7, C–2, E–10

ORAVERSION environment variable C–2

ORDERED-JOIN option 2–55

Ordering results
USE-INDEX 2–7

OUTPUT option 2–40, 2–41, B–6

Output parameters
stored procedures 2–41

P

-P startup parameter 4–12

PARAM option 2–41

Parameter files 4–13

Parameters
stored procedures 2–41

Password (-P) startup parameter 4–12

Passwords 2–37, 3–19, 4–12, 4–24

Performance
analyzing 4–19

Permissions 3–18

Physical Database Name (-db) startup parameter 4–12

Physical database names 4–12

PL/SQL
sending statements 2–45

PRO*C 1–6, E–12

PRO/SQL utility 5–20

PROBRKR command F–12

PROBRKR environment variable 3–10, 4–7, C–2, E–10

PROBUILD utility E–4

Procedures
compiling 2–33

PROC-HANDLE function 2–47, B–2, B–4

PROC-STATUS function 2–41, B–2, B–5

proc-text-buffer 2–40, B–4, B–7

PROEXE environment variable 3–8, 3–10, C–3, E–9, E–10

Progress
  CHARACTER data type 2–16
  Data Dictionary 5–14
  data types 2–15
  database objects 2–2
  DECIMAL data type 2–16
  INTEGER data type 2–17
  LOGICAL data type 2–17
  naming conventions 2–3, 5–25
  ORACLE utilities 3–22, 5–2
  size limitations 5–23
  triggers 2–12
  two-phase commit 2–25
  unknown value 2–19

Progress SQL-92
  data types 2–18

Progress_RECID
  creating in ORACLE A–3

Progress_RECID column 2–32

Progress-to-ODBC utility
  parameters 5–2

Progress-to-ORACLE utility 5–21, 5–27

PROLOAD E–5

PROLOAD environment variable E–4

PROSHUT command F–13

protoora utility 5–21, 5–27
  batch mode 5–30

Q

qt_bind_where option 4–17

qt_cache_size option 4–17

qt_debug
  EXTENDED option 4–18
  SQL option 4–18

qt_lookahead option 4–18

qt_no_bind_where option 4–17

qt_no_debug option 4–18

qt_no_lookahead option 4–18

Queries
  records not visible 2–23, 2–37
  USING option 2–35

Query tuning
  startup parameters 4–19

QUERY-TUNING phrase 2–7, 2–49

R

RAW data type 2–18

r-code
  size 2–34

Read-only (-RO) startup parameter 4–13

RECID function 2–23, 2–29
  joins 2–59
  replacing with ROWID 2–30
  transactions 2–23

Record creation 2–22

Record ID 5–22

Record locking 2–23

RELEASE statement 2–23

Remote databases
  starting a DataServer broker F–12

Remote DataServer
  starting 4–3

Repositioning indexes 2–9

REVERSE-FROM option 2–55

-RO startup parameter 4–13

ROWID data type 2–15

ROWID function 2–15, 2–29, 3–25
  joins 2–59

record creation 2–29
selecting support 5–17

RUN STORED-PROCEDURE statement 2–38, 2–40, 2–41, B–6
send-sql-statement option 2–12, 2–45

S

-S startup parameter 4–10
PROSHUT command F–14

SAVE CACHE statement 4–22

Schema 1–6

Schema Cache File (-cache) startup parameter 4–16

Schema caching 1–6, 4–22

Schema holders 1–5, 1–6
connecting 4–9
creating 3–18
deleting 5–20
deploying 3–27
logical names 5–13
modifying 5–14
updating 5–5
verifying 5–7
version 7 to 9 A–2
version 8 to 9 A–2

Schema objects
ORACLE 2–2

SELECT . . . FOR UPDATE (SQL) 2–24

send-sql-statement option 2–45
accessing views 2–12

Sequence generators 2–12

Sequences 5–23

Server Join (-nojoinbysqldb) startup parameter 2–54, 2–58, 4–19

Service Name (-S) startup parameter 4–10
PROSHUT F–14

SESSION:TIME-SOURCE handle 2–37

SETUSERID function 2–37

SHARE-LOCK option
FIELDS 2–37

SHLIB_PATH (HP-UX) environment variable 3–8, 3–9, 4–2, 4–7, C–2, E–9, E–10

Shutdown commands F–13

Single-user Mode (-1) startup parameter 4–13

Size limitations 5–23
column name 5–24
column width 5–24

Spaces
leading and trailing 2–7

.sql script
generating for ORACLE 5–20

SQL statements 2–24
4GL applications B–4
SELECT 2–24
sending to ORACLE 2–45

SQL*DBA 2–25, 3–20

SQL*Net 3–2, 3–3, 4–13, E–3

SQL*PLUS A–2

sqlnet connection string 4–14, 4–15

Starting a DataServer broker F–12

Startup parameters
Database Type (-dt) 4–15
DataServer (-Dsrv) 4–16
qt_cache_size 2–32
Index Cursors (-c) 4–16
Index Hint (-noindexhint) 2–7, 2–54, 4–19
Internal Code Page (-cpinternal) 2–5, 3–13, 4–12, 5–27
Logical Database Name (-ld) 4–15

network options
Host Name (-H) 4–10
Network Type (-N) 4–10
Service Name (-S) 4–10
optional 4–15
Password (-P) 4–12
Physical Database Name (-db) 4–12
query tuning 4–19
Read-only (-RO) 4–13
Schema Cache File (-cache) 4–16
Server Join (-nojoinbysqldb) 2–54, 2–58, 4–19
Single-user Mode (-1) 4–13
Stream Code Page (-cpstream) 3–13, 4–12, 5–27
User ID (-U) 4–13

Statements
CLOSE STORED-PROCEDURE 2–40, 2–41
COMPILE 2–33
CONNECT 4–9, 4–23
CREATE 2–22, 2–36
CURRENT-VALUE 2–36
DEFINE BROWSE 2–31
FIND 2–6, 2–23, 2–36
FIND FIRST 2–33
FIND LAST 2–33
FIND PREV 2–33
FOR EACH 2–6
OPEN QUERY 2–23, 2–37
RUN STORED-PROCEDURE 2–12, 2–38, 2–41
SAVE CACHE 4–22
UPDATE 2–24
VALIDATE 2–23

Stored procedures
closing all 2–40
cursors 2–42
handle 2–40
identifying 2–40
NLS support 2–45
retrieving parameters 2–40, 2–41
retrieving return status 2–41, B–5
returning results 2–42, B–2
running 2–37, 2–40

Stream Code Page (-cpstream) startup parameter 3–13, 4–12, 5–27

Synonyms 2–14
distributed databases 2–15

Syntax notation xiv

System-owned objects 3–24

T

Table properties
modifying 5–14

Table validation
adding to schema holder 5–15

Terminology 2–2

TNS_ADMIN environment variable C–3

Transaction Processing option 2–23

Transactions 2–23, 2–25, 2–29

Triggers 2–12
adding to schema holder 5–15

Two-phase commit 2–25
distributed databases 2–26

Typographical conventions xiii

U

-U startup parameter 4–13

ubroker.properties file
hierarchy F–4, F–9

Unique indexes
blank spaces 5–27

UNIX environment variable E–4

Unknown value 2–19

? (unknown value) 2–19

UPDATE statement 2–24

Updating schema holders 5–5

Upgrading for RECID
  manually A–2

Upgrading to ORACLE8 A–6

Upgrading to the DataServer 1–18

USE-INDEX option 2–6, 2–7, 2–30

User ID (-U) startup parameter 4–13

User IDs 2–37, 3–19, 4–13, 4–24

USERID function 2–37, 4–14

USING option 2–35
  joins 2–59

V

VALIDATE statement 2–23

Validation
  expression 5–17

VARCHAR2 data type 2–16

Verifying schema holders 5–7

Views 2–10
  multi-table 2–11

W

WHERE clause
  literals 2–51
  ORACLE time option 2–37

Word indexing 1–1
  CONTAINS 2–35

Z

-znotrim startup parameter 2–7