Perl for Oracle DBAs

Perl is a very powerful tool for Oracle database administration, but too few DBAs realize how helpful Perl can be in managing, monitoring, and tuning Oracle databases. Whether you’re responsible for Oracle9i, Oracle8i, or earlier databases, you’ll find Perl an invaluable addition to your database administration arsenal.

You don’t need to be a Perl expert to use the excellent applications and scripts described in *Perl for Oracle DBAs*. The book explains what you need to know about Perl, provides a wealth of ready-to-use scripts developed especially for Oracle DBAs, and suggests many resources for further exploration. It covers:

- The Perl language—an introduction to Perl, its rich history and culture, and its extensive text processing and data transformation capabilities.
- The Perl/Oracle architecture—detailed information about Perl DBI, *DBD::Oracle*, the Oracle Call Interface (OCI), *Oracle::OCI*, *extproc_perl*, and *mod_perl*, the modules that allow Perl programs to communicate with Oracle databases.
- The Perl Database Administration (PDBA) Toolkit—a comprehensive suite of specialized scripts designed to help Oracle DBAs perform both routine and special-purpose tasks: monitoring the Oracle alert log and databases, creating and managing Oracle user accounts, maintaining indexes and extents, extracting DDL and data, troubleshooting and tuning database problems, and much more. The book also explains how you can extend the toolkit and solve your own database administration problems using Perl.

“Any tools that help the skilled and often demanding work of an Oracle DBA are to be welcomed—and this book is full of them. It describes not only many powerful utilities, but also the Open Source components from which they’re built. It will empower Oracle DBAs to customize and develop solutions for their unique requirements.”

—Tim Bunce, Author of the Perl DBI

“A wonderful book! Real-world problems that every DBA has struggled with, solved using a practical step-by-step approach, plus a pinch of humor :). DBAs will find the toolkit and utilities extremely valuable; they might even find that they write a few lines of Perl themselves.”

—Andrew Carr, Director, Developer Evangelism, Oracle Corporation

www.oreilly.com
Perl for Oracle DBAs
# Table of Contents

## Preface
ix

## Part I. Introducing Perl for Oracle

1. **Perl Meets Oracle**
   - What is Perl? 4
   - The Perl/Oracle Architecture 10
   - Perl for Oracle DBAs 15
   - For Further Information 24

2. **Installing Perl**
   - Installing Perl 28
   - Installing Perl DBI 38
   - Running Perl DBI 48
   - Installing Cygwin 51

## Part II. Extending Perl

3. **Perl GUI Extensions**
   - Perl/Tk 66
   - OraExplain 72
   - StatsView 74
   - Orac 82
   - DDL::Oracle 84
   - SchemaDiff 90
   - Senora 92
<table>
<thead>
<tr>
<th>4. Perl Web Extensions</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>111</td>
</tr>
<tr>
<td>Oracletool</td>
<td>118</td>
</tr>
<tr>
<td>Karma</td>
<td>127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Embedding Perl into Apache with mod_perl</th>
<th>136</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_perl</td>
<td>137</td>
</tr>
<tr>
<td>Apache::OWA</td>
<td>155</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. Embedded Perl Web Scripting</th>
<th>161</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embperl</td>
<td>162</td>
</tr>
<tr>
<td>Mason</td>
<td>169</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. Invoking the Oracle Call Interface with Oracle::OCI</th>
<th>178</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Oracle::OCI?</td>
<td>179</td>
</tr>
<tr>
<td>What Is OCI?</td>
<td>180</td>
</tr>
<tr>
<td>Installing Oracle::OCI</td>
<td>185</td>
</tr>
<tr>
<td>Coding with Oracle::OCI</td>
<td>190</td>
</tr>
<tr>
<td>The Future of Oracle::OCI</td>
<td>198</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. Embedding Perl into PL/SQL</th>
<th>201</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Between Perl and PL/SQL</td>
<td>202</td>
</tr>
<tr>
<td>Embedding Perl Within Oracle</td>
<td>204</td>
</tr>
</tbody>
</table>

---

**Part III. The Perl DBA Toolkit**

<table>
<thead>
<tr>
<th>9. Installing the PDBA Toolkit</th>
<th>227</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing the PDBA Toolkit</td>
<td>228</td>
</tr>
<tr>
<td>Toolkit Modules</td>
<td>232</td>
</tr>
<tr>
<td>Installing the PDBA Toolkit for Unix</td>
<td>257</td>
</tr>
<tr>
<td>Installing the PDBA Toolkit for Win32</td>
<td>261</td>
</tr>
<tr>
<td>Configuring the PDBA Toolkit</td>
<td>265</td>
</tr>
</tbody>
</table>
10. Performing Routine DBA Tasks with the PDBA Toolkit .......................... 279
   Managing User Accounts .......................... 280
   Maintaining Indexes .......................... 297
   Killing Sniped Sessions .......................... 305
   Managing Extent Usage .......................... 312
   Extracting DDL and Data .......................... 315

11. Monitoring the Database with the PDBA Toolkit .......................... 328
   Monitoring the Alert Log .......................... 328
   Monitoring the Databases .......................... 342

12. Building a Database Repository with the PDBA Toolkit .......................... 366
   Repository Table Structure .......................... 367
   Installing the Repository .......................... 369
   Loading the Repository with Data .......................... 374
   Reporting on Database Changes .......................... 377
   Reporting on SQL Execution Plans .......................... 392

13. Extending the PDBA Toolkit ......................................................... 403
   Modifying a Script in the Toolkit .......................... 404
   Modifying a Module in the Toolkit .......................... 421

Part IV. Appendixes

A. The Essential Guide to Perl ......................................................... 439
B. The Essential Guide to Perl DBI ......................................................... 459
C. The Essential Guide to Regular Expressions ......................................................... 482
D. The Essential Guide to Perl Data Munging ......................................................... 521

Index ......................................................... 575
There are many books on Perl and many books on Oracle, but until now there have been no books dedicated to describing the relationship between these two popular technologies. Our aim is to bridge the gap between the world’s leading data-processing language and the world’s leading database. The Perl language was created with the goal of making “the easy things easy and the hard things possible.” Oracle’s ever-expanding purpose is to provide a complete database environment for the entire interconnected world. This mission makes Oracle a tough taskmaster—and thus an environment ideally suited to Perl, because being an Oracle database administrator is one of the toughest jobs around.

Oracle DBAs need enormous intelligence, infinite patience, and considerable courage. We think they also need Perl. The Perl open source language is a many-splendored thing; you can write scripts with it, develop GUIs with it, create web sites with it, generate XML with it—and you can probably hang your towels from it! Perl fills data warehouses and runs on virtually every operating system around. Perl is the toolkit without limitations, the salvation of your 24x7 lifestyle. If Gandalf the Wizard were to choose a scripting language, he would choose Perl.

Our mission in this book is to show you how Perl can revolutionize your life as an Oracle DBA. We’ll focus on four aspects of the Perl/Oracle connection:

*The Perl language itself*

We’ll introduce you to the Perl language, with its rich history and culture, present some language basics, and shine some light on CPAN, the Comprehensive Perl Archive Network, the main distribution point for Perl modules.

*The Perl/Oracle architecture*

We’ll introduce you to the modules that allow Perl programs to communicate with Oracle databases.

*Perl applications for Oracle DBAs*

We’ll profile about a dozen of the best ready-made applications written in Perl for use by Oracle DBAs. These provide help with database administration,
monitoring, tuning, and daily troubleshooting. They also provide components you can use in your own Perl scripts, should you choose to add a little program development to your daily DBA routine.

The PDBA Toolkit
We’ll present, for your enjoyment, an Oracle database administration toolkit we’ve written ourselves. The Perl Database Administration (PDBA) Toolkit contains nearly 100 Perl scripts and reusable modules that perform operations ranging from creating new Oracle users to monitoring the Oracle alert log to building a repository of database information for use in tuning and troubleshooting. All of this code is available on the O’Reilly web site (http://www.oreilly.com/catalog/oracleperl/pdbatoolkit).

Audience for This Book
If you’re an Oracle DBA who is trying desperately to keep up with the daily demands of administering, monitoring, and tuning your Oracle databases, this book is for you. We are Oracle DBAs ourselves, and we know how difficult your job can be. This book provides both information and software that we hope will ease your burden.

Although the primary audience is Oracle DBAs, many DBAs end up being developers from time to time, and there is no better language than Perl for writing those quick scripts. Anyone doing Perl development will find Chapters 7, 8, and 13, as well as the appendixes, particularly useful.

This book assumes no prior experience with Perl, though some knowledge of the language will help you get the most out of the material presented here. Although the book’s appendixes explore the essential syntax of Perl, Perl DBI (DataBase Interface), Perl’s regular expressions, and Perl’s data-munging modules, a complete Perl tutorial is beyond our scope. Our goal here is to jump-start your explorations into the intersection where Perl meets Oracle. We’ll provide plenty of suggestions for where to go next on your journey.

Which Platform and Version?
Both Perl and Oracle run on virtually every hardware platform and operating system. To demonstrate this ubiquity, we’ve used a wide variety of OS platforms and Oracle versions in the preparation of this book. Oracle versions range from Oracle7.3 through Oracle9i. OS platforms include Linux Red Hat 6, Linux SuSE 7.3, Solaris 8, Windows NT 4, Windows 2000, and others. We’ve focused on Unix and Win32 operating systems, but we’ve also included specific installation instructions for particular operating system variants when necessary.

Against this irresistible surge of platforms, our immovable rock is the Perl version we’ve used on all of these operating systems. Perl 5.6.1 was the latest stable Perl
release available as we wrote this book and developed the toolkit software. We also used the most current stable version of Perl DBI, Version 1.20, in conjunction with Perl DBI’s Oracle-specific driver module, DBD::Oracle (DataBase Driver for Oracle), Version 1.12.

By the time you read this book, it’s possible that the latest stable versions on the CPAN web site will have been upgraded, particularly if Perl itself is upgraded to Perl6, which was under development as we wrote this book. We’ll be updating our toolkit as an open source project in order to cope with any such Perl enhancements.

Structure of This Book

This book is divided into four parts:

Part I, Introducing Perl for Oracle

• Chapter 1, Perl Meets Oracle, introduces the Perl language and explains why it is such a helpful language for Oracle database administrators. It also provides an overview of the main components of the Perl/Oracle architecture.

• Chapter 2, Installing Perl, describes how to install Perl on Unix and Win32 systems. It also describes how to install Cygwin, a Unix-like development environment you can install on your Win32 machine.

Part II, Extending Perl

• Chapter 3, Perl GUI Extensions, describes Perl/Tk, an extensive GUI-based toolkit for Perl, as well as a number of applications that provide Oracle DBAs with graphically oriented tools for performing database administration. These include OraExplain, StatsView, Orac, DDL::Oracle, SchemaDiff, Senora, DBD::Chart, SchemaView-Plus, and a variety of Perl GUI integrated development environments (IDEs) and debuggers.

• Chapter 4, , discusses the relationship between Perl and the Apache web server, and focuses on two Oracle applications that use a web browser as their user interface: Oracle tool and Karma.

• Chapter 5, Embedding Perl into Apache with mod_perl, explains how the use of Apache’s mod_perl module can greatly improve the performance of Perl web-based CGI (Common Gateway Interface) scripts used with Oracle. This chapter also covers several related Apache modules: Apache::Registry, Apache::DBI, and Apache::OWA (used to connect mod_perl to Oracle’s PL/SQL Web Toolkit).

• Chapter 6, Embedded Perl Web Scripting, describes two applications, Embperl and Mason, that demonstrate the advantages of embedded scripting, a method that allows Perl code to be embedded within web pages. These tools provide a mechanism for filling web pages with dynamic Oracle data and
creating your own Oracle web tools, while separating content from design issues.

- Chapter 7, *Invoking the Oracle Call Interface with Oracle::OCI*, covers *Oracle::OCI*, a Perl module that provides a more extensive interface to Oracle’s Oracle Call Interface (OCI) than is possible with Perl DBI.

- Chapter 8, *Embedding Perl into PL/SQL*, discusses *extproc_perl*, a Perl module that communicates with the Oracle PL/SQL language’s external procedure C library system (known as EXTPROC). This module and the others described here allow Perl code to be embedded directly in PL/SQL programs.

### Part III, *The Perl DBA Toolkit*

- Chapter 9, *Installing the PDBA Toolkit*, introduces the components of the Perl Database Administration Toolkit (PDBA) and explains how to install it and build the toolkit’s password server.

- Chapter 10, *Performing Routine DBA Tasks with the PDBA Toolkit*, describes the toolkit’s Perl scripts that help DBAs perform day-to-day administration. We’ll cover managing user accounts, maintaining indexes, killing sniped sessions, managing extent usage, and extracting DDL (Data Definition Language) and data.

- Chapter 11, *Monitoring the Database with the PDBA Toolkit*, describes the toolkit’s Perl scripts that can be used to monitor both the Oracle alert log (containing database error and status messages) and the connectivity of the databases.

- Chapter 12, *Building a Database Repository with the PDBA Toolkit*, describes the toolkit’s Perl scripts that allow you to build a repository in which to store information about the many changes made to an Oracle database’s tables, indexes, roles, schemas, and other objects.

- Chapter 13, *Extending the PDBA Toolkit*, provides information that will be helpful if you decide to modify any of the scripts or modules in the toolkit. We’ll take a detailed look inside one of the toolkit’s scripts and modules and illustrate how you can change it to suit your specific database administration needs.

### Part IV, *Appendixes*


- Appendix B, *The Essential Guide to Perl DBI*, presents the main Perl DBI application programming interface (API) functions.


- Appendix D, *The Essential Guide to Perl Data Munging*, summarizes the Perl data-munging modules that are helpful in formatting and transforming data for
data warehouses and other such Oracle applications; it includes sections on numeric, date, conversion, and XML modules.

**About the Perl DBA Toolkit and Examples**

The full source code for the PDBA Toolkit is available on the O’Reilly web site at:

http://www.oreilly.com/catalog/oracleperl/pdbatoolkit

The toolkit is a fully open source-compliant project, and we welcome all contributions to extend it. In line with the OSI (Open Source Initiative) guidelines, the PDBA Toolkit is freely available for download over the Internet under the Perl Artistic License.† We’ll try our best to keep this code up to date as Perl and the many modules described in this book are upgraded. Our goal is to have you be able to download the latest and greatest version of the toolkit at all times as we seek constantly to improve it.

In addition to the toolkit programs, we have also provided a large number of stand-alone Perl programs in the book and on our site. We’ll also try to keep this code up to date and available for download at the O’Reilly web page cited earlier.

**Conventions Used in This Book**

The following typographical conventions are used in this book:

*Italic*

Used for filenames, directory names, and URLs. It is also used for emphasis and for the first use of a technical term.

*Constant width*

Used for code examples.

*Constant width bold*

Used occasionally in code examples to highlight statements being discussed.

*Indicates a tip, suggestion, or general note. For example, we’ll tell you if a certain feature is version-specific.*

---

* The Open Source Initiative (OSI) is a “non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community” (see http://www.opensource.org/).

† The Perl Artistic License “state(s) the conditions under which a package may be copied, such that the copyright holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the package in a more-or-less customary fashion, plus the right to make reasonable modifications” (see http://www.perl.com/pub/language/misc/Artistic.html).
Indicates a warning or caution. For example, we’ll tell you if a certain operation has some kind of negative impact on the system.

Comments and Questions

We have tested and verified the information in this book and in the source code to the best of our ability, but given the number of tools described in this book and the rapid pace of technological change, you may find that features have changed or that we have made mistakes. If so, please notify us by writing to:

O’Reilly & Associates
1005 Gravenstein Highway
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

You can also send messages electronically. To be put on the mailing list or request a catalog, send email to:

info@oreilly.com

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

We have a web site for this book where you can find updated links to Perl and Oracle software discussed in this book, along with errata (previously reported errors and corrections are available for public view there). You can access this page at:

http://www.oreilly.com/catalog/perlroracledba

To download the PDBA Toolkit, you can go directly to:

http://www.oreilly.com/catalog/oracleperl/pdbatoolkit

For more information about this book and others, see the O’Reilly web site:

http://www.oreilly.com

Acknowledgments

As you might expect, a tremendous number of people from the Perl and Oracle communities have helped us put this book together, including many of the creators of the actual tools discussed here. We cannot thank them enough (although we’ll do our best). We also are very grateful to the whole O’Reilly editorial and production team.

---

easy computing
From Andy

First of all I have to thank my wife and beg her not to throw me out of the house for having deserted our family (my son Ross, four-and-a-half, and daughter Ellie, two-and-a-bit) for the last six months while completing this *magnum opus*. I had thought my previous book, *Oracle & Open Source*, written with Sean Hull, was as tough as it would get on a family, but I was proved wrong. Daddy could often only be located by following the trail of pizza crumbs and Dr. Pepper cans from the fridge to the darkest recesses of the house where he hid, tip-tapping away on a variety of workstations. But all will be redeemed if you get as much out of this book as I have, in its long-fingered probings into the darkest recesses of Perl for Oracle DBAs. My wife Sue has been the magnificent rock upon which I built my effort, and without her and our beautiful children, you can just take everything else and give it all away. I am forever in their debt.

As with all O'Reilly books, this has also been an immense collaborative effort involving more than just the writing team of myself, Jared, and our omniscient editor, Debby Russell. I would like to thank Tim Bunce, the father of Perl DBI, for his help and support over the past five years, and everyone else who has helped us achieve our goal of producing this book, particularly our technical reviewers who did such a magnificent job under tight deadline pressure: Stephen Andert, Tim Bunce, Ben Evans, Lance Hollman, Thomas A. Lowery, Ilya Sterin, and Richard Sutherland.

Many others also helped us ensure that this book was both as accurate and as up-to-date as we could possibly make it. My deepest thanks to all of them: Dean Arnold, Jeffrey W. Baker, Doug Bloebaum, Ronald Bourret, Thomas Boutell, Hans-Bernhard Broeker, Alan Burlison, Damian Conway, Martin Drautzburg, Thomas Eibner, Kim Fowler, Andy Gillen, Lars Hecking, Russell Herbert, Roger Hipperson, Dan Horne, Jeff Horwitz, Sean Hull, Randy Kobes, Robert Lupton, Doug MacEachern, Edmund Mergl, Julian Moss, Alistair Orchard, Ian Pilgrim, Alan Ranger, Eric S. Raymond, Gerald Richter, Dave Rolsky, Dave Roth, Nick Semenov, Steve Shaw, Jonathan Swartz, Svante Sörmark, Jesse Reed Vincent, Adam vonNieda, and Ken Williams.

My future bar bill is now immense, possessing gravitational mass in its own right. However, before I finish, I have to thank our editor Debby Russell, who has done so much to support us and sculpt our natural techno-speak into what we hope you’ll find to be an invaluable guiding light towards the Perl and Oracle mithril of a deep subterranean world. She also brought this book down from being a ridiculous 1,000-page cave troll into the more-or-less manageable wood-elf you hold in your hands without losing a single important point. And finally, I have to thank David Gray, for his album, *White Ladder*, without which the completion of my half of this book would have been simply impossible, Dr. Pepper or no Dr. Pepper.
From Jared

I must first thank my own personal goddess, my wife, Carla. Although her husband spent many early mornings, late nights, weekends, and even several days of vacation time over a period of months, sequestered away in his hobbit hole of a computer warren producing code and text, she remained supportive and understanding.

Next I must thank my coauthor Andy. A true human dynamo, Andy was a whirlwind of activity while we collaborated on this project, and a source of inspiration on several occasions. He and Debby Russell are responsible for bringing our massive first draft down to a manageable size.

Andy and I both owe a deep debt of gratitude to everyone who has assisted us in the creation of this book. Andy has already mentioned those who have given us direct help in this project. I would like to reiterate our thanks to Tim Bunce for his outstanding work on the Perl DBI and \textit{DBD::Oracle} modules. I would take it even further and thank Larry Wall for the inspired moment when he first decided to create Perl.

Hats off to the entire open source community as well. Without the dedication and hard work of so many talented individuals, computing would be far less interesting. Trite? Maybe, but nonetheless sincere.

I’m grateful to my friends who listened patiently when they asked what “the book” was about when they learned I was working on one, even though they had never heard of Perl or Oracle, and for the fact they are still my friends, in spite of missed social occasions.

Finally, I thank my parents, Jerry and Betty. They had no idea that their combined genes would create progeny with a predilection for Perl, Unix, Linux, databases, and a fascination with ones and zeros arranged in meaningful patterns. Thanks Dad, thanks Mom.
This first part of the book introduces Perl and the architecture that allows it to connect to the Oracle database. It consists of the following chapters:

Chapter 1, *Perl Meets Oracle*, introduces the Perl language and explains why it is such a helpful language for Oracle database administrators. It also provides an overview of the main components of the Perl/Oracle architecture.

Chapter 2, *Installing Perl*, describes how to install Perl on Unix and Win32 systems. It also describes how to install Cygwin, a Unix-like development environment you can install on your Win32 machine.
Perl Meets Oracle

Perl is the world’s number one solution for transforming and gluing data together, and Oracle is the world’s number one solution for storing that data. In this book we’ll explore the interface between two of the finest American inventions since baseball and pretzels. We’re going to grab that Oracle data, we’re going to flip that Oracle data, and we’re going to munge that Oracle data. And we’re going to do it all in Perl!

The goal of this book is to explore the frontier connecting the Perl and Oracle worlds, having as much fun along the way as possible. There are many routes through this largely unexplored territory, and one we think is particularly important is the one focused on Oracle database administration. We are Oracle DBAs ourselves and we know the frustrations the job can bring. We’ve found Perl an enormous help to us in performing administrative tasks—both routine ones, like adding new users to the database, and more complex ones, like monitoring database connectivity in real time and tracking down database performance problems by comparing SQL execution plans. We want to share the information we’ve acquired over the years about Perl and its many Oracle applications. We also want to give you access to our own Oracle database administration scripts, which we’ve packaged up in the Perl Database Administration (PDBA) Toolkit described in this book and freely available on the O’Reilly web site.

This chapter sets the scene by introducing you to Perl and how it connects to Oracle. We’ll look at the following:

Perl’s origins and advantages
   We’ll take a look at where Perl came from and what makes it such a popular and powerful language.

Perl/Oracle architecture
   We’ll see how Perl connects to the Oracle database via the Perl DBI module, the DBD::Oracle program, and Oracle’s own OCI product. These modules interact to allow Perl programs access to Oracle databases.
Perl for Oracle DBAs

We’ll discuss why Perl is a particularly appropriate language for Oracle DBAs to learn and use.

We’ll also provide a list of additional Perl resources.

What is Perl?

Perl is a wonderful language with a rich history and culture. Many books have been written about its capabilities and roots. In this book we’ll be focusing on how Perl and Oracle work together, and we’ll only skim the surface of Perl’s overall capabilities, giving you just enough detail so you’ll appreciate what Perl can do for you.

In a nutshell, Perl is a freely available interpreted scripting language that combines the best capabilities of a variety of other languages. Despite borrowing other language capabilities, the whole of Perl is far greater than the sum of its parts. Perl was designed especially to be:

• Extremely fast, in order to be useful when scanning through large files
• Especially good at text handling, because data comes in many different forms and Perl has to handle them all
• Extensible, in order for Perl to expand users’ horizons, not restrict them

A tutorial for basic Perl is outside the scope of this book. Fortunately, there are many excellent web sites and books containing the information you need to get going. We’ve collected references to what we consider to be the best Perl books and online documentation in the “For Further Information” section at the end of this chapter. The appendixes provide quick references to different aspects of Perl’s capabilities. For online information, check out the main Perl portals at:

http://www.perl.com
http://www.perl.org
http://www.activestate.com (for Win32)

Before we get into the details of how Perl and Oracle interact, let’s take a step back to look at where Perl came from.

The Origins of Perl

Larry Wall created Perl back in 1987 with the goal of making “the easy things easy and the hard things possible”—originally just for himself, but ultimately for a whole generation of developers. Larry had been working on a complex system and had been trying to get Unix’s awk utility to do his bidding. He finally gave up on it and under the auspices of a secret project for the National Security Agency known as the “Blacker,” he decided to create a new utility by raiding a primeval soup of tech-
nologies and splicing together the genetic structures of awk, sed, sh and C, as well as csh, Pascal, and BASIC. The first release of Perl, Perl 1.0, arrived after a nine-month gestation period.

Perl was unlike any other computer language that had come before it, and this sea change was partially reflected in the name. The original name, “Pearl,” stood for “Practical Extraction And Report Language,” but in the spirit of this compact language, Larry wanted to save typing that extra fifth character. The name quickly morphed into Perl, which by now also stood for “Pathologically Eclectic Rubbish Lister.” This self-irreverence further distinguished the language and gave it a certain counterculture cachet.

Perhaps the most accurate summary of what Perl is best for can be found in the README file written by its author for Perl Version 1.0:

Perl is a interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It’s also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines (in the author’s opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of csh, Pascal, and even BASIC|PLUS.) Expression syntax corresponds quite closely to C expression syntax. If you have a problem that would ordinarily use sed or awk or sh, but it exceeds their capabilities or must run a little faster, and you don’t want to write the silly thing in C, then perl may be for you. There are also translators to turn your sed and awk scripts into perl scripts. OK, enough hype.

The Unix world embraced the Perl language, and the fast-growing Perl development community gradually built their favorite language into the world’s supreme text-processing engine. Over the next few years, Perl grew ever more powerful. Perl’s regular expression handling was enhanced, the ability to handle binary files was added to the language, and the three main variable types were honed and sculpted. Soon the Perl Artistic License was adopted, and with the publication of the first edition of Programming Perl, the definitive guide to the language, the camel became the Perl trademark.*

Perl has become hugely popular, largely because of its extremely fast text processing and its ability to glue difficult things together with ease. With the explosion of the interactive Internet in the 1990s, Perl found itself superbly pre-adapted to become the new tool of an Internet generation. It glued those trillions of text packets into one big global village! And as the World Wide Web burst on the scene, Perl continued to evolve, emerging as the premier language for developing web applications. Perl 4 brought the release of modules allowing Perl to interact with Oracle (and other)

* The camel is a great image that suggests it was designed by more than one voice—perhaps a bit challenged in looks, but perfectly adapted for a difficult ecological niche.
The current version of Perl, Perl 5, contains long-sought object-oriented features.

Perl on Win32

Although Perl’s origins were in the Unix world, it was ported to Windows back in 1995 by Dick Hardt and Hip Communications, the forerunners of ActiveState. Windows NT administrators then discovered a whole new world of functionality via the Win32 modules supplied by ActiveState, and Perl became their dominant scripting language. Perl was a lifesaver for busy administrators performing large NT system updates. (Adding 100 users to a system via the repetitive and arthritic point-and-click method really is no fun!)

Win32 Perl became so popular that there was some danger that the Unix and Windows versions would diverge. But Larry Wall was not about to let this happen. Those not familiar with Perl may wonder why it matters. What difference would it make if the Unix and Win32 Perls were different? In fact, it is this hard-won unity that gives Perl its power. You can write a single script on one operating system, and as long as you don’t use native methods, you can run it unchanged on every other kind of machine, from Linux to Windows NT to Solaris and back again. That is a huge advantage in our multiplatform, networked computing world.

CPAN (the Comprehensive Perl Archive Network)

Over the years, an enthusiastic and partisan army of Perl volunteers has extended Perl in a myriad of ways. CPAN (the Comprehensive Perl Archive Network), an online repository of Perl core files, documentation, and contributed modules, has become a model for an open source development community. Check out:

http://www.cpan.org

Literally thousands of Perl modules are now available on CPAN, providing virtually any application you can imagine—and many you haven’t yet imagined. Just about every Perl module we describe in this book, from core modules like Perl and Perl DBI themselves to Oracle-specific database administration scripts like OraExplain and Orac, can be downloaded from CPAN.

New Perl modules go through an evolutionary process that begins with an individual developer’s code, which he or she posts to CPAN. As others learn about the new module and start downloading and testing it, and relying upon it, it becomes more and more acceptable. If it’s good enough, and if enough people and products rely upon it, the Perl gods ultimately might decide to include the new module in the next general Perl distribution.
Perl and the corporate world

When Java, Microsoft’s Active Server Pages (ASP), and similar corporate tools came along, many people assumed that they would sweep the inelegant Perl away. However, this hasn’t come to pass. Instead, Perl has grown exponentially both in market share and stature, especially since its 1994 Perl 5 adoption of reference technology, which greatly increased its scope in terms of both extensibility and object orientation. Tim Bunce’s Perl DBI module, built on the object-oriented base, gave Perl the ability to interface with Oracle and other databases. The fact that Perl can now dynamically glue the Internet to the database has greatly increased corporate acceptance of the language.

The Perl Advantage

There are nearly as many reasons why people choose to use Perl as there are people who use Perl. Aside from the language’s specific capabilities, we think there are a few key reasons for Perl’s awesome acceptance among programmers and nonprogrammers alike:

Practicality

Unlike some languages that have developed within the ivory towers of computer science departments, Perl is a practical language. It is unbound by dogma and driven by day-to-day practicalities. With its flexible syntax, it gives users enormous freedom to do what they want to do.

Bandwidth

Perl is one of the most concise languages around. In ten lines of Perl code, you can achieve more than is possible in any other language. Disciplined use of Perl can thus reduce program maintenance costs (because there’s less to maintain) and aid clarity (because there’s less code to try to understand).

Range

Literally thousands of Perl modules are available for download from CPAN, covering virtually every computing requirement imaginable. The abundance of prebuilt code modules makes Perl the number one choice for anyone with a wide range of programming needs—and that description fits most Oracle DBAs.

We believe that Perl’s popularity is based to a large extent on the fact that it has resisted the temptation to try to become the most elegant language of its time. A linguist by training, Larry Wall took many lessons from the development of real-world natural human languages, and blended the necessary messiness of those languages into his evolving design for Perl. In the following sections we’ll look at how the English language itself offers some important Perl analogies.
Flat learning curve

Although natural languages such as English are difficult and messy, even a baby can learn them. The messiness of such languages aids learning, develops expression, and allows the human mind to map complex real-world problems onto the symbolic logic of complex real-world languages. Perl tries to follow this pattern—it’s very intentionally designed for humans rather than computers. You need only a little Perl to get going, just as a baby needs only a little language to ask for a chocolate ice cream. Indeed most of the fun of Perl is that you never stop learning about its new elements. This characteristic of Perl contrasts with some other languages where you have to learn virtually the entire shooting match before you can do the simplest thing, such as print:

"Hello World! :-)

It also means that it’s okay to know only parts of the whole language—every Perl programmer is on the same flat learning curve as every other Perl programmer, merely at a different position.

Expression

Perl is optimized for expressive power, rather than ease of operation. Once you’ve learned an element of Perl, such as the structure of hashes (described in Appendix A,
There’s more than one way to do it (TMTOWTDI)

In many computer languages, there’s often a single acceptable way to do a certain thing—for example, communicate with a distant server. Perl is different. So is English. In real life, when you introduce yourself to other people, there are many different ways to successfully perform this occasionally tricky verbal task. It’s the same in Perl. What counts is what works best for you, not some rigid adherence to a strictly enforced protocol. As with formal introductions, of course, there are certain conventions that most people use. There is peer pressure even among Perl programmers. But Perl itself doesn’t care; if you want to do something different, you are free to do so.

Flexibility

English is a successful language mainly because it looks forward into the future, rather than backward towards its origins. It’s built up from Latin, Greek, French, Anglo-Saxon, and many other elements. And if it needs to borrow the word “veranda” from the Portuguese in order to describe a covered porch, it just goes right ahead without worrying about whether doing so breaks some rule. Perl is the same: if it sees a great idea in Java, it just goes right ahead and borrows it, slipping it in so the join is invisible. Eventually, if it’s a successful graft, even Java programmers may come to think that the idea originally came from Perl. It is this continuous evolution that transforms Perl from the ordinary into the extraordinary.

Ambiguity

English is also successful because it’s so good at handling ambiguity. Although there are few cases, genders, or definitive word endings in the English language, local ambiguities are quickly resolved by the juxtaposition of certain other words, conventions, and punctuation. Perl is the same: some pieces of isolated code can be quite ambiguous, but the ambiguity is quickly resolved in the context of its word order, punctuation, and relationship to other code fragments. There are even pronouns in Perl, such as $_ and @_ for “it” and “they”!

Acceptance of the real world

In a pure computer language world, you could visit the local cinema in an infinite number of ways; for example, you could float up to 10,000 feet, disappear, and then rematerialize in your favorite seat to watch *The Lord of the Rings*. But the fact is that you’d most often walk or drive there. Similarly, Perl recognizes that most people tend
to want to do things in familiar ways (e.g., opening a file, processing the lines in it, and then closing the file). So Perl will typically assume that you’ll be following a natural order unless you tell it explicitly that you won’t be.

**Simplicity**

Lawyers have taken the once straightforward English language and twisted it into the most tortuous logic the human mind could devise—unfortunately, this is the route most often taken by other computer languages. They start simply enough, but develop a rigid straitjacket of theoretical perfection before drowning in a bog of complexity. You’ll be pleased to hear that Perl is much friendlier. There is no ideology that must be obeyed. A country run by Perl programmers would be a really cool place to live!

**Cooperation and divergence**

Natural languages have evolved with the involvement of different people over a long period of time—indeed, they continue to evolve. They’re also continuously diverging into separate dialects and even other languages. Perl too began as an amalgam of different ideas, shepherded together by Larry Wall. It has since continued as a cooperative effort, with many contributing voices. The eventual creation of Perl 6 will be one vast community effort (something we hope you’ll be part of).

But language fragmentation has been an ongoing problem for Perl. The solution has been a continuous release program over the last decade that has accommodated divergent tendencies. The CPAN architecture also offers an outlet for those with independent voices. The threatened Win32 divergence we discussed earlier in the “Perl on Win32” section could have had a dramatic impact on the unity of Perl—and all that implies in terms of portability and extensibility. Thankfully, as we described earlier, that threat came to a happy conclusion. And it’s still true that if you write a Perl script on one operating system, then as long as you haven’t used native methods and system commands, the script can be copied to any other machine and will work there identically, regardless of operating system.

**The Perl/Oracle Architecture**

How do Oracle DBAs, developers, and users take advantage of everything that Perl has to offer? The architecture illustrated in the figures in the following sections show how the various Perl and Oracle modules fit together to make the Perl/Oracle connection clean and efficient. In the following sections we’ll take a look at the main components of this architecture:

- Perl DBI
- `DBD::Oracle`
- OCI
Perl DBI and DBD::Oracle are Perl modules available from CPAN. OCI is an Oracle Corporation product that comes with all versions of the Oracle database.

**Perl DBI and DBD::Oracle**

Perl DBI is a generic application programming interface (API). It is similar in concept to ODBC (Oracle DataBase Connectivity) and JDBC (Java DataBase Connectivity), but it has a Perl-based object-oriented architecture. Perl DBI’s object-oriented architecture allows it to have a single routing point to many different databases (shown in Figure 1-1), each via a database-specific driver. Oracle uses the DBD::Oracle driver, another Perl module that provides the actual communication to the low-level OCI code. It is OCI that makes the final connection to the Oracle database.

![Figure 1-1. Perl DBI can interface to many databases](image)

The beauty of Perl DBI is you can forget the details of the necessary connections beneath its simple API calls. The DBI package glides serenely over the surface of our databases, while the driver module, DBD::Oracle, does all the hard paddling beneath the surface.

Figure 1-2 shows how all the modules fit together on the Perl and Oracle sides.

**The origins of Perl DBI**

The origins of Perl DBI date back more than a decade. Way back in 1991, an Oracle DBA, Kevin Stock, created a database connection program called OraPerl that was released for Perl 4. Over time, similar Perl 4 programs appeared, such as Michael Peppler’s Sybperl, designed for communication with the Sybase database. In a parallel development, starting in early September 1992, a Perl-based group was working on a specification for DBP4, a database-independent specification for Perl 4.
Within two years they were just ready to start implementing DBPerl when Larry Wall started releasing the alpha version of the object-oriented Perl 5. Taking advantage of both Perl 5 and the earlier Call Level Interface (CLI) work from the SQL Access Group, the DBPerl team relaid the foundations of Perl DBI within an object-oriented framework, creating this new architecture in a similar form to that employed by the familiar API of ODBC. Meanwhile, Tim Bunce wrote an emulation layer for OraPerl Version 2.4 that let people easily move their legacy Perl 4 OraPerl scripts over to Perl 5 and Perl DBI.

With the new DBI architecture, you could now transparently employ just one Perl module to connect to every type of database, as long as you had the right driver. Fortunately for Oracle DBAs, Tim Bunce, the main creator of Perl DBI, is also the main creator of DBD::Oracle, which automatically keeps Oracle on the cutting edge of Perl DBI’s development schedule.

**The Perl DBI API**

We won’t try to describe all of the capabilities of Perl DBI here, but Table 1-1 provides a summary of the main calls (e.g., DBI class methods) to OCI. For additional background information about Perl DBI, see Appendix B, *The Essential Guide to Perl DBI*. And for much more information, consult the references listed under “Further Information on Perl DBI” at the end of this chapter.

**Table 1-1. Main Perl DBI functions**

<table>
<thead>
<tr>
<th>DBI function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>available_drivers()</td>
<td>Lists all of the available DBD drivers including DBD::Oracle</td>
</tr>
<tr>
<td>data_sources()</td>
<td>Lists all of the databases available to DBD::Oracle</td>
</tr>
<tr>
<td>connect()</td>
<td>Establishes an Oracle database connection</td>
</tr>
</tbody>
</table>
The Oracle Call Interface

As we’ve said, Oracle Corporation’s Oracle Call Interface (OCI) is the component in the Perl/Oracle architecture that makes the final connection to the Oracle database servers. This C-based API provides a comprehensive library used to connect into Oracle from the external world. Use of OCI lets your Perl programs take advantage of the following OCI capabilities:

- High performance
- Security features, including user authentication
- Scalability
- N-tiered authentication
- Full and dynamic access to Oracle objects
- User session handles
- Multi-threaded capabilities
- Support for accessing special Oracle datatypes such as LOBs (large objects)
- Transactions
- Dynamic connection and session management
- Asynchronous event notification
- Access to other databases
- Full character set support

**Table 1-1. Main Perl DBI functions (continued)**

<table>
<thead>
<tr>
<th>DBI function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>disconnect()</code></td>
<td>Disconnects a login session from Oracle</td>
</tr>
<tr>
<td><code>err()</code></td>
<td>Returns the relevant Oracle error code</td>
</tr>
<tr>
<td><code>errstr()</code></td>
<td>Supplies an associated Oracle error message</td>
</tr>
<tr>
<td><code>prepare()</code></td>
<td>Prepares a SQL statement for execution</td>
</tr>
<tr>
<td><code>execute()</code></td>
<td>Executes a prepared statement</td>
</tr>
<tr>
<td><code>do()</code></td>
<td>Prepares and executes a single SQL statement all together</td>
</tr>
<tr>
<td><code>bind_param()</code></td>
<td>Binds a value to a prepared statement</td>
</tr>
<tr>
<td><code>commit()</code></td>
<td>Commits a transaction</td>
</tr>
<tr>
<td><code>rollback()</code></td>
<td>Rolls back a transaction</td>
</tr>
<tr>
<td><code>table_info()</code></td>
<td>Fetches metadata information from a table</td>
</tr>
<tr>
<td><code>fetchrow_arrayref()</code></td>
<td>Fetches a row of data into a referenced array</td>
</tr>
<tr>
<td><code>fetchrow_array()</code></td>
<td>Fetches a row of data into an array</td>
</tr>
<tr>
<td><code>selectrow_array()</code></td>
<td>Executes <code>prepare()</code>, <code>execute()</code> and <code>fetchrow_array()</code> all in one call</td>
</tr>
</tbody>
</table>
About the Authors

Andy Duncan is the coauthor of Oracle & Open Source (O’Reilly, 2001), as well as Perl for Oracle DBAs (O’Reilly, 2002). The first book arose after Andy’s creation, in 1998, of the Orac Perl/Tk tool for Oracle DBAs. Since then, he has worked mainly as an independent development and DBA consultant, and has counted both Oracle Corporation and Sun Microsystems among his long-term clients. In addition to performing Oracle, Perl, and Java consultancy work, Andy teaches as a senior instructor for Learning Tree International, covering both introductory and advanced Perl courses. He lives in Oxfordshire, England, and can be reached via andy_j_duncan@yahoo.com.

Jared Still has been an Oracle DBA in health insurance and manufacturing environments since 1994 (and version 7.0.13 of Oracle). He first began dabbling with Perl in 1993 and it was love at first sight. Perl became an integral part of his Oracle toolkit when it was used to rapidly prototype and implement complex reporting based on data in Oracle databases. Jared has been working with databases of various ilks since 1988, and along the way has also worked as a Unix system administrator. When not riding herd on the databases at the OK Corral, he likes to spend time tinkering on his car, fly fishing, or sitting on the deck in the backyard at his home doing absolutely nothing. He can be reached at jkstill@cybcon.com.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animals on the cover of Perl for Oracle DBAs are thread-winged lacewings. Lacewings can be found all over the world, primarily in warmer climates. They live mostly in sheltered, sandy areas such as wooded dunes, forest floors, and river-banks, until they reach adult form, at which time their wings enable them to roam more freely.

In their larvae state, lacewings prey voraciously on such unsuspecting victims as aphids, mites, and scale insects. They hide under pieces of wood or debris, wait for insects to pass, then attack with their pincer-like mandibles.

Lacewings undergo full metamorphosis throughout their lives. The adult form is characterized by two sets of wings, a long, slender abdomen, and clubbed antennae. Lacewings are one type of many nerve-winged insects because of the intricate pattern of lines (nerves) running through their transparent wings.

Darren Kelly was the production editor for Perl for Oracle DBAs. Nancy Crumpton provided production services and wrote the index. Jan Fehler was the copyeditor. Tatiana Apandi Diaz and Claire Cloutier provided quality control.
Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from *The Riverside Natural History: Volume 2*. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe’s ITC Garamond font.

David Futato designed the interior layout. This book was converted to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont’s TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Linley Dolby.