THE EXPERT'S VOICE® IN PEOPLESOFT

OakTable PRESS

PeopleSoft for the Oracle DBA

A PeopleSoft Survival Guide for Oracle DBAs

David Kurtz

Foreword by Wolfgang Breitling

Apress®

PeopleSoft for the Oracle DBA

DAVID KURTZ

Apress®

PeopleSoft for the Oracle DBA

ISBN (pbk): 1-59059-422-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Tony Davis
Technical Reviewers: Wolfgang Breitling, Tim Gorman
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, John Franklin, Jason Gilmore, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editors: Andy Carroll and Nicole LeClerc
Production Manager: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Kinetic Publishing Services, LLC
Proofreader: Elizabeth Berry
Indexer: John Collin
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, e-mail `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

Contents at a Glance

Contents

Foreword

This book should have been written years ago, but then it would not have been as comprehensive as it is today. It bridges the gap between the worlds of PeopleSoft and Oracle, explaining where and how the two sides meet.

My own experience with PeopleSoft began in 1992 as the DBA on a GL 1.1/PeopleTools 2.1 implementation project. The DBA is often the only "technology-savvy" person on such a project, with the rest of the team consisting of functional experts, developers and, of course, management. As such, the DBA often is tasked with helping to investigate and solve any technology-related problems, even if they have nothing to do with the database. In those early days, one of the big problems was to shoehorn all the pieces of the then client/server application into the first 640KB of a Windows 3.1 client. As a consultant on another implementation project, by now PeopleSoft 5.1, I learned firsthand the performance penalty of the client/server model in a wide area network (WAN) over frame relay, and the distinction between bandwidth and latency. The purchased bandwidth was sufficient for the demand, and that had been all the capacity planners were interested in. However, the accumulated latency of the frame relay network wreaked havoc with the performance. Of course, we were not alone in experiencing this, and in response PeopleSoft added the Tuxedo Application Server as a middle tier.

The application server is another piece of the growing technology stack in a PeopleSoft implementation that the DBA is tasked with administering simply because he or she is the most knowledgeable person around. Of course, with all the improvements Oracle continually makes to its software, databases these days manage themselves, and it is therefore high time to find something else for the DBA to do.

That was when I first encountered David's name. He was the author of a paper titled "Advanced Tuxedo" that went a long way toward improving my knowledge of the application server and my ability to fine-tune its configuration and administration. Although I've yet to meet David in person, I have, since then, gotten to know him through his work, expressed in papers or simply in contributions in newsgroups. So, when I received an e-mail asking me if I would consider being a technical reviewer for the book he was writing, I was honored—and eager, since I knew it would be a great book. Although the book is titled PeopleSoft for the Oracle DBA, I believe that much of what David explains about how PeopleSoft works with Oracle is also applicable to any other database back-end. Therefore, DBAs with PeopleSoft on DB2 or Microsoft SQL Server, for example, will get many of the same benefits from reading this book.

As I said at the beginning of this foreword, this book bridges the gap between the PeopleSoft and the database worlds. For that reason, I am convinced that PeopleSoft administrators and developers also can learn a lot from what David lays out in this book and discover how some of the decisions they make affect what is happening at the database end. While readers who are new to PeopleSoft will gain the most from the material David presents, even seasoned users will benefit, particularly since the book is up-to-date with PeopleTools 8.44. What I like most about the book is that it is not a dry rehash of PeopleSoft manuals; rather, it is filled with solid advice and helpful scripts from David's many years of experience.

—Wolfgang Breitling

About the Author

DAVID KURTZ studied physics at the University College London. He has worked with Oracle since 1989, spending six years as an Oracle developer and DBA working on assurance, insurance, and actuarial software. In 1996 he joined PeopleSoft, starting out in support and moving into consultancy. In 2000 he set up Go-Faster Consultancy Ltd. (www.go-faster.co.uk), an organization that provides specialist performance and technical consultancy to PeopleSoft users.

A member of the UK Oracle User Group since 1994, David became chairman of the Unix SIG in 2000. He presents regularly at PeopleSoft and Oracle user group conferences and SIG meetings. He is also a member of the OakTable network (www.oaktable.net).

About the Technical Reviewer

WOLFGANG BREITLING was born in Stuttgart, Germany, and studied mathematics, physics, and computer sciences at the University of Stuttgart. He joined IBM Germany in 1974 in the development laboratory, where he worked in the QA department. One of his tasks was to co-develop an operating system to test the hardware of the /370 model machines developed in Boeblingen, Germany, and Poughkeepsie and Endicott, New York.

His first direct foray into performance-related tasks was a program to test the speed of individual operating codes. After IBM Germany, he worked as a systems programmer on IBM's hierarchical databases DL/1 and IMS for a company in Switzerland before emigrating to his current home in Calgary, Canada. After several years as systems programmer for IMS and then DB2 on IBM mainframes, in 1992 he got involved in the project to implement PeopleSoft GL. In 1996, he became an independent consultant specializing in administering and tuning PeopleSoft, particularly on Oracle. Since then he has been involved with several PeopleSoft installation and upgrade projects. The particular challenges in tuning PeopleSoft caused him to delve into the Oracle cost-based optimizer in an effort to better understand how it works and to use that knowledge in tuning. He has shared his findings in papers and presentations at IOUG, local Oracle user groups, and other conferences dedicated to Oracle performance topics.

Wolfgang has been married to his wife, Beatrice, for over 30 years, and has two children, Magnus and Leonie.

Acknowledgments

You don't get to put your name on the cover of a book without a great deal of help from a lot of other people.

My thanks to Jonathan Lewis for introducing me to Mogens Nørgaard and the OakTable network. My thanks to Mogens for his hospitality, sage advice, whisky, and for founding the OakTable network and introducing me to Tony Davis. It is a privilege and an education to be associated with them. My thanks to Tony Davis of Apress for being convinced that this book was a viable idea and for guiding me through the process of producing it.

I want to acknowledge my technical reviewers, Wolfgang Brietling and Tim Gorman, for their many probing questions, which have led to days of research and experimentation, and for their insightful comments, many of which have been incorporated into the text. Their contribution has been huge.

My thanks also to the many companies using PeopleSoft with whom I have worked. The challenge of solving problems and optimizing PeopleSoft and Oracle technologies in real-world situations has led directly to this book.

Most important, I want to thank my wife, Angela, for her support and patience while I have spent even more time than usual in front of a computer.

Introduction

This book is aimed at helping Oracle DBAs understand and use PeopleSoft technology. For the typical DBA, the introduction to PeopleSoft is likely to include some surprises, not all of them agreeable. Many—if not most—DBAs have to deal with many different databases, usually supporting different applications. Often they will want to be able to administer all databases in a standard fashion. However, this is not always possible with a PeopleSoft system.

Most surprising to Oracle DBAs may be what is missing. In a vanilla PeopleSoft database, there is only minimal use of Oracle-specific features and Oracle-specific SQL constructions. There are no referential constraints. Very few optimizer hints are used, and only where there is no alternative. All PeopleSoft processes connect to the one database schema that contains all the database objects, so security is maintained by the application, not the database. Oracle sequences are never used; instead, sequence numbers are generated using ordinary tables.

In order to avoid the use of platform-specific SQL constructions, most of the delivered SQL conforms to a lowest common denominator subset of SQL accepted by the supported RDBMS platforms (Oracle, Microsoft SQL Server, DB2, Sybase, and Informix). The data model is kept uniform across all platforms, although there are variations in the column data types between platforms. There can be some differences in the indexing between platforms. There is some capability for different code on different platforms in PeopleSoft, but its use in the delivered product is kept to an absolute minimum.

In PeopleTools 8, there has been some expansion of the areas in which it is possible to introduce database-specific features and code. From PeopleTools 8.1, database triggers were used to write audit records, although these have to be enabled by customization. In PeopleTools 8.4, to support mobile agents, database triggers are also generated by the Application Designer to increment sequence numbers (held on tables not in Oracle sequences) and update timestamps when records are added. Constraints are used to validate the lengths of character columns in a Unicode database.

Hence, PeopleSoft is sometimes described as a *platform agnostic* product. It is my experience that this approach generally does not produce optimal performance. It may assist PeopleSoft to manufacture and maintain a single product on many platforms, but it does not help PeopleSoft customers to achieve optimal performance from their systems on their chosen database platform.

The other area of confusion for typical Oracle DBAs, particularly those familiar with Oracle's management tools, is that unless certain DBA tasks are incorporated into the application with PeopleSoft's Application Designer, they may be lost. Consequently, this can restrict the effectiveness of generic Oracle administration and monitoring tools.

In short, to be effective, DBAs must become PeopleSoft aware. They must work with the PeopleSoft development tools and the application, rather than continually fighting against it—otherwise it will bite back! One of the goals of this book is to outline areas of database administration that require special handling by DBAs and provide workaround techniques where possible.

Who Should Read This Book?

Though primarily aimed at the Oracle DBA who is responsible for maintaining PeopleSoft databases, this book can justifiably claim a wider audience. Much of the material is relevant to other database platforms.

The chapters dealing with the general PeopleSoft architecture and its evolution, and with Tuxedo and WebLogic, will also be of interest to PeopleSoft administrators. Also, it is not uncommon for Tuxedo and web server installation, administration, tuning, and troubleshooting to fall to the DBA for lack of other qualified resources.

What Does This Book Cover?

The following is a chapter-by-chapter breakdown summarizing some of the key topics that we will cover:

- **Chapter 1: An Overview**. This chapter presents a brief history of the evolution of PeopleSoft and its technology.
- **Chapter 2: BEA Tuxedo: PeopleSoft's Application Server Technology**. This chapter explains what Tuxedo is, how it works, and how PeopleSoft introduced Tuxedo into its product. Of all the people concerned with a system, the DBA is most likely to have the skills needed to assimilate this technology.
- **Chapter 3: Database Connectivity**. Nearly all the objects in a PeopleSoft database are in a single schema in an Oracle database. This chapter explains how a PeopleSoft database is structured, and how PeopleSoft processes securely authenticate the user and connect to the database.
- **Chapter 4: PeopleSoft Database Structure: A Tale of Two Data Dictionaries**. In order to deliver the same application to different database platforms, PeopleSoft maintains its own data dictionary, and then uses it to dynamically generate application SQL. This chapter examines the relationship between the PeopleSoft data dictionary and the Oracle database catalogue.
- **Chapter 5: Keys and Indexing**. This chapter describes how indexes are defined in the PeopleSoft Application Designer and how that definition is stored in the PeopleSoft data dictionary.
- **Chapter 6: PeopleSoft DDL**. This chapter shows how the PeopleSoft Application Designer generates DDL to build and analyze tables and indexes. It also explains to what extent the DBA can adjust that DDL to introduce Oracle-specific features, and when it is necessary to work outside the PeopleSoft design tools.
- **Chapter 7: Tablespaces**. This chapter discusses the tablespaces that are created when PeopleSoft is installed in an Oracle database. It also explains how to introduce some modern Oracle tablespace features.
- **Chapter 8: Locking, Transactions, and Concurrency**. This chapter explains how PeopleSoft maintains consistency of data, without holding database locks for long periods. It also shows how PeopleSoft creates sequences without using Oracle sequences.

- **Chapter 9: Performance Metrics**. This chapter explains the various sources of performance metrics in PeopleSoft and how to harvest them.
- **Chapter 10: PeopleSoft Performance Utilities**. This chapter describes the additional performance instrumentation that has been added in PeopleTools 8.4, including the Performance Monitor in 8.44, which provides a sophisticated wait interface.
- **Chapter 11: SQL Optimization Techniques in PeopleSoft**. This chapter describes how to enable Oracle's SQL trace on PeopleSoft processes and, once the DBA has identified SQL bottlenecks, how to apply tuning techniques through PeopleSoft development tools.
- **Chapter 12: Configuring the Application Server**. The application server has an intimate relationship with the database, which can affect database and system performance; therefore, the DBA needs to know how to configure the application server.
- **Chapter 13: Tuning the Application Server**. This chapter explains how to appropriately size the application server. It also covers other features that can affect system performance.
- **Chapter 14: The Process Scheduler**. This chapter describes how the PeopleSoft Process Scheduler is used to initiate batch and report processes. Regulating the batch load has implications for overall system performance.

Software Versions

The rate at which new versions of software appear can be bewildering and terrifying. On completion of work on this book, I am using the following software versions:

- PeopleTools 8.20, 8.43, and 8.44
- Tuxedo 6.5 and 8.1
- WebLogic 5.1 and 8.1
- Oracle 8.1.7.4 and 9.2.0.5

The good news is that PeopleSoft has built progressively on the structures established in previous versions, and many of the underlying principles have not changed.

Other Resources

This book does not seek to explain how to administer or tune an Oracle database. There are many excellent books and other sources on these subjects. Also, if read in isolation, this book will not tell you absolutely everything that a DBA needs to know about PeopleSoft. There are other resources that you should also make use of:

- **PeopleSoft Customer Connection** (www.peoplesoft.com): This is PeopleSoft's support website. It gives you access to product support, patches, and additional documentation. You will need an account and password to access this site.
- **PeopleBooks**: This is the documentation shipped with PeopleSoft's products.

- **PeopleSoft Red Papers**: These are technical documents available on the Customer Connection site that discuss how to optimally configure various aspects of PeopleSoft technology. This documentation is good at telling you what to do, but not always why you should do it.
- **BEA eSupport** (http://support.bea.com): You can sign up for free access to the BEA support resources.
- **BEA Tuxedo and WebLogic documentation** (http://e-docs.bea.com): Documentation for all versions of all BEA products is freely available on this website.
- **PeopleSoft DBA Forum** (http://groups.yahoo.com/group/psftdba/): This Yahoo group is where PeopleSoft DBAs and other interested technicians discuss ideas, ask questions, and share information.
- **Go-Faster Consultancy** (www.go-faster.co.uk): That's me!

Online Resources for This Book

You've read the book, now surf the website. This book has its own website, www.psftdba.com, which includes the scripts and code examples in the text, and any necessary corrections and additions.

I started the PeopleSoft DBA Forum (http://groups.yahoo.com/group/psftdba/) after a roundtable discussion group at the PeopleSoft EMEA User Conference in 2002. It is a moderated forum aimed at the needs of DBAs who administer PeopleSoft systems. It is therefore the perfect place to discuss the subject matter of this book and ask related questions.

Contacting the Author

From time to time in the course of this book, I express my opinions about various things. Those opinions are purely my own and are not necessarily the opinions of any other person or organization.

Despite every effort to the contrary, there is no guarantee that the content in this book is error-free. If you find any errors, please contact me via e-mail at info@go-faster.co.uk.

CHAPTER 1

An Overview

PeopleSoft makes packaged business-application software for larger companies. This chapter provides an introduction and overview of PeopleSoft, its technology, and its history. We'll take a very high-level look at some of the major parts that make up today's PeopleSoft systems, namely the database (in this case Oracle) that stores both the PeopleSoft application data and much of the application code, the Tuxedo Application Server, and the PeopleTools integrated development environment, which is used for most aspects of developing and administering PeopleSoft applications.

We'll then step through the overall architecture of a PeopleSoft system and see how it has evolved from the initial client/server architecture to the modern four-tier Internet architecture.

Finally, we'll take a look at what all this means to the database administrator (DBA) charged with maintaining a PeopleSoft application, and we'll consider the implications it has for the relationship between developers and DBAs on PeopleSoft systems. This introduction will help to put some of the following chapters into context.

What Is PeopleSoft?

Reuters' abridged business summary for PeopleSoft begins with this statement:

> *PeopleSoft, Inc. designs, develops, markets and supports enterprise application software products for use throughout large and medium-sized organizations worldwide. These organizations include corporations, educational institutions and national, state, provincial and local government agencies. The Company provides enterprise application software for customer relationship management, human capital management, financial management and supply chain management, each with a range of industry-specific features and functions.*[1]

In 2003 PeopleSoft acquired J.D. Edwards. The products that were formerly PeopleSoft are now referred to as "PeopleSoft Enterprise." The products that were formerly J.D. Edwards are now called "PeopleSoft EnterpriseOne" and "PeopleSoft World." This book is about PeopleSoft Enterprise software.

1. From http://finance.yahoo.com/q/pr?s=PSFT, August 2004.

There are currently eight Enterprise product lines that contain various modules:

- **Campus Solutions**: This product is designed for universities and other higher-education institutions.
- **Customer Relationship Management (CRM)**: PeopleSoft started to use Vantive CRM internally in the Global Support Center (GSC) in 1997. They liked it so much that they purchased the Vantive Corporation in October 1999 and sold the Vantive product alongside the PeopleSoft 7.5 products. Vantive CRM was rewritten to run under PeopleTools and was re-released as PeopleSoft CRM 8.
- **Financial Management**: This is a complete financial management and accounting package.
- **Human Capital Management (HCM)**: This product was referred to as Human Resource Management (HRMS) until release 8.1. Traditionally, it has been PeopleSoft's strongest product. There are various modules including Time and Labor, Benefits, and Student Administration. There is a North American local payroll, and there are other payroll interfaces for various countries. A separate Global Payroll (GP) module was released with version 8.1, and the list of Country Extensions for GP continues to grow.[2]
- **Service Automation**: This product provides self-service access to various modules. From PeopleTools 8.44, offline mobile clients are also available for some modules.
- **Supplier Relationship Management**: This product supports purchase and procurement processes.
- **Supply Chain Management**: This product supports business-to-business interaction along the supply chain.
- **Enterprise Tools and Technology**: This product contains all the PeopleSoft proprietary technology and development tools (often referred to simply as PeopleTools) that are used by PeopleSoft to develop its applications. The development tools are included in all of the other products so that companies can customize and extend the delivered products to match their own requirements. However, it is also possible to license PeopleTools separately to develop a system from scratch.

There is no unique master-detail relationship between product lines and modules. Some modules are included in more than one product line. For example, there are HRMS and Payroll modules from HCM, and there are also Receivables and General Ledger modules from Financials in Service Automation.

Components of a PeopleSoft System

PeopleSoft installations have evolved over the years into a combination of technologies that are used to develop and deliver PeopleSoft applications to desktop Internet (HTML) browsers. PeopleSoft calls this its "Pure Internet Architecture" (PIA).[3]

---

2. As of August 2004, there still is no Global Payroll Country Extension for North America. Payroll for North America, often just called Payroll, is a completely separate product.
3. In some PeopleSoft material, PIA is said to stand for "PeopleSoft Internet Architecture."

When we step through the overall architecture, you will see more clearly that PeopleSoft is a chain of linked technologies that stretch between the user and the database. The database is fundamental to the entire structure, but the technology stack stretches out through the BEA Tuxedo Application Server, and the Java servlet to the user's browser. PeopleSoft has also developed its own in-house development tools. All of this is collectively referred to as PeopleTools. Figure 1-1 shows, in a simplified fashion, how these pieces interact.

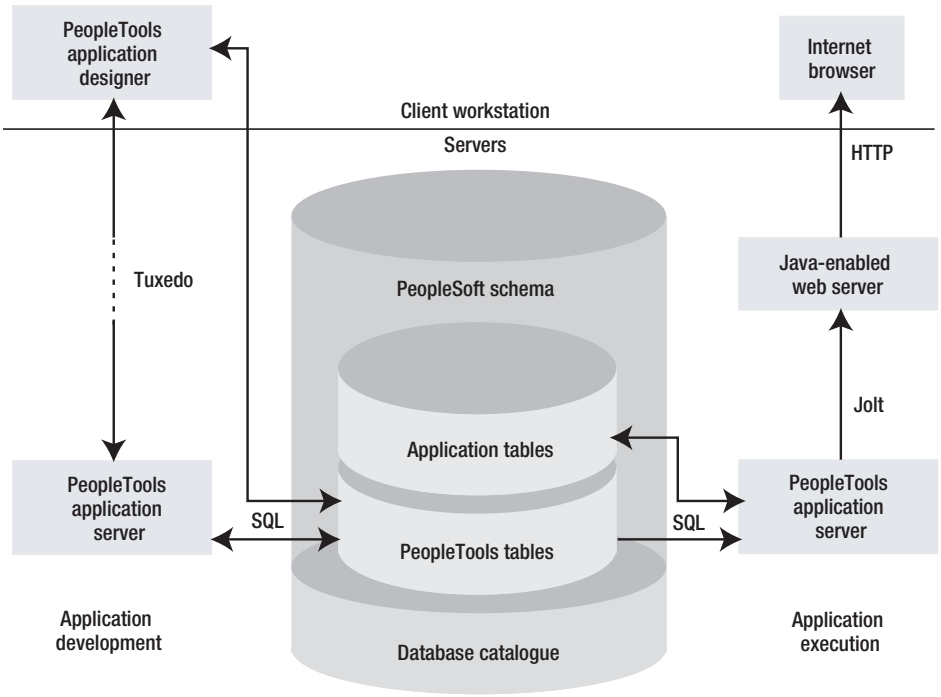


Figure 1-1. *A simplified version of the PeopleTools architecture*

The Database

From the beginning, PeopleSoft was designed to run on various database platforms. The majority of installations are on Oracle, but there are also many companies using IBM's DB2 and Microsoft SQL Server, and a few still use Sybase and Informix. Therefore, as far as possible, PeopleSoft delivers the same SQL code to all database platforms.

Both the PeopleSoft application and the application data are stored in the database:

- The database catalogue describes all the objects in the database.
- The PeopleTools tables describe the application data and contain the source code for much of the PeopleSoft application.
- The application tables contain the users' data.

The PeopleTools tables and the application tables all coexist in the same database schema.

When the Application Designer saves a development object, it writes to the PeopleTools tables. When the application is executed, that information is compiled into a cache format that is then interpreted by the application server. Upon execution, much of the application SQL is generated dynamically, based on the contents of the PeopleTools tables.

The Application Designer is also responsible for defining data structures in PeopleSoft applications and for building database objects, so it is inevitable that some of the PeopleTools tables that describe data structures will correspond closely to the Oracle database catalogue views. This means that in a PeopleSoft database we are dealing with two data dictionaries rather than one: the database's catalogue defines all of the objects that exist in the database, and the PeopleSoft dictionary defines all the objects that should exist in the PeopleSoft schema and that can therefore be referenced by the application. The interrelationship between these data dictionaries is discussed in detail in Chapter 3.

Tuxedo Application Server

PeopleSoft has selected various "best of breed" solutions for its system architecture, including BEA Tuxedo for their application server.

At run time, the application server interprets the application data in the PeopleTools tables, executes the business logic of the application, and even generates the HTML pages that are served up to the browser. The "publish and subscribe" technology that generates and accepts XML messages has also been incorporated into the application server.

The middleware and batch processes run on the major Unix platforms and Windows (from PeopleTools 8.44 onwards, Linux is also supported).[4] However, some components only run on Windows. Chapters 2 and 12 discuss the inner workings of the application server in more detail.

PeopleTools

All of the PeopleSoft products listed so far are developed using PeopleSoft's proprietary development tools, namely PeopleTools. The development tools are also available to PeopleSoft customers so that they can develop their own customizations and apply PeopleSoft upgrades and patches.

Since PeopleTools 7, many of the formerly separate development utilities have been consolidated into the Application Designer utility, and it is this PeopleTools utility that we'll encounter most often in this book. These are some of its functions:

- **Defining records**: The data model for the application is defined in the Application Designer. A record in PeopleSoft can correspond to a table or view on the database, or to a set of working storage variables (see Chapter 3). The indexes on the table are defined (Chapter 5). The Application Designer will also generate the data definition language (DDL) to build the database objects (see Chapter 6). Column definitions in the DDL will differ on different database platforms. Different databases also specify different physical attributes for objects.

4. As of August 2004, only Red Hat Advanced Server 2.1 (32-bit) is certified by PeopleSoft for all components in its architecture. Other versions of Red Hat Linux are certified, but only for use as a database server.

- **Creating PeopleCode**: PeopleCode is PeopleSoft's proprietary programming language. It is used to perform additional processing in the PIA and in some Application Engine batch processes. PeopleCode is specified on records in the Application Designer. In PeopleTools 8, the Application Designer can also be used as an interactive debugging tool in which you can step through the application including PeopleCode programs as they are executed.
- **Defining pages (formerly called panels)**: Originally panels were drawn in the graphical design tool, and they corresponded exactly with how they appeared in the Windows client. In PeopleTools 8, pages are developed in much the same way. Then, at run time, HTML pages are generated by the application server and are served up to the web browser. Thus, the developer can produce a web application without having to code any HTML or JavaScript, although additional HTML and Java can be included in a PeopleSoft application.
- **Defining menus**: The Application Designer also maintains the menu navigation for the application.
- **Upgrading**: The Application Designer is used to migrate sets of source code, called *projects*, between PeopleSoft systems. In version 8, projects can be exported to and imported from flat files. PeopleSoft also uses this mechanism during initial installation and to deliver patches.

PeopleSoft also delivers a number of other development tools and utilities:

- **Data Mover** is capable of exporting and importing data to and from a PeopleSoft database. It is used during installation to import all the objects and the data they contain into the database. PeopleSoft also uses Data Mover to deliver standing data to go with patches.[5] The same Data Mover export file can be imported into any database platform. Therefore, it can be used to migrate a PeopleSoft database from one platform to another, although it may not be fast enough for large databases. It is also capable of running some SQL scripts. It only connects in two-tier mode.
- The **Upgrade Assistant** was introduced in PeopleTools 8 to assist with the automation of PeopleSoft upgrade and patch processes.

Batch and report processing is provided by a number of technologies:

- **Application Engine** is PeopleSoft's proprietary batch processing utility. In PeopleTools 8 it was rewritten in C++ so that it could also execute PeopleSoft, and it is now developed in the Application Designer. This means that Application Engine programs can also be migrated by Application Designer.
- **SQR** (licensed by PeopleSoft from Hyperion Solutions Corporation) is used to perform some reporting and batch processing. It is a procedural language into which SQL statements can be embedded.

5. Until PeopleTools 7.x, Data Mover was used to import patches and their projects directly into the PeopleTools tables. As of PeopleTools 8, the Application Designer will export and import projects directly to and from disk. In PeopleTools 8.4, these project files are in XML format.

- The **PeopleSoft Query** tool (Query) still exists as a Windows client utility, although this functionality is also available in the PIA. Users can use this tool to develop and run ad hoc SQL queries without knowledge of SQL. Queries can be migrated with the Application Designer.
- **Seagate Crystal Reports** is used for reports that require a sophisticated look and that include graphics. Crystal is capable of connecting directly to the database via an ODBC driver. Within PeopleTools it connects via a PeopleSoft ODBC driver that presents PeopleSoft queries as database procedures. The ODBC driver can connect in either two- or three-tier mode. Crystal only runs on Windows, so reports can only be scheduled on a Windows process scheduler.
- **nVision** is a reporting utility that plugs into Microsoft Excel. Although it can be used in any PeopleSoft module, it is most extensively used in General Ledger reporting. A user can drill down into a report, unrolling hierarchical data, such as a chart of accounts. In PeopleTools 8.4, the reports can be executed and viewed in the PIA,[6] or nVision can be installed on the client and executed in two- or three-tier modes.

Evolution of the PeopleSoft Architecture

Over the years and releases, PeopleSoft has progressively built on its previous achievements. I think that the easiest way to understand the current PeopleSoft architecture is to review how it evolved.

PeopleSoft was founded in 1987 and was launched on the crest of the client/server technology wave. Some of us can still remember a time when personal computers were just starting to appear within the workplace. Bill Gates' vision of a computer on every desktop was still a radical idea. Before then, the typical model for business computing was to use a monolithic mainframe. The following sections describe how computing models have evolved.

The Monolithic Mainframe

Traditionally, the mainframe was a beast that lived deep in the bowels of the data center, tended by surly white-coated technicians. As depicted in Figure 1-2, the mainframe was responsible for all aspects of the application; data access, business logic, and presentation layers were all handled centrally. The users accessed the system via completely dumb green-screen terminals that were often wired directly to a serial port on the back of the mainframe.

6. It has always been possible to schedule nVision reports to run on a process scheduler, but only on a Windows server. In PeopleTools 8, the generated reports are retrieved from the report repository. When a user drills down into an nVision report in the PIA, the drill-down is scheduled to run on a Windows process scheduler and the user must wait for the report to execute and complete.

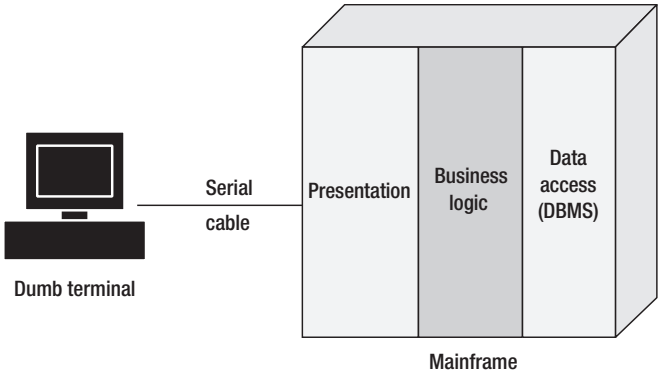


Figure 1-2. *Monolithic mainframe model*

Today, the most obvious legacy of this model on a PC is the choice choice of emulation in a terminal emulator program.

Client/Server Architecture

The appearance of the Apple II in 1977 and the IBM Personal Computer (PC) in 1981 brought processing power to the desktop, and the original killer desktop application was VisiCalc, a spreadsheet package released in 1979. By the end of the 1980s, PCs were becoming common in the workplace. When they were also connected to networks, client/server applications started to appear.

In a typical client/server program, depicted in Figure 1-3, all the business logic, as well as the presentation layer, is contained within the client program itself and executes on the client machine. Only the database remains central.

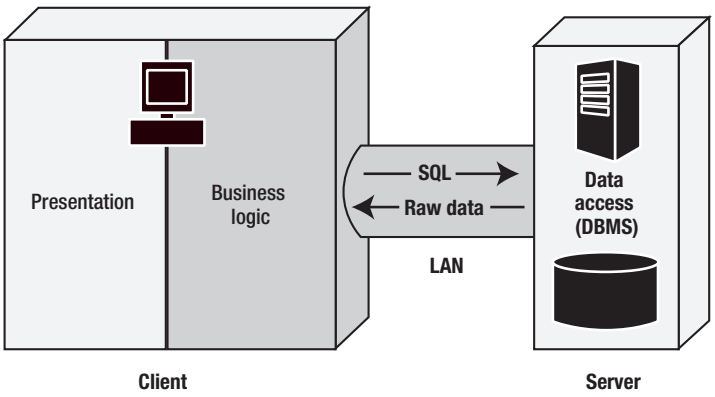


Figure 1-3. *The original client/server model*

This two-tier model started to show strain in a number of areas. The traffic across the network was mostly SQL conversations with the database, and because a relatively simple action in an application could generate many messages, network latency became a problem. Another issue was the problem of rolling out new clients and synchronizing the rollout with changes to the database.

Up to version 6, PeopleSoft was a fairly typical two-tier client/server application. PeopleTools and the database connectivity software (SQL*Net in the case of Oracle) had to be installed on every desktop.

The online parts of PeopleSoft applications were stored as metadata in the PeopleTools tables in the database. The client program loaded, interpreted, and executed the instructions in these tables, and the development tools manipulated this metadata. This approach mitigated, but did not eliminate, the problem of rolling out new versions of the client every time the slightest application change was made. However, the act of querying the PeopleTools tables in turn created significant amounts of database activity and network traffic, so the PeopleTools client cached the information that it retrieved from the PeopleTools tables to the local disk of the Windows PC. If the cache was up to date, the PeopleTools client did not query the PeopleTools tables. This approach is still used in the current version of PeopleTools.

Three-Tier Architecture

Three-tier computing models emerged to deal with the limitations of the two-tier model. As much as possible of the business logic layer was moved back into the data center, close to the database (see Figure 1-4). This reduced the size of the client, and the volume and number of messages that needed to travel between client and server.

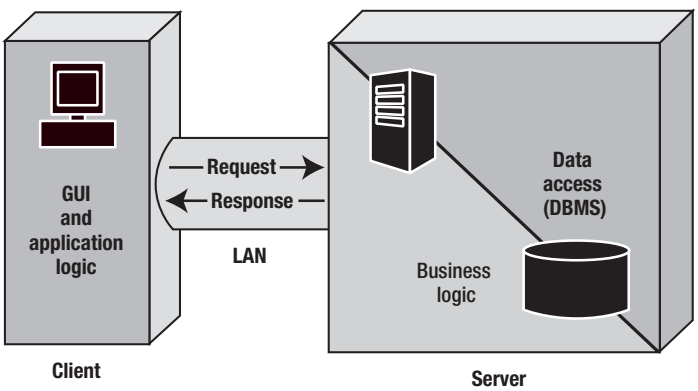


Figure 1-4. *A first-generation three-tier client/server model*

Oracle developed PL/SQL (Procedural SQL) to make it possible to write procedural program code that could either be stored and executed within the database or within Oracle applications or bespoke applications written with Oracle development tools. In this scenario, some of the business logic was moved inside the database server itself.

However, PeopleSoft's platform-independent approach precludes the use of database server-based code. None of the PeopleSoft application is executed within the database, although much of it is stored in the database in the form of metadata. However, in version 8, PeopleSoft has started to make limited use of database triggers.

When PeopleSoft introduced a three-tier client with an application sever in PeopleTools 7.0, it chose to use BEA's Tuxedo product. The application server is a completely separate tier from the database, and it can indeed be placed on a completely different machine. This model (see Figure 1-5) can therefore be scaled horizontally by using multiple application servers on different nodes.

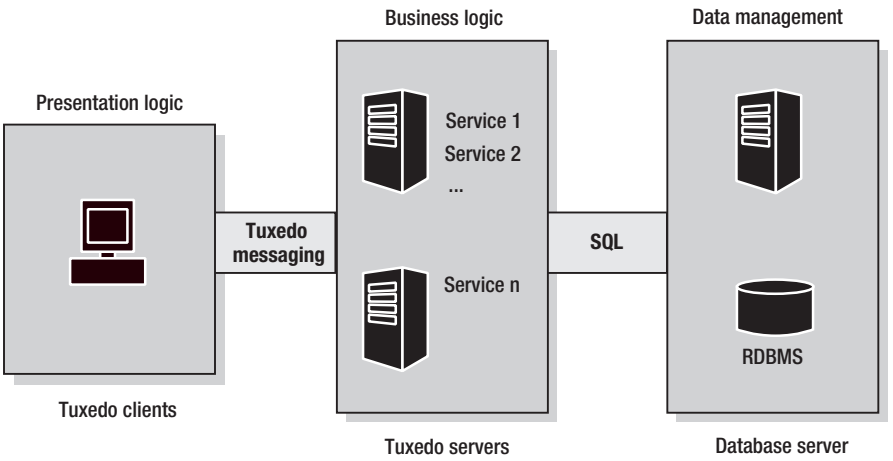


Figure 1-5. *The Tuxedo second-generation three-tier client/server model*

PeopleSoft developed its own procedural language, PeopleCode, in which it codes additional business logic. Pieces of PeopleCode are defined on certain columns in the application, and that code will be executed when certain events in the client occur. In the two-tier client, PeopleCode was executed on the client, but in three-tier mode most of the PeopleCode is executed by the application server.

As a result, it is now the application server, rather than the client, that makes the connection to the database. The server loads the application metadata from the PeopleTools tables, executes most of the application code, and retrieves and updates the application data. Both the client and the application server cache the application code retrieved from the PeopleTools tables. The end result is that there are fewer connections to the database, easing memory management.

■ **Note** Only application code (metadata) is cached by client processes. This includes the application server, and Application Engine from PeopleTools 8. Since PeopleTools 7, application data is never cached by the client processes, although in earlier versions the PeopleTools client could optionally cache static application data.

Not only is the total volume of network traffic between the client and the application server reduced in the three-tier model, but the number of network messages is greatly reduced. There are fewer, although individually larger, messages in three-tier mode. Every time a two-tier client fetches a row from a SQL cursor, a message is sent to the database, and the client must wait for a response. In a three-tier application, all of the data, possibly from many cursors, is returned to the client in a single message. Thus, three-tier clients are less susceptible to network latency.

Four-Tier Architecture

In the latest release, version 8, PeopleSoft has replaced the Windows client with its Internet architecture. The client is now just a web browser. There is no longer a problem rolling out PeopleSoft software or packaging it into a corporate desktop. Now you can use PeopleSoft from something other than a Windows PC.

This change has been achieved by building on the existing PeopleSoft three-tier model. PeopleSoft now delivers a a series of Java servlets that run on the web server (see Figure 1-6), and this servlet is the client of the application server. The web server can be placed on the same or a different physical server from the application server.

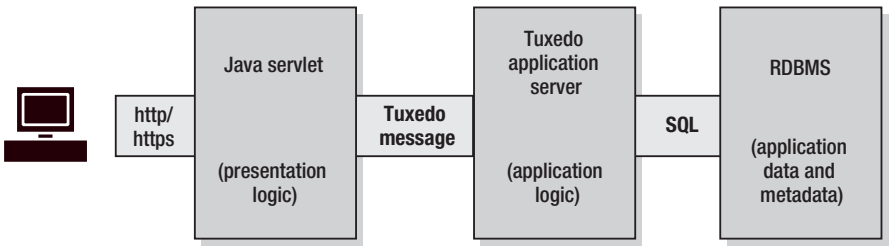


Figure 1-6. *The PeopleSoft Internet Architecture (PIA)*

"No code on the client" was the mantra of every PeopleSoft presentation I saw when version 8 was introduced. PeopleSoft code does execute on the client, but you don't have to preinstall it. The HTML pages include JavaScript, which is generated by the application server, and is then served up to the browser via the servlet. The JavaScript is executed when certain buttons are pressed or fields are navigated. Some output formats require Microsoft Excel or Adobe Acrobat to be installed on the client.

Release History

PeopleSoft's applications and PeopleTools have different version numbers that change independently. The first digit for both of them, sometimes called the Enterprise Release Number, is now always the same. There was no PeopleTools version 4 because it's an unlucky number in some Asian countries. Financials 3 and HRMS 4 ran with PeopleTools 3. Some Enterprise releases have also had major upgrades (PeopleTools 7.5 and 8.4, for example) to both the application

and PeopleTools that must effectively be treated as if they were new Enterprise releases. It is confusing, so it is important to specify whether a version number refers to an application or to PeopleTools.

PeopleTools versions up to 5.x were 16-bit Windows applications.[7] They only connected directly to the database via the database API, which on Oracle is SQL*Net. Since it was a 16-bit application, it was only possible to use 16-bit SQL*Net. PeopleTools 5.x saw a major overhaul in the interface, replacing push buttons with a menu. PeopleTools 5.1 has not been supported since 1999, and was never certified for Y2K.

PeopleTools 6 was a 32-bit Windows application. It corresponded to the introduction of Windows 95, the first 32-bit Windows release. However, it was still almost exclusively a two-tier application. I say "almost" because this release saw the first appearance of the BEA Tuxedo application server, although it was only used for Remote Call functionality. This allowed batch processes to be initiated from the client, apparently synchronously, but to be executed on the application server. Within the delivered PeopleSoft applications, Remote Call was only used in Financials for online voucher editing and posting.

PeopleTools 7.0 was released in September 1997. This release introduced the full three-tier model in PeopleSoft. The Windows client was also able to connect to the BEA Tuxedo application server, which in turn connected to the database as the client did in two-tier mode. In this version, the separate development utilities (record designer, panel designer, and menu designer) were consolidated in the new Application Designer.

PeopleTools 7.5 was released in May 1998. It saw the consolidation of the application server—new application-server services were introduced to combine several service calls into one. This release also saw the introduction of the Java client, a Java applet that was downloaded to the browser on a client PC. It ran in the Java Virtual Machine (JVM) within the browser and connected to the application server. Java-enabled browsers were, at that time, still relatively new, and this client was never particularly popular, mainly due to the size of the applet download and memory leaks in some JVMs.

PeopleTools 8.0 was released at the very end of 1999, and it introduced a new and radical concept: a pure Internet client. Each screen or panel in the PeopleTools applications was a page in this iClient. This meant that the only software now needed on the user's PC was a standard web browser. The pages were rendered with JavaScript to enable the buttons on the page, and the client's session is a thread within a Java servlet that runs in a JVM that is either in or close to the web server. The servlet connects to the application server, just as the Java client did in the previous release.

In August 2000, PeopleTools version 8.1 was introduced. The iClient was renamed as the PeopleSoft Internet Architecture (PIA). Although the Windows client was still delivered, it was no longer a supported runtime environment. Query (the ad hoc reporting tool) and the nVision reporting plug-in for Excel still exist as Windows executable applications to provide an alternative to the PIA functionality. Application development is still done via a Windows client that connects in both two- and three-tier modes.

PeopleTools 8.4 is the second release of the PIA, and there have been some changes to the look and feel of the products. The breadcrumb navigation of version 8.1 has been replaced with a menu portlet. This release also no longer includes the Windows runtime client, `pstools.exe`. With the exception of Query and nVision, the PeopleSoft application is only available via the PIA.

Component Connectivity

Table 1-1 sets out the various PeopleSoft components and shows how they have connected to the database in different releases.

Table 1-1. *PeopleSoft Component Connectivity*

PeopleTools Version	5	6	7.0	7.5	8.0	8.1	8.4
Windows client	Two-tier	Two-tier (except remote call)	Two-tier and three-tier			Delivered but not supported—two-tier and three-tier	Not delivered
nVision & Query	Two-tier		Two-tier and three-tier				Two-tier, three-tier, or PIA
Crystal Reports	Two-tier only		Two-tier or via PeopleSoft ODBC driver, which connects in either two-tier or three-tier modes				
Data Mover	Two-tier only						
Application Designer	Two-tier only		Two-tier and three-tier				
Application upgrade	Two-tier only						
Java client	n/a		Three-tier only		n/a		
IClient/PIA	n/a				Via application server		

As a result of the evolutionary development of PeopleSoft, different components require different types of connections:

- Database connectivity from client workstation PCs to the database server is a prerequisite for a PeopleSoft installation. It is required for two-tier connections.
- The Data Mover utility, used to import PeopleSoft objects into the database, only works in two-tier mode.
- The PeopleTools client programs connected in three-tier mode do not require database connectivity.
- The Application Designer can connect in both two-tier and three-tier modes. However, the application upgrade process within the Application Designer, which copies objects from one database to another, only works in two-tier mode.
- Crystal Reports is capable of connecting directly to the database via an Oracle ODBC driver, but within PeopleTools it connects via a PeopleSoft ODBC-style connection. This driver connects to the database in the same way as the Windows client. It is capable of connecting in either two- or three-tier modes.
- Cobol batch programs and SQR reports can only connect to the database in two-tier mode.

Developing and Administering PeopleSoft Systems

The diagram of the PIA (Figure 1-6) illustrates the most important point about PeopleSoft systems: they comprise a chain of linked technologies that stretch between the user and the database.

The database is fundamental to the entire structure. It must function efficiently, and it must itself be built upon solid foundations. However, as the PeopleSoft technology has evolved, additional tiers and various layers of software and infrastructure have been inserted between the user and the database. If any of these layers do not function efficiently, the users will be affected.

Developers and DBAs will have different perspectives on the system and its technologies.

The Developer

The PeopleSoft development tools provide an environment in which to manage development and upgrading that is consistent across all platforms. This is both a strength and a weakness. The advantage is that PeopleSoft delivers and supports a single set of tools and procedures for all database platforms. This means that a PeopleSoft developer needs one set of skills, regardless of the database that is used.

The disadvantage of this consistency is that while the developer defines the basic application components, much of the SQL in a PeopleSoft application is generated dynamically and does not appear in its final form in the application. This has the effect of isolating the developer from the database. When processes are migrated to a test or production database, the DBA may show some of the resulting SQL to the developer, who may then struggle to relate it to its source. (This will be discussed in Chapter 11.)

The DBA

The disadvantage of the PeopleSoft development tools for DBAs is that the application is hidden from them, and some of the tasks that are properly a part of their job must be either managed from PeopleSoft utilities or at least the change must be retrofitted back into PeopleSoft.

Usually DBAs have many different databases to manage, and most of that work is done with standard Oracle tools, so (they claim) they do not have sufficient time to manage a PeopleSoft database differently. However, the DBA needs to take the time to get sufficiently acquainted with the PeopleSoft tools or they will be in for some nasty surprises.

The following are just a few of the main "idiosyncrasies" that a PeopleSoft DBA might expect to encounter (with references to where they are covered in more detail in this book):

- If a DBA decides to add an index to a table to improve the performance of a query, that index should be specified in and built with the Application Designer or it could be lost (see Chapters 5 and 6).
- Changes to schema passwords must either be done with PeopleSoft utilities or be synchronized with PeopleSoft configuration changes (see Chapter 4).
- DBAs always want to know who is running a particular piece of SQL code (presumably so they can have a friendly word). The application server concentrates connections so that a database session does not correspond to a single user (see Chapter 2), but PeopleSoft can track which user's service request is on the server (see Chapter 9).

While it is helpful for DBAs to understand the application running on the database, it is essential that they at least understand the underlying technology. This book will focus heavily on how the PeopleSoft technology relates to the database.

Relationship Between the Developer and DBA

I believe that it is important for DBAs and developers to work closely together. The DBA needs to know what administrative tasks should be performed within PeopleTools, and how to manage them. Developers, in turn, need to know what effect they are having on the database. In my experience, some of the most successful and efficient PeopleSoft implementations are those where a DBA is dedicated to, if not fully integrated into, the PeopleSoft project.

It is almost inevitable that when a user reports a problem with the system, the finger of blame is instinctively pointed at the database, and the DBA becomes embroiled in the resolution (and blame-allocation) process. That alone is reason enough for both DBAs and PeopleSoft developers to work together to understand the relationship between the database and PeopleTools.

Chapter 9 looks at the metrics that can be obtained from the various PeopleTools tiers and that will help determine whether a performance problem is a database problem or not. Chapter 10 looks at the performance monitoring utilities that PeopleSoft introduced in release 8.4.

For a DBA to say that poor performance is a result of poor SQL and that there is nothing they can do about it is simply not an adequate response. If poor SQL is the problem, it needs to be addressed. PeopleSoft supplies packaged applications, but the majority of the SQL applications can be customized if necessary. Chapter 11 explains how to locate and adjust problem SQL.

Summary

This chapter has provided a high-level introduction to the PeopleSoft Enterprise technology. PeopleSoft is a chain of linked technologies that stretch between the user and the database, as was pointed out in Figure 1-6.

The database is fundamental to the entire structure, and it must function efficiently and be built upon solid foundations. However, as the technology has evolved, additional software tiers and infrastructure have been introduced between the user and the database. If any of these layers do not function efficiently, the users will be affected.

In a PeopleSoft environment, developers are so far removed from the system that the DBA is likely to be the only member of the team who can appreciate the whole picture. It is essential that the DBA know what PeopleSoft is doing to their database and how it is doing it.

CHAPTER 2

BEA Tuxedo: PeopleSoft's Application Server Technology

DBAs might be forgiven for wondering what a chapter about middleware is doing in this book at all. They might initially think that the application server has nothing to do with them. However, it is a fundamental component in the PeopleSoft architecture, and it has an intimate relationship with the database. The sizing and configuration of the application server or servers can have a significant effect on the database and on the flow of work from the end user to the database, and therefore on the performance of the whole system. The DBA must at least have an appreciation of Tuxedo.

This chapter provides an overview of the parts of Tuxedo that PeopleSoft utilizes. I have always found BEA's documentation to be impressive and valuable, and this chapter should provide sufficient background for you to start using that documentation.[1] PeopleSoft's product documentation, PeopleBooks, also includes a section on the application server.

What Is Tuxedo?

BEA Tuxedo[2], in its own documentation, describes itself as "middleware for building scalable multi-tier client/server applications in heterogeneous distributed environments." The name Tuxedo is an acronym standing for "Transactions Under Unix Extended for Distributed Operations," which is a fair description of how Tuxedo works.

When a program executes a subroutine, it simply executes code in the same executable, or it dynamically loads a library. Tuxedo allows that subroutine to execute synchronously on a different physical machine. It passes the parameters to the subroutine and the return codes back from it. Hence, Tuxedo is sometimes referred to as a messaging protocol. It is much more than that, but PeopleSoft only uses a part of what Tuxedo provides—the part referred to by BEA as the Application-to-Transaction Monitor Interface (ATMI). This book will also only deal with that part of Tuxedo.

---

1. The Tuxedo documentation is delivered with the product and is also available on the (currently) freely accessible BEA support web site (www.bea.com).
2. Tuxedo was originally developed in the Bell Laboratories in 1983, and in 1993 it was transferred to Novell. In 1996 BEA Systems entered into an exclusive agreement with Novell to distribute and continue developing the Tuxedo system (http://edocs.bea.com/tuxedo/tux81/overview/overviea.htm).

Every child has at some time made a telephone from two tin cans and a piece of string (see Figure 2-1). If you imagine that the string is the network and the tin cans are the client- and server-side programs, Tuxedo is the knots that hold the ends of the string to the cans. Tuxedo links the client and server sides of an application across a network.

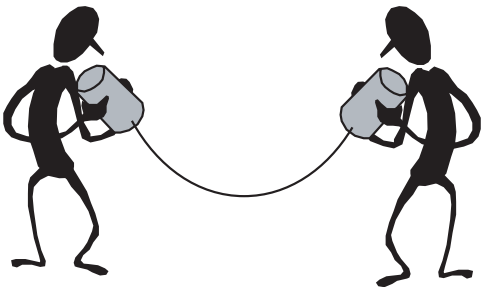


Figure 2-1. *Tuxedo is the knots in a tin can and string telephone.*

Conceptually, the PeopleTools Windows client was the same in two-tier mode as it is in three-tier mode. However, when it is used in three-tier mode, the client has been effectively cut in half, as shown in Figure 2-2. The front half consists of the presentation layer and some of the application logic processing. The back half includes the rest of the application logic layer, and this part makes the connection to the database. These two halves are reconnected by Tuxedo, as shown in Figure 2-3.

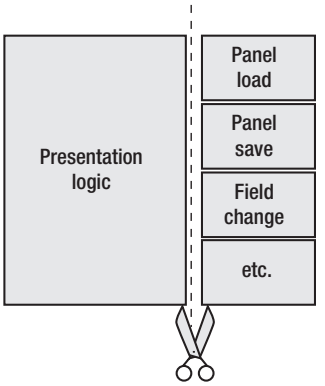


Figure 2-2. *The two-tier Windows client is divided in two when three-tier mode is used.*

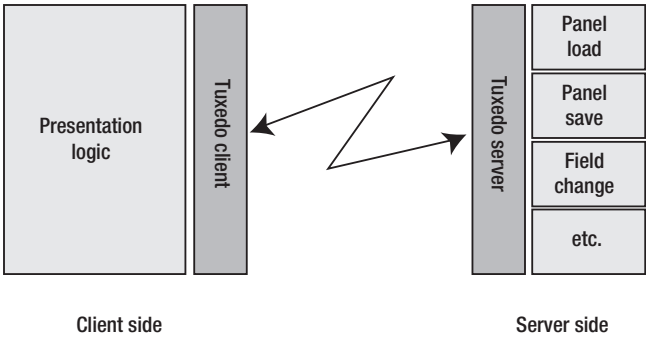


Figure 2-3. *Tuxedo connects the client to the functions in the server process.*

When this division was carried out, PeopleSoft had over 30 functions on the server side that executed at the request of the client, such as loading a panel, executing PeopleCode, executing particular SQL queries, saving the panels, and so on. Each of these functions became a Tuxedo service in the application server.

The application server consists of a number of server processes that communicate with the client via shared memory segments and queues. Some of those processes are delivered by Tuxedo to manage the communication within the application server, but most have been developed by PeopleSoft to service the requests from the PeopleTools clients and include some libraries provided by BEA. The service requests correspond to subroutines in the application server processes that execute the application. Tuxedo provides the infrastructure to route the messages to the appropriate server process and back to the client.

The structure of the application server has not changed significantly with the introduction of the PeopleSoft Internet Architecture (PIA) in PeopleTools 8, although the balance of the workload has shifted. Another tier has been added, and a Java servlet is now the client to the application server, instead of a Windows application. These are the principal changes:

- The application server performs more of the work in PeopleTools 8. There is a new set of PIA services, reflecting the fact that the PeopleTools client was heavily rewritten in PeopleTools 8. There are fewer services overall, but they are more generic. The application server remains stateless.
- The application server still executes all the PeopleCode and submits SQL to the database, and now it also generates all the HTML, JavaScript, and images, packages them into messages, and passes them to the servlet.
- The Java servlets provide a thinner presentation layer than the previous Java client applet was. They unpack the Tuxedo messages, writing the JavaScript and images to the web server file system and passing the HTML directly back to the browser.
- The PeopleSoft application server's processes make persistent connections to the database. All of the online activities from real live users, some of the reporting activity, and some of the interfaces to and from other systems are provided via the application server.

PeopleSoft has used Tuxedo 6.5 since PeopleTools version 7.5. With PeopleTools 8.44, Tuxedo 8.1 was introduced. While there are many new features in this upgrade, PeopleSoft has not changed the way it uses Tuxedo. All of the material in this chapter is applicable to both versions of Tuxedo.

The Simple Application Server

PeopleSoft delivers fully compiled client and server processes, and it defines and develops all the services. The source code is not revealed to the user, and there is neither opportunity nor reason to make changes. However, PeopleSoft also ships a full development version of Tuxedo as part of the PeopleSoft distribution, and this includes a number of sample applications. One of these, the Simple application (shown in Figure 2-4), gives some insight into how Tuxedo works. It illustrates how a call to a subroutine can be distributed between two processes. All the source files can be found in `$TUXDIR/samples/atmi/simpapp`.

The Simple application has only one function: to convert a string to uppercase. A function called `TOUPPER` has been placed in an application server, and a service called `TOUPPER` has also been defined and is called from the client.

The Simple client can be called from the command line. The string that is to be converted to uppercase is specified as the first parameter to the command, as follows.

Listing 2-1. *Simple application command line*

```
simpcl "Hello World"
```

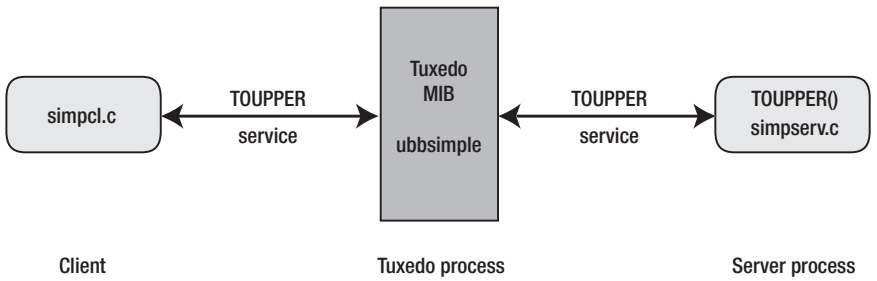


Figure 2-4. *The Simple client and application server*

The Simple Client

The C code examples in this section are taken from the Simple client, `simpcl.c`, and they illustrate the actions that a Tuxedo client must perform.

The client process must first connect to the Tuxedo Management Information Base (MIB) with the `tpinit()` function, as shown in Listing 2-2. The MIB is a shared memory segment that describes the application server. It is also referred to as the Bulletin Board.

Listing 2-2. *The client attaches to the Tuxedo MIB*

```
/* Attach to System/T as a Client Process */
if (tpinit((TPINIT *) NULL) == -1) {
     (void) fprintf(stderr, "Tpinit failed\n");
     exit(1);
}
```

Memory is allocated by the `tpalloc()` function for the outgoing and incoming return message, the locations being assigned to pointers `sendbuf` and `rcvbuf` respectively, as shown in Listing 2-3. The length of the message is derived from the length of the string specified on the command line, `argv[1]`.

Listing 2-3. *The client allocates memory for Tuxedo messages*

```
sendlen = strlen(argv[1]);
if((sendbuf = (char *) tpalloc("STRING", NULL, sendlen+1)) == NULL) {
    (void) fprintf(stderr,"Error allocating send buffer\n");
    tpterm();
    exit(1);
}
if((rcvbuf = (char *) tpalloc("STRING", NULL, sendlen+1)) == NULL) {
    (void) fprintf(stderr,"Error allocating receive buffer\n");
    tpfree(sendbuf);
    tpterm();
    exit(1);
}
```

The string to be converted to uppercase is copied from the first command-line parameter to the send buffer, and the service call is made by the `tpcall()` function. Pointers to the buffers are then passed to this function. It is a synchronous call, so `tpcall()` will wait until it gets a response from the server, or times out.

Listing 2-4. *The client submits the TOUPPER service request*

```
(void) strcpy(sendbuf, argv[1]);
ret = tpcall("TOUPPER", (char *)sendbuf, 0, (char **)&rcvbuf, &rcvlen, (long)0);
...
```

The return message from the service call is placed in the receive buffer by `tpcall()`.

Listing 2-5. *The client prints the return message*

```
(void) fprintf(stdout, "Returned string is: %s\n", rcvbuf);
```

Finally, the client releases the memory that it allocated for the service call, and it disconnects from the application server.

Listing 2-6. *The client releases memory, disconnects, and terminates*

```
/* Free Buffers & Detach from System/T */
tpfree(sendbuf);
tpfree(rcvbuf);
tpterm();
return(0);
```

The Simple Server

The Simple server, `simpserv.c`, is little more than a function that has the same name as the service. The parameter to the function is a pointer to a Tuxedo-defined memory structure, and incoming data is retrieved from that structure. The structure is defined, as shown in Listing 2-7.

Listing 2-7. *An extract from* `%TUXDIR%/include/atmi.h`

```
/* interface to service routines */
struct tpsvcinfo {
#define XATMI_SERVICE_NAME_LENGTH  32
    char     name[XATMI_SERVICE_NAME_LENGTH]; /* service name invoked */
    long     flags;             /* describes service attributes */
    char     *data;             /* pointer to data */
    long     len;               /* request data length */
    int      cd;                /* connection descriptor */
    long     appkey;            /* application authentication client key */
    CLIENTID cltid;             /* client identifier for originating client */
};
typedef struct tpsvcinfo TPSVCINFO;
```

Listing 2-8 shows the TOUPPER service. When it is compiled, a Tuxedo stub is added. When the server process is started, the Tuxedo stub connects the server to the MIB, and it polls for incoming service requests. When it finds one, it takes the message, populates the `tpsvcinfo` structure, calls the service routine, and returns the result via the structure.

Listing 2-8. *An excerpt from* `simpserv.c`

```
TOUPPER(TPSVCINFO *rqst)
{
    int i;
    for(i = 0; i < rqst->len-1; i++)
        rqst->data[i] = toupper(rqst->data[i]);
    /* Return the transformed buffer to the requestor. */
    tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
}
```

The Simple Tuxedo Domain

Between the client and the server is the Tuxedo MIB (also referred to as the Bulletin Board). The Tuxedo configuration file is compiled from the source file `ubbsimple`, shown in Listing 2-9. It specifies that a single server advertises a single service. When the service request is received, it is routed to the server process that handles it.

Listing 2-9. *An excerpt from* `ubbsimple`

```
*SERVERS
DEFAULT:
    CLOPT="-A"
```

```
Simpserv    SRVGRP=GROUP1 SRVID=1

*SERVICES
TOUPPER
```

More information about the sample applications can be found in the Tuxedo documentation, and on the Tuxedo web site (http://e-docs.bea.com/tuxedo/tux81).

Anatomy of the Application Server

Essentially, PeopleSoft works in exactly the same way as the Simple application. Tuxedo passes parameters to certain functions between client and server. The only difference is that there are more server processes and more services, and the Simple application example did not include the Tuxedo listener processes.

Processes, Memory, and Messages

A Tuxedo application server domain consists of a number of server processes that communicate via shared memory segments and message queues. These structures are a part of the Unix Interprocess Communication (IPC) model. They are created and administered using Unix system functions that are supplied as a standard part of the operating system.

There is no concept of protected and shared memory on Windows, so BEA has provided `tuxipc.exe`, the BEA Process Manager service (called the Tuxedo IPC Helper service in Tuxedo 6.5), which supports the Unix IPC system call functions required by Tuxedo.[3] BEA has also implemented the `ipcs` and `ipcrm` commands on Windows. The Tuxedo documentation does not explain these commands because they are standard Unix commands, common to all flavors.

When any Tuxedo application server domain is booted, the first process to be started is the Bulletin Board Liaison process, called BBL. This process is the heart of the application server. It reads the configuration of the domain from a binary configuration file, which in a PeopleSoft domain is called PSTUXCFG (see Figure 2-5). The BBL then establishes a shared memory segment, referred to as the Bulletin Board (BB) or Management Information Base (MIB), some message queues (determined by the specifications set out in the configuration file), and two semaphores.

The Bulletin Board holds all the information about the rest of the application server domain. It is used as a form of database by the application server processes to determine how they should behave.

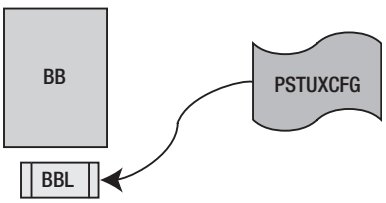


Figure 2-5. *The Bulletin Board is established by the BBL process.*

3. This means your Windows server now has a Unix kernel to tune! See Chapter 13 for more on this.

I'll start by describing the Tuxedo structure required for the Windows three-tier client processes, and add the PIA structure later on.

The Workstation Listener (WSL) is a Tuxedo process in the application server domain. It is configured to listen on a specified IP address and port for incoming connections from Tuxedo client process, which are any of the PeopleTools Windows client processes that operate in three-tier mode.

The WSL spawns at least one Workstation Handler (WSH) process, and it can be configured to start further handlers on demand. The WSHs listen for incoming service requests on the same IP address as the WSL and, unless they are configured to use a specific range of ports, will use the next available port after the WSL port.

Initially the client contacts the WSL on the port specified in the configuration. The WSL then assigns the client to one of the WSH processes. The client then closes the connection to the WSL and thereafter only communicates with the WSH. This initial communication process is outlined in Figure 2-6.

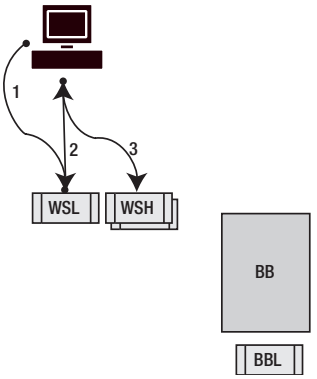


Figure 2-6. *A Tuxedo client making an initial WSL connection*

The client processes must be told where to find a WSL. They can be configured to load balance or fail over between multiple WSLs. WSL addresses and ports are configured in the Workstation Configuration Manager (Figure 2-7).

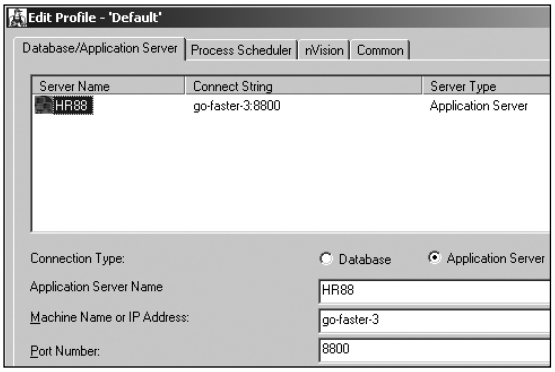


Figure 2-7. *Configuring a workstation listener for a Windows client*

PeopleSoft delivers various server processes, and each new release of PeopleTools has introduced more. Some server processes are optional and need only be configured when particular functions are required.

In PeopleTools 8.4, there must be at least one instance of each of the following four servers: PSAPPSRV, PSSAMSRV, PSMONITORSRV, and PSWATCHSRV. When the application server is booted, the minimum number of instances of the specified servers are started. Tuxedo can also be configured to spawn more instances of these servers on demand. All of the PeopleSoft server processes (except PSWATCHSRV) make a persistent connection to the database on startup.

Whenever a message passes between any two processes in the application server, it is sent via a queue to which the receiving process listens. Other information can pass between processes by being written to a shared memory segment.

The following simplified steps (illustrated in Figure 2-8) describe the activity in the application server during a transaction with a Windows client.

1. The service request is sent by the client to the WSH process to which it is connected.
2. The WSH looks up the service on the BB to determine which server or servers are advertising the service and which queues lead to those servers.
3. The WSH then enqueues the message requesting that service on an appropriate queue. In this case the service is placed on the queue called APPQ. Note that not all the queues are explicitly named in Tuxedo.
4. The PSAPPSRV polls for service requests on APPQ, dequeues that request, and executes the PeopleSoft code associated with that service.
5. Processing the service request may involve several interactions with the data. The PSAPPSRV server may need to read or update the physical files that cache PeopleTools objects, as shown in this example.
6. Each WSH process has a return message queue. When the service is complete, the PSAPPSRV server enqueues the return message on the queue (WSHQ in this example) that leads back to the WSH process that sent the request.
7. The WSH process polls and dequeues the return message from the WSHQ.
8. WSH sends the return message back to the client that has been waiting for a response.

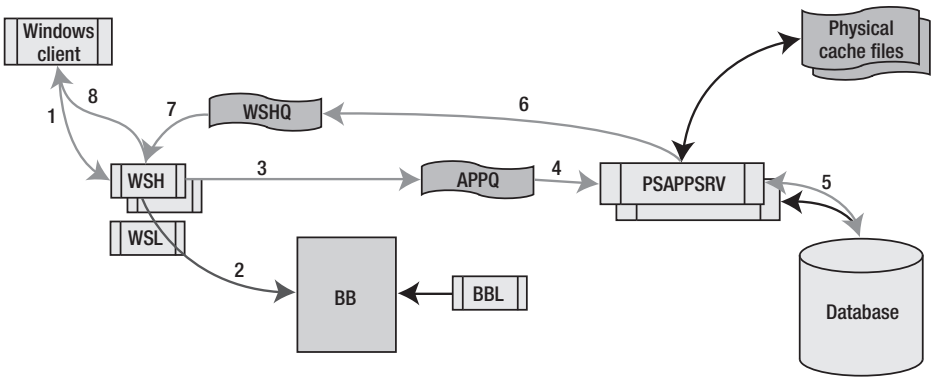


Figure 2-8. *A simplified three-tier Windows client transaction*

The PIA servlet in PeopleTools 8.4 runs in a JVM in the web server. Each PeopleSoft operator is assigned to a thread in the JVM, which is the operator's PeopleSoft session. The thread connects to the application server domain via the Jolt Server Listener (JSL), and the thread is then assigned to a Jolt Server Handler (JSH). The JSL and JSH are very similar to the WSL and WSH, except that they receive Java class requests.

Figure 2-9 illustrates what happens when a PIA transaction is processed by the application server. The steps in the figure are as follows:

1. A Java class request is received from the servlet.
2. The JSH needs to know which Tuxedo service corresponds to the Java class, so it enqueues a request for the JREPSVR.
3. JREPSVR picks up the request from the JSH.
4. JREPSVR looks up the Java class in the JREPOSITORY database (a flat file) to find the Tuxedo service that maps to the Java class.

5–6. JREPSVR returns the Tuxedo service name to the JSH via the queue JSHQ.

7. The JSH interrogates the BB to determine which queue the request should be placed on.

8–13. From this point, the transaction is the same as the three-tier transaction. The Tuxedo service request is sent to the appropriate server, and the response is transmitted back via the same JSH to the servlet.

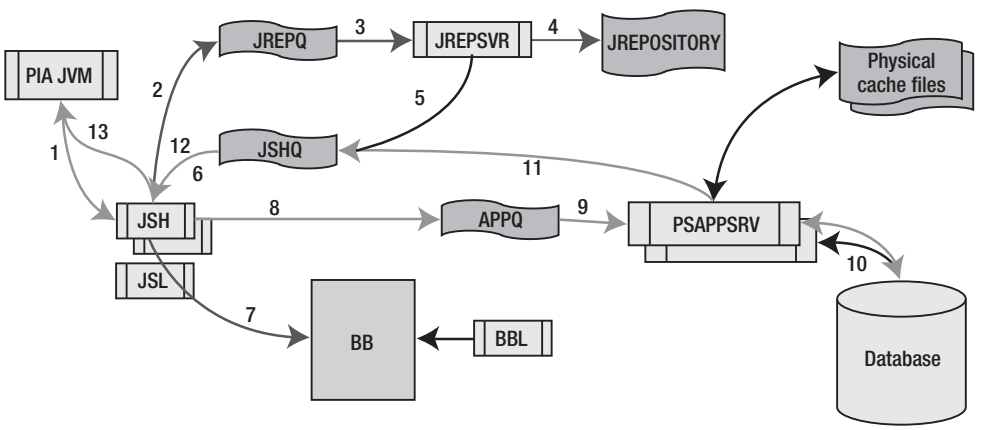


Figure 2-9. *A simplified PIA transaction*

IPC Resources

If you use the Unix `ipcs` command to examine all the IPC resources in use, you will see that the full picture is rather more complicated than was shown in Figure 2-8.

The following report was produced on Windows with the version of `ipcs` supplied by BEA with Tuxedo 8.1. It provides a snapshot of a PeopleSoft 8.44 application server domain. On Windows, the OWNER, GROUP, CREATOR, and CGROUP columns always have a value of 0. They have been removed from the following example for clarity.

Listing 2-10. *Typical* `ipcs` *output*

```
ipcs -a
IPCS status from BEA_segV8.1 as of Mon May 10 14:39:08 2004
T    ID     KEY        MODE    CBYTES QNUM QBYTES LSPID LRPID   STIME    RTIME    CTIME
Message Queues:
q  5632 0x00000000 --rw-rw-rw-     0    0  65536     0     0 no-entry no-entry 14:38:42
q   257 0x0000bbe2 -Rrw-rw-rw-     0    0  65536  4072  1716 14:39:00 14:39:00 12:10:42
q   515 0x00000000 -Rrw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:26:14
q   516 0x00000000 -Rrw-rw-rw-     0    0  65536  2904  3952 13:12:51 13:12:51 12:26:14
q   517 0x00000000 -Rrw-rw-rw-     0    0  65536  2904  3544 12:37:30 12:37:30 12:26:14
q   518 0x00000000 --rw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:28:11
q   519 0x00000000 --rw-rw-rw-     0    0  65536  1716   688 14:37:14 14:37:14 12:28:11
q   520 0x00000000 --rw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:28:46
q   521 0x00000000 --rw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:28:46
q   522 0x00000000 -Rrw-rw-rw-     0    0  65536  3544  2548 12:37:26 12:37:26 12:30:13
q   523 0x00000000 --rw-rw-rw-     0    0  65536  3544  2548 12:37:26 12:37:26 12:30:13
q   524 0x00000000 -Rrw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:30:59
q   525 0x00000000 -Rrw-rw-rw-     0    0  65536  1132  3648 14:39:07 14:39:07 12:30:59
q   526 0x00000000 -Rrw-rw-rw-     0    0  65536  1132  2616 14:39:07 14:39:07 12:30:59
q   783 0x00000000 -Rrw-rw-rw-     0    0  65536  2616  1708 12:46:04 12:46:04 12:32:22
q   784 0x00000000 --rw-rw-rw- 13608    1  65536  2616  1132 14:39:08 14:39:07 12:33:04
q  1297 0x00000000 --rw-rw-rw-     0    0  65536  1716  2904 12:37:30 12:37:30 12:33:04
q  3602 0x00000000 --rw-rw-rw-     0    0  65536     0     0 no-entry no-entry 12:37:26
q   532 0x00000000 --rw-rw-rw-     0    0  65536  1716  1132 14:38:53 14:38:53 14:38:51
T    ID     KEY        MODE    NATTCH  SEGSZ  CPID  LPID   ATIME    DTIME    CTIME
Shared Memory:
m    50 0x0000bbe2 --rw-rw-rw-   27 646432  1716  4072 14:39:00 14:39:00 12:10:40
m   101 0x00000000 --rw-rw-rw-    3    504  2196  3544 12:26:14 no-entry 12:26:14
m   102 0x00000000 --rw-rw-rw-    3   1112   260  2616 12:30:59 no-entry 12:30:59
T     ID     KEY        MODE       NSEMS   OTIME    CTIME
Semaphores:
s  1024 0x0000bbe2 --ra-ra-ra-       5 14:39:07 12:10:40
s  3073 0x00000000 --ra-ra-ra-      52 13:34:04 12:10:40
```

If you start the application server processes one by one and watch the changes in the `ipcs` output, you can determine which queues are associated with which processes. The LSPID and LRPID columns show the IDs of the processes that last sent a message to and retrieved a message from the relevant queue. Hence, it is possible to determine the flow of messages between servers and queues.

Figure 2-10 shows the full picture. The process, queue, and segment IDs in the figure correspond to the preceding `ipcs` report.

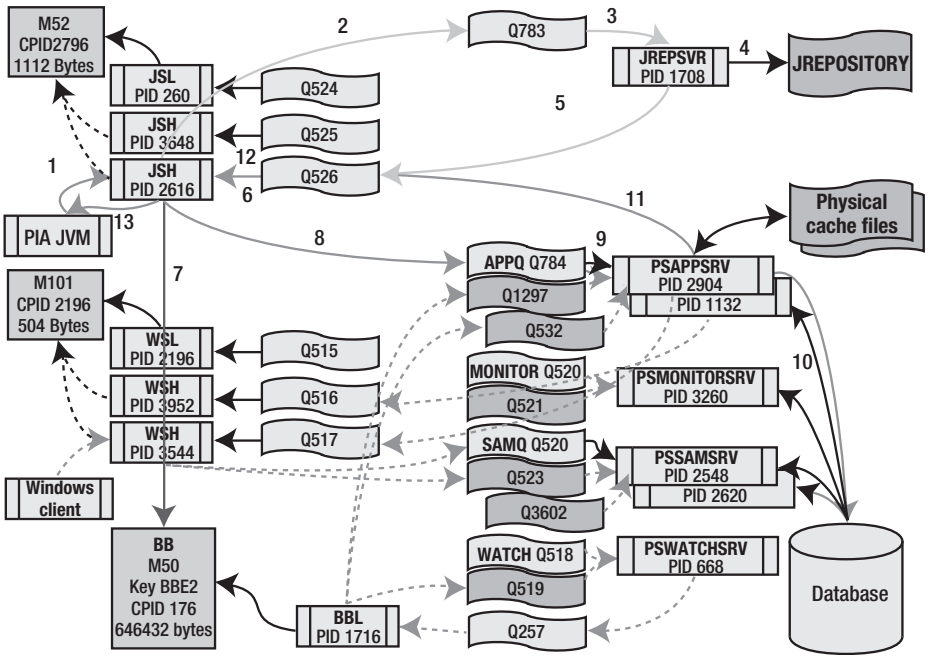


Figure 2-10. *A PIA transaction showing the full IPC model*

Various additional queues are created as each process starts. Each is used to send messages to a specific server. You should think of the queues as being associated with the process that receives the message via the queue. These are the queues you will see:

- A queue is created for each instance of the Tuxedo command-line utility `tmadmin` when it is running, and the related queue is deleted when the instance is terminated.
- The BBL process has one queue. It receives messages from PSWATCHSRV and `tmadmin`.
- A queue is created for each Tuxedo message queue (APPQ, SAMQ, etc.). These queues are used to send service request messages from the handler processes to the server processes, and they are revealed in `tmadmin` with the `printqueue` (or `pq`) command.
- A queue is created for each PeopleSoft server process (PSAPPSRV, PSSAMSRV, PSWATCHSRV, etc.). This is an administrative queue for receiving instructions from the BBL.
- There is one queue for each WSL and each WSH. The WSH queues are used to receive return messages from the application server processes after the service requests have been processed. The queue to the WSL process appears not to be used.
- There is a queue for each JSL and JSH process (as for the WSL and WSH processes).

There are three shared memory segments:

- The Bulletin Board is created by the BBL process.
- A small shared memory segment is created by the WSL. It is used to permit the WSL and WSH processes to communicate.
- The JSL process similarly creates a small shared memory segment for interprocess communication.

tmadmin and ipcs

The `printqueue` command in the `tmadmin` utility reports only on the status of the queues that carry inbound service-request messages to the application server processes.

In this example, `printqueue` (`pq`) reports that the APPQ that serves requests to two PSAPPSRV processes has a message queued on it.

Listing 2-11. *Tuxedo* `printqueue` *output*

```
> pq
Prog Name      Queue Name  # Serve Wk Queued  # Queued  Ave. Len    Machine
---------      ------------------ ---------  --------  --------    -------
PSSAMSRV.exe   SAMQ              2         -         0         - GO-FASTER+
JSL.exe        00095.00200       1         -         0         - GO-FASTER+
WSL.exe        00001.00020       1         -         0         - GO-FASTER+
JREPSVR.exe    00094.00250       1         -         0         - GO-FASTER+
PSMONITORSRV.e MONITOR           1         -         0         - GO-FASTER+
PSAPPSRV.exe   APPQ              2         -         1         - GO-FASTER+
BBL.exe        48098             1         -         0         - GO-FASTER+
PSWATCHSRV.exe WATCH             1         -         0         - GO-FASTER+
```

The output in Listing 2-11 corresponds to the entry in the `ipcs` report in Listing 2-12 that provides this information:

- One message of 13,608 bytes is on queue 784.
- The last message sent to the queue was placed by process 2616, a JSH.
- Process 1132, a PSAPPSRV, was the last process to receive a message from this queue.

Listing 2-12. `ipcs` *report*

```
ipcs -a
IPCS status from BEA_segV8.1 as of Mon May 10 14:39:08 2004
T    ID     KEY        MODE    CBYTES QNUM QBYTES LSPID LRPID   STIME    RTIME    CTIME
Message Queues:
...
q   784 0x00000000 --rw-rw-rw- 13608    1  65536  2616  1132 14:39:08 14:39:07 12:33:04
...
```

PIA Servlets

The PIA servlets are not part of the application server; they are its clients. It is important to understand their relationship with the application server.

When you boot a WebLogic server, you are effectively starting a JVM that runs a servlet engine. Various PIA servlets are then registered with this servlet engine. The WebLogic server is not really a web server at all—it is simply shipped with a servlet that acts as a web server.

The PIA connects to the application server as illustrated in Figure 2-11 and outlined in the following steps:

1–2. HTTP requests for the PeopleSoft servlet are received by the web server listener and routed to the servlet. When a PIA session is established, a new servlet thread is created. That thread is the operator's session and it is stateful. HTTP messages are not stateful, but an in-memory cookie is sent back to the browser, and that cookie is then sent to the web server with subsequent HTTP requests to identify the user's servlet thread.

3. Each servlet thread makes a persistent connection to the application server via one of the JSH processes. The JSL ports are specified in the servlet configuration file, `configuration.properties`. All of the HTML, JavaScript, and graphics are generated by application server and are sent back to the servlet in a single message from Tuxedo.

4. The servlet unpacks the message, writing the static files (JavaScript and graphics) to the physical file system that is referenced by the web server. The main HTML page is sent back via the web server thread to the client.

5–6. The main HTML page sent back to the client will contain references to the static files. If those files have not already been cached locally, the browser will make further HTTP requests for those files that will be served by the web server without further reference to the servlet.

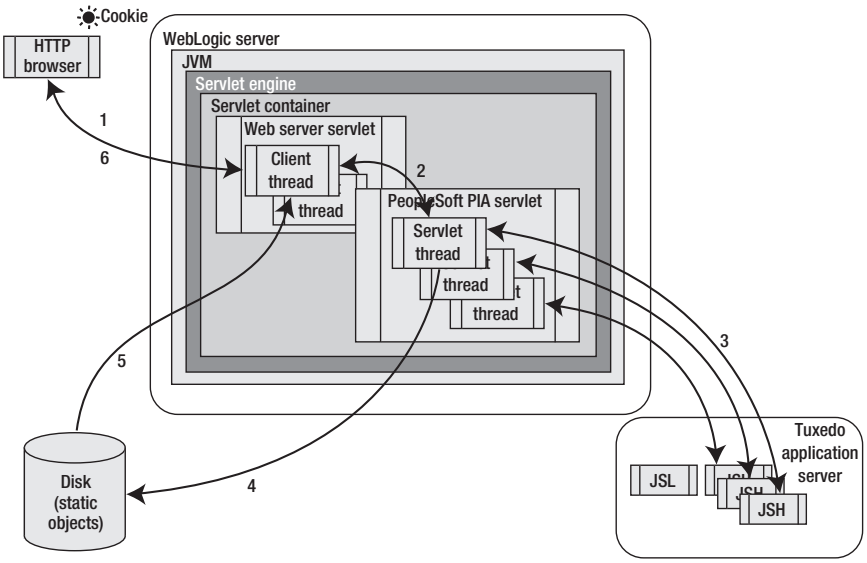


Figure 2-11. *A PIA servlet transaction*

PeopleSoft also supports Apache HTTP Server 1.3 with `mod_jserv` as a servlet container, but only with PeopleTools 8.1. PeopleTools 8.4 also supports IBM Websphere, but it only supports Apache as an HTTP server. In these servers, the web server itself is a separate process distinct from the JVM. Otherwise the function of the PIA is the same on all web servers.

Summary

This chapter has discussed the architecture of the Tuxedo application server in some detail. The client and server processes, written by PeopleSoft, communicate with each other via the infrastructure provided by Tuxedo and using shared memory segments and queues provided by the Unix IPC system. Tuxedo passes the parameters in the function calls to the functions in the server, and it passes the return values back again. The application server also provides a degree of connection concentration so that relatively few server processes can handle many client sessions.

CHAPTER 3

Database Connectivity

All but one of the database objects in a PeopleSoft database are contained in a single Oracle schema. All PeopleSoft processes that connect directly to the database via SQL*Net go through the standard PeopleSoft signon processing, negotiating security and ending up connected to that schema. (Thereafter, security is handled by PeopleTools, not the database.)

This chapter describes the database schemas that PeopleSoft creates, their privileges, and how those privileges are used by the signon process, explaining what this reveals about the structure of PeopleSoft databases.

PeopleSoft Database vs. Oracle Database

It is not uncommon in the IT industry for different vendors to use the same word to mean very different things. *Database* is one of those words.

The word *database* is often used by PeopleSoft to refer to the collection of tables in the PeopleSoft administrative schema within an Oracle (or other) database. In contrast, Oracle would describe a database as the collection of physical data files, log files, and control files. An Oracle *instance* is the collection of intercommunicating processes that administer the database. The importance of these distinctions will become apparent as I describe the signon process.

Sometimes even DBAs are guilty of using terminology carelessly. Unless you are running Real Application Clusters (or Oracle Parallel Server in Oracle 7) there is a one-to-one relationship between instances and databases. You may hear DBAs talking about starting the database when really they mean starting the instance.

PeopleSoft recommends that each PeopleSoft database be created in a separate Oracle database. Thus, each instance can be started, shut down, backed up, and tuned independently. Nevertheless, it is possible to have more than one PeopleSoft database in a single Oracle database, in which case each PeopleSoft database would reside in a different Oracle schema, and the signon process would have to connect accordingly.

On the rare occasions where I have seen this arrangement, a number of PeopleSoft development databases have been co-located. With the exception of customizations or patches that have not yet been migrated, each of the PeopleSoft databases will contain the same tables with the same structures. Unless the tablespace model is explicitly overridden, the same table in different PeopleSoft databases, and therefore in different Oracle schemas, will be built in the same tablespace. Thus, each tablespace will contain objects for all the PeopleSoft databases that reside in the same Oracle database.

I consider that the administrative complexity of co-locating PeopleSoft databases generally outweighs the additional memory, disk, and CPU overhead in having additional Oracle databases and instances.

Oracle Database Users

Every process that makes a two-tier connection to the database identifies itself with a PeopleSoft user or operator ID. The purpose of this signon processing is to securely validate that the PeopleSoft operator, authenticated by password, is permitted to access the PeopleSoft application.

As such, it will be useful to quickly review the database user accounts available in a PeopleSoft database before walking through the actual signon process. From version 8 onwards, a PeopleSoft database requires only three Oracle database schemas:

- Owner/Access ID: This schema contains most of the database objects, and it is as this database user that PeopleSoft processes access the database.
- Connect ID: This low-security database user is used by the signon process until the password is validated.
- PS: This schema contains a table that describes which PeopleSoft databases are in the Oracle database.

Owner ID/Access ID (SYSADM)

Once the signon process is complete, a PeopleSoft process (such as a two-tier client or an application server process) is connected to the database via the administrative schema, which is referred to as the "Owner ID" because it contains nearly all of the database objects. This schema can access any table or view in the PeopleSoft database. Access to objects in the database is controlled within the PeopleSoft application, rather than by the database.

■ **Note** The OwnerID is the schema that contains the PeopleSoft objects, while the AccessID is the database user with which PeopleSoft connects and references it. In a standard PeopleSoft installation there is only one AccessID, which is the same as the OwnerID.

By convention, the administrative schema is usually called SYSADM, although there is nothing to prevent you from using a different name. The password to this account is the key to the kingdom. It should be treated with the same respect as the password to the superuser (root) account on Unix or the passwords to the SYS and SYSTEM accounts on an Oracle database.

The Owner ID has certain privileges granted by a role called PSADMIN (discussed in the "Oracle Database Roles" section, shortly).

Connect ID (PEOPLE)

From PeopleTools 8 on, the first connection that each process makes to the database is via a low-security user account, referred to as the "Connect ID" and usually named PEOPLE. It is only granted CREATE SESSION privilege via the PSUSER role and three explicit SELECT privileges:

Listing 3-1. *SELECT privileges being granted to the PEOPLE account*

```
GRANT SELECT ON PSSTATUS TO PEOPLE;
GRANT SELECT ON PSOPRDEFN TO PEOPLE;
GRANT SELECT ON PSACCESSPRFL TO PEOPLE;
```

The Connect ID provides only the bare minimum of access to the PeopleTools tables in order to permit authentication by the signon process. Later, after successful validation of the password, the process reconnects as the Access ID.

Up to PeopleTools 7.x, instead of a single Connect ID, there was a database user corresponding to each PeopleSoft operator. Every time a PeopleSoft operator was created in the application, the PSUSER role was granted to that PeopleSoft operator ID. The PSUSER role contains only the CREATE SESSION privilege, and granting this privilege implicitly creates the database user, as can be seen in Listing 3-2.

Listing 3-2. *A PeopleTools trace showing a user being created in PeopleTools 7.5*

```
5-312  0.961 Cur#1 RC=0 Dur=0.310 COM Stmt=GRANT PSUSER TO NEWOP
5-313  0.811 Cur#1 RC=0 Dur=0.811 COM Stmt=GRANT SELECT ON PSLOCK TO NEWOP
5-314  0.531 Cur#1 RC=0 Dur=0.531 COM Stmt=GRANT SELECT ON PSOPRDEFN TO NEWOP
```

When an operator was deleted from the application under version 7.x, the grants were revoked, but this leaves the empty schema without any privileges.

When a PeopleSoft system has been upgraded from PeopleTools 7 to 8, it is common to find these empty schemas left behind because there is no step in the upgrade procedure to remove them. This is not a security risk, because no one can connect to these schemas. However, I still recommend that they be removed.

PS Schema

The PS schema is used to hold only the table PSDBOWNER, which maps the name of the PeopleSoft database to the schema in the database that holds it. PS.PSDBOWNER is the only table in a PeopleSoft database that is in a different schema from the rest of the objects.

The PS schema and PS.PSDBOWNER table are created during the installation procedure by the `dbowner.sql` script, shown in Listing 3-3. All privileges are revoked from the schema after the table has been created.

Listing 3-3. *An extract from* `dbowner.sql`

```
GRANT CONNECT, RESOURCE, DBA TO PS IDENTIFIED BY PS;
CONNECT PS/PS;
CREATE TABLE PSDBOWNER (DBNAME VARCHAR2(8) NOT NULL
, OWNERID VARCHAR2(8) NOT NULL ) TABLESPACE PSDEFAULT;
CREATE UNIQUE INDEX PS_PSDBOWNER ON PSDBOWNER (DBNAME) TABLESPACE PSDEFAULT;
CREATE PUBLIC SYNONYM PSDBOWNER FOR PSDBOWNER;
GRANT SELECT ON PSDBOWNER TO PUBLIC;
CONNECT SYSTEM/MANAGER;
REVOKE CONNECT, RESOURCE, DBA FROM PS;
```

As shown later in Listing 3-12, the PeopleTools 8.x signon procedure explicitly references the table and schema as PS.PSDBOWNER. The "PS" is hard-coded into the process, so you cannot choose a different name for this schema.

The creation of a public synonym in the `dbowner.sql` script is a throwback to earlier versions of PeopleTools where the signon procedure did not explicitly specify the schema. The synonym still permits the table to be referenced from the application, but this table can also be queried by any user connected to the database. It would be more appropriate to create a user synonym in the Owner ID schema and only grant SELECT privilege to that user.

When a PeopleSoft database is created, a Data Mover script (for example, `hr88ora.dms`) is generated to import the PeopleSoft objects into the Oracle database. An extract of that script is shown in Listing 3-4, where it also populates PS.PSDBOWNER with one row that describes that PeopleSoft database.

Listing 3-4. *An excerpt from* `hr88ora.dms`, *which builds a PeopleSoft database*

```
REM - Final Database cleanup
REM -
REM - Based on your inputs to Database Setup, you will be using
REM - ConnectID's to connect to your PeopleSoft Application
REM -
/
INSERT INTO PS.PSDBOWNER VALUES('HR88', 'SYSADM');
UPDATE PSSTATUS SET OWNERID = 'SYSADM';
```

Note that while it is easy to populate the PSDBOWNER table from SYSADM, this user cannot delete or update those rows because the user only has SELECT privilege, and PS does not even have CONNECT privilege.

If you duplicate a PeopleSoft database by performing a full Oracle export and import, the PS.PSDBOWNER table must exist in the target database before the import is run. Otherwise, the import process will be unable to create the table, because the PS schema does not have RESOURCE privilege any more.

Oracle Database Roles

Two Oracle database roles, PSUSER and PSADMIN, are created during the installation of the PeopleSoft database, as shown in Listing 3-5. These two roles are discussed in the following sections.

Listing 3-5. *An extract from* `PSADMIN.SQL`

```
CREATE ROLE PSUSER;
GRANT CREATE SESSION TO PSUSER;

CREATE ROLE PSADMIN;
GRANT
ANALYZE ANY,
ALTER SESSION, ALTER TABLESPACE, ALTER ROLLBACK SEGMENT,
CREATE CLUSTER, CREATE DATABASE LINK, CREATE PUBLIC DATABASE LINK,
CREATE PUBLIC SYNONYM, CREATE SEQUENCE, CREATE SNAPSHOT,
```

```
CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW,
CREATE PROCEDURE, CREATE TRIGGER, CREATE TABLESPACE, CREATE USER,
CREATE ROLLBACK SEGMENT,
DROP PUBLIC DATABASE LINK, DROP PUBLIC SYNONYM, DROP ROLLBACK SEGMENT,
DROP TABLESPACE, DROP USER, MANAGE TABLESPACE, RESOURCE,
EXP_FULL_DATABASE, IMP_FULL_DATABASE,
GRANT ANY ROLE, ALTER USER, BECOME USER
TO PSADMIN WITH ADMIN OPTION;
```

PSUSER Role

The PSUSER role was introduced in PeopleTools 7, where it was granted to each of the database users that corresponded to PeopleSoft operators. However, in PeopleTools 8 the role is never granted to anyone. Instead, the CREATE SESSION privilege is explicitly granted to the Connect ID (usually PEOPLE), as shown Listing 3-6.

Listing 3-6. *An extract from* `connect.sql`

```
CREATE USER people IDENTIFIED BY people DEFAULT TABLESPACE psdefault
TEMPORARY TABLESPACE pstemp;

GRANT CREATE SESSION to people;
```

PSADMIN Role

The PSADMIN role is granted to the Access ID (usually SYSADM). PeopleSoft describes the privileges defined by this role as the minimum for running PeopleSoft. The Access ID is an administrative account, so there needs to be a degree of trust and cooperation between the DBA and the PeopleSoft administrator (if they are not the same person).

However, not all of the privileges in the PSADMIN role are absolutely essential to the operation of PeopleSoft, and I have come across some sites where the DBAs want to limit the privileges granted by this role even further. Table 3-1 explains the purpose of the various privileges granted to PSADMIN.

Table 3-1. *PSADMIN Role Privileges*

Privilege	Comments	Removable?
ANALYZE ANY	Various batch processes issue analyze commands to update statistics on working storage tables (new in PeopleTools 8).	No
ALTER SESSION	This privilege is required during some SQR processing, and it is also necessary to enable SQL tracing during performance tuning.	No
ALTER TABLESPACE	This is only required if Data Mover creates the tablespace during an import; otherwise PeopleSoft does not alter the tablespaces directly. This is a job for the DBA instead.	Yes

(Continues)

Privilege	Comments	Removable?
ALTER ROLLBACK SEGMENT	In the installation process, the rollback segments are created by the Access ID. However, they should generally be managed by the DBA. From Oracle 9i on, if you use system-managed undo, the question does not arise.	Yes
CREATE CLUSTER	PeopleSoft does not cluster any objects—PeopleTools cannot generate the DDL to create them. It is just conceivable, but highly unlikely, that a DBA might choose to introduce a cluster in the course of performance tuning.	Yes
CREATE DATABASE LINK	Private database links are no longer used. They were used during the upgrade compare process in PeopleTools 7.x and earlier. In PeopleTools 8, this function has been brought into the ApplicationDesigner, which connects directly to the two databases.	Yes
CREATE PUBLIC DATABASE LINK	Public database links are not used by PeopleSoft.	Yes
CREATE PUBLIC SYNONYM	A public synonym is created for PS.PSDBOWNER at installation time (see Listing 3-3), but while connected to database user PS.	Yes
CREATE SEQUENCE	Sequences are not used by PeopleSoft.	Yes
CREATE SNAPSHOT	PeopleSoft does not use snapshots or materialized views.	Yes
CREATE SESSION	This privilege allows connection to the database and is essential.	No
CREATE SYNONYM	This is no longer required. In PeopleTools 6.x and earlier, private synonyms were created for use by the upgrade compare processes.	Yes
CREATE TABLE	Application Designer can create tables online, and Data Mover creates tables during the initial database build. If this privilege is revoked, all DDL generated by Application Designer will have to be executed as another schema, and it would have to be amended to include the owner of the object.	No
CREATE VIEW	Application Designer can create views online, and Data Mover does create views during the initial database build.	No
CREATE PROCEDURE	This privilege is not used by PeopleSoft.	Yes
CREATE TRIGGER	From PeopleTools 8.1 on, triggers are used for auditing. Beginning with PeopleTools 8.4, the Application Designer can also generate triggers used for synchronizing with mobile clients.	No
CREATE TABLESPACE	This privilege is only required if Data Mover creates the tablespace during an import, which the default installation process does not attempt. This is a job for the DBA instead.	Yes
CREATE USER	From PeopleTools 8 on, only three schemas are required. After installation, this privilege is no longer used. It is really left over from PeopleTools 7.x and earlier versions, in which a new database user was created for each PeopleSoft operator ID.	Yes (after installation)
CREATE ROLLBACK SEGMENT	Rollback segments should be maintained by the DBA.	Yes

Privilege	Comments	Removable?
DROP PUBLIC DATABASE LINK	Public database links are not used by PeopleSoft.	Yes
DROP PUBLIC SYNONYM	There is only one public synonym, and it should not be dropped.	Yes
DROP ROLLBACK SEGMENT	The rollback segments should be completely maintained by the DBA.	Yes
DROP TABLESPACE	PeopleSoft never attempts to drop tablespaces. This is a job for the DBA.	Yes
DROP USER	PeopleTools never drops any users. Even in PeopleTools 7.x and earlier versions, when a PeopleSoft operator was deleted, the privileges were revoked from the corresponding schema but the database user not dropped.	Yes
MANAGE TABLESPACE	This privilege allows tablespaces to be taken online and offline during a backup. This is a job for the DBA.	Yes
RESOURCE	This provides a number of system privileges, including CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, and CREATE TRIGGER. It is used for compatibility with previous versions of Oracle. This privilege is not required.	Yes
EXP_FULL_ DATABASE	This provides a range of privileges required to perform full and incremental exports, and the DBA rather than the PeopleSoft system administrator could perform the exports. However, this role also provides SELECT_CATALOG_ROLE, which is required when generating DDL scripts in Application Designer.	Only if replaced with SELECT_ CATALOG_ ROLE
IMP_FULL_ DATABASE	This provides a range of privileges required to perform full imports, and the DBA rather than the PeopleSoft system administrator could perform the imports. However, this role also provides SELECT_CATALOG_ROLE, which is required when generating DDL scripts.	Only if replaced with SELECT_ CATALOG_ ROLE
GRANT ANY ROLE	This privilege is no longer required because schemas are not created when operators are created. This, like any ANY privilege, is dangerous as it opens the door to the entire Oracle database. Since it is no longer necessary in PeopleTools 8, I recommend removing it.	Yes
ALTER USER	This privilege allows the database user to change passwords. It is required when the PeopleSoft administrator changes the password of the Access ID. Up to PeopleTools 7.5, it was also required when an operator password was changed.	No
BECOME USER	This privilege is required by and implicit in IMP_FULL_USER, but it is not otherwise required by PeopleTools.	Yes

Signing On to a PeopleSoft 8 Database

Having covered the available database users, I am now going to step through the processing that a PeopleTools program performs when it makes a two-tier connection to the database. When PeopleSoft Windows client processes, such as the Application Designer, make a three-tier connection, they connect to the application server; each application server process has already

made a two-tier connection to the database when it was started. (The PIA is sometimes referred to as a fourth tier. As described in Chapter 2, the browser connects the servlet, which in turn connects to the application server.)

As discussed earlier in this chapter, any process wishing to connect to the database must identify itself with a PeopleSoft user or operator ID. The purpose of the signon processing is to securely validate that the PeopleSoft operator, authenticated by password, is permitted to access the PeopleSoft database. When a PeopleSoft Windows client process is started, a signon dialog box is presented, as shown in Figure 3-1.

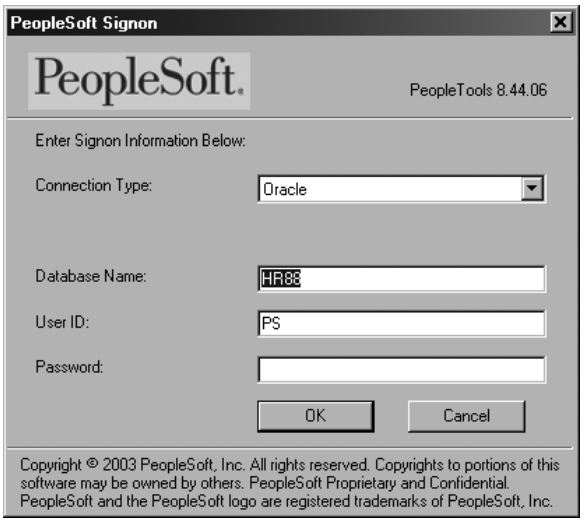


Figure 3-1. *The PeopleSoft signon dialog box for the Windows client*

This signon dialog box can be suppressed if the signon options are supplied as command-line options, as follows:

```
pside.exe -ct ORACLE -cd HR8D -co PS -cp PS
```

The full set of command-line options are specified in PeopleBooks.

To best see what happens when a PeopleTools program makes a two-tier connection to the database, I will walk through a PeopleSoft trace of the process in the following sections.

Making the Initial Connection

As discussed earlier, the nature of the very first connection to the database has changed in PeopleTools 8, compared to version 7.5x. As of version 8, all PeopleSoft processes, irrespective of the PeopleSoft operator, connect to the same low-security database user, referred to as the Connect ID.

For Windows client programs, various settings, including the name and password for the Connect ID user, are set with the Configuration Manager (see Figure 3-2). The database name and user ID are used as default values in the login screen for Windows client programs. This utility also encrypts the Connect ID password.

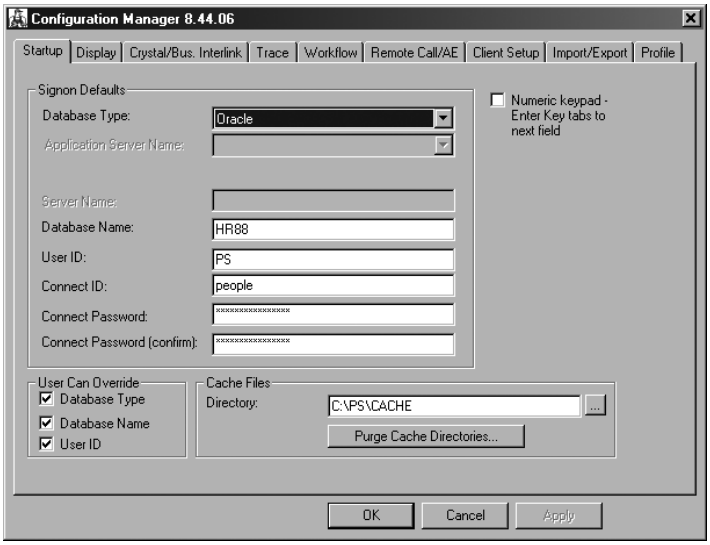


Figure 3-2. *The Configuration Manager for the PeopleSoft Windows client allows you to set the Connect ID password.*

The Configuration Manager is essentially a tool for setting registry values on the client (see Figure 3-3). A Windows client process, such as Application Designer, will read these registry settings.

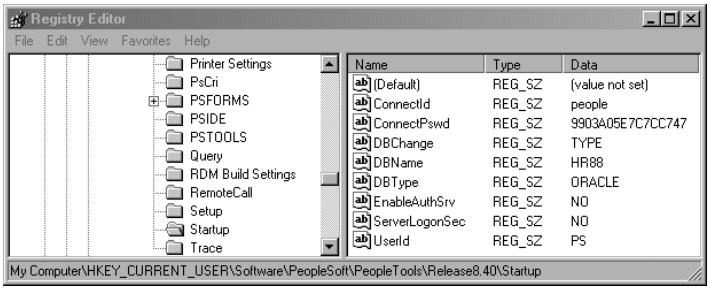


Figure 3-3. *PeopleTools 8.4 registry entries in Windows set by the Configuration Manager*

If you do not use the same Connect ID and password across all your databases, the Windows client processes (mainly Application Designer) will not be able to connect to all the databases in two-tier mode without changing those registry keys. In particular, you will not be able to migrate Application Designer projects directly between databases because Application Designer must log on to both databases in two-tier mode, and it will try to use the same Connect ID for both. The workaround for this problem is to export the project from the source database to a flat file, and then import that file into the target database.

The Configuration Manager is also used to install or remove the PeopleSoft shortcuts in the Windows Start menu, the ODBC driver, and a local installation of the PeopleTools executables.

■ **Note** PeopleSoft places registry settings in both the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE files. Therefore, the configuration for Windows clients needs to be set up separately for each user who needs it, unlike Oracle, which only puts registry entries into HKEY_LOCAL_MACHINE.

The PeopleSoft application server processes, COBOL programs, and Application Engine processes perform similar logon procedures. Their connection details are stored in either the application server configuration file `psappsrv.cfg` or the Process Scheduler configuration file `psprcs.cfg`, as shown in Listing 3-7.

■ **Note** Application server and Process Scheduler processes running on a Windows server do not use the registry settings. Instead, all their settings are read from the configuration files, just as on other operating systems.

Listing 3-7. *An extract from the Process Scheduler configuration file* `psprcs.cfg`

```
[Startup]
;=====================================================================
; Database Signon settings
;=====================================================================
DBName=HR88
DBType=ORACLE
UserId=PS
UserPswd=F204C795327BDFFFB88E3BB0AB1D2BDCB88E3BB0AB1D2BDCB88E3BB0AB1D2BDC
ConnectId=people
ConnectPswd=9903A05E7C7CC747
ServerName=
```

The PeopleTools client trace (discussed in detail in Chapter 9) can show all the SQL operations performed during the signon process. As shown in Listing 3-8, the first action of any PeopleSoft process is to connect to the `ConnectId` with the `ConnectPswd` (both of which are shown in bold in the listing).

Listing 3-8. *The start of a PeopleTools client trace*

```
PID-Line  Time      Elapsed Trace Data...
--------  --------  ------- -------------------->
 1-1      01.54.48          Tuxedo session opened {oprid='PS', appname='TwoTier',
addr='//TwoTier:7000', open at018F6888, pid=3496}
 1-2      01.54.55    6.990 Cur#1.3496.HR88 RC=0 Dur=1.302
Connect=Primary/HR88/people/
```

In Oracle terms, the SQL*Net connect string is `people/peop1e@HR88` (though for obvious reasons the trace does not show the password). The TNS service name in the connect string, `HR88`, is taken from the PeopleSoft database name entered on the signon screen. If an Oracle database

contains more than one PeopleSoft database, there must be a TNS service name for each of the PeopleSoft database names, and they should all point to the same Oracle connection descriptor.

The TNS service must also exist in the default SQL*Net domain, because PeopleSoft will not specify any domain in the connect string. In the example in Listing 3-9, an Oracle database contains two PeopleSoft databases, so the TNS service has two names.

Listing 3-9. *An extract from* `tnsnames.ora`

```
HR88, HR8DREP =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(Key = ora92))
      (ADDRESS = (PROTOCOL = TCP)(HOST = go-faster-3)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = HR88)
      (ORACLE_HOME = d:\oracle\ora92)
    )
  )
```

Though not specified in Oracle's documentation, SQL*Net supports the syntax in the preceding example, where more than one service name can be mapped to a single connection descriptor. However, the Oracle Net Manager utility cannot handle this, generating the warning message in Figure 3-4. You can click OK to get past the error, but the additional service names will be lost if more than one has been defined. The workaround is to create separate service entries with identical connection descriptors.

Figure 3-4. *An Oracle Net Manager error message resulting from mapping more than one service name to a single connection descriptor*

Direct Shared Memory Connections (UseLocalOracleDB)

When a SQL*Net client process is on the same physical server as the database, it can connect directly to the database via shared memory, without using the Oracle Listener or any network protocols. This should provide better performance and consume less CPU time.

This connection through shared memory is sometimes called an IPC (interprocess communication) connection, because on Unix it uses IPC resources and the IPC protocol is specified in the SQL*Net configuration files. In earlier versions of Oracle, the IPC protocol was called the *bequeath* protocol.

If the TNS service name is not appended to the user ID and password in the connect string, SQL*Net will make a shared memory connection. The database instance is specified by the ORACLE_SID environment variable.

If the `UseLocalOracleDB` option in the application server configuration files is set to 1, as shown in Listing 3-10, the SQL*Net connect string is then simply the username and password. PeopleSoft does not append the TNS service name. The `tnsnames.ora` file is not used in this case.

Listing 3-10. *An extract from* `psappsrv.cfg`

```
[Database Options]
;============================================================================
; Database-specific configuration options
;============================================================================

SybasePacketSize=
; Please see Chapter "Tuning and Administration", in
; Oracle Installation and Administration Guide for details
UseLocalOracleDB=1
ORACLE_SID=HR88
```

The Process Scheduler (which initiates batch and report processes and is discussed in more detail in Chapter 14) can be similarly configured, in which case it and any COBOL or Application Engine processes that it spawns will also make a direct shared-memory connection.

PeopleSoft recommends setting `UseLocalOracleDB` to 1 for performance reasons. This way an Oracle Listener is not required in order to make the connection, and therefore the connection to the database should be slightly faster than using an IPC key. However, the improvement is tiny in proportion to the overall response time.

This setting also makes the application server and Process Scheduler sensitive to the value of the `ORACLE_SID` variable in the environment, although this can also be set in the configuration files (`psappsrv.cfg` and `psprcs.cfg`).

I prefer to keep the `UseLocalOracleDB=0` setting and specify an IPC key in the `tnsnames.ora` file, as shown previously in Listing 3-9, and to use a corresponding entry in the `listener.ora` file, as shown in Listing 3-11. Then all SQL*Net clients that run on the database server will make IPC connections.

Listing 3-11. *An extract from* `listener.ora`

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = ORA92))
    )
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = go-faster-3)(PORT = 1521))
    )
  )
```

Occasionally, if a client process connected to the database via an IPC connection terminates without disconnecting from the database, especially while waiting for a response from the database, the Oracle shadow process can continue to run. This situation can happen when

PSAPPSRV or PSQRYSRV processes crash or exceed the Tuxedo timeout while waiting for a long-running SQL query to return. Consider setting `SQLNET.EXPIRE_TIME` in the `sqlnet.ora` file. Although this is not foolproof, it will help to clean up disconnected shadow processes.

Determining the PeopleSoft Schema

Having connected to Oracle, the first action of the signon process is to determine which database schema contains the PeopleSoft database. The next few lines in the PeopleTools client trace (shown in Listing 3-12) show how the PSDBOWNER table is used to map the database to the schema.

Listing 3-12. *An excerpt from a PeopleTools client trace showing the mapping of database to schema*

```
1-3      01.54.56    0.130 Cur#1.3496.HR88 RC=0 Dur=0.000
COM Stmt=SELECT OWNERID FROM PS.PSDBOWNER WHERE DBNAME=:1
 1-4      01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=4 value=HR88
 1-5      01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000 Fetch
```

The schema that contains the PeopleSoft database is referred to as the Access ID. The signon process explicitly references three objects in that schema until it connects directly when the signon process completes successfully. Therefore, SELECT privilege is explicitly granted on those tables during the installation process by the Data Mover script (`hr88ora.dms`), as shown in Listing 3-13. The installation process itself connects to the schema that will contain the PeopleSoft database, so there is no schema name in the `GRANT` commands in the script.

Listing 3-13. *An excerpt from the Data Mover script* `hr88ora.dms`*, which builds a PeopleSoft database*

```
GRANT SELECT ON PSSTATUS TO PEOPLE;
GRANT SELECT ON PSOPRDEFN TO PEOPLE;
GRANT SELECT ON PSACCESSPRFL TO PEOPLE;
```

The PS.PSDBOWNER table allows PeopleSoft to manage multiple PeopleSoft databases in one Oracle database. It contains a row to map each PeopleSoft database name to the Oracle schema that holds it.

It is also possible to give two names to the same PeopleSoft database by adding rows that map to the same OWNERID. There are a few situations where this could be useful, such as these:

- If you are changing the name of a PeopleSoft database, both names can be valid during the transitional period.
- If the production database is copied to a read-only reporting database, you may wish to query that database without making any changes to it.

Either way, for every database name (DBNAME) specified in PS.PSDBOWNER, there must also be a corresponding SQL*Net TNS service in the default SQL*Net domain that maps to the database instance.

Checking the PeopleTools Release

The next job is to ensure that the signon process is attempting to connect to a PeopleSoft database that is at the same PeopleTools version as the client. The structure of the PeopleTools tables has always changed between major PeopleTools releases (8.1 to 8.4). From PeopleTools 8.x on, there can now also be structural changes between minor releases (for example, from 8.43 to 8.44). There may also be new functionality and therefore new objects or messages that are delivered.

The major and minor release numbers are stored in the database in PSSTATUS.TOOLSREL, as shown in Listing 3-14.

Listing 3-14. *An extract from the PSSTATUS table*

```
OWNERID  TOOLSREL           TO_CHAR(LASTREFRESH TO_CHAR(LASTCHANGED
-------- ------------------ ------------------- -------------------
SYSADM   8.44               2004-03-24 19:27:32 2004-03-24 19:27:32
```

The PeopleTools client trace in Listing 3-15 shows the client querying the PeopleTools release from PSSTATUS.

Listing 3-15. *An excerpt from a PeopleTools client trace showing the client getting the version release number*

```
1-6      01.54.56    0.010 Cur#1.3496.HR88 RC=0 Dur=0.010 COM Stmt=SELECT
OWNERID, TOOLSREL, TO_CHAR(LASTREFRESHDTTM,'YYYY-MM-DD HH24:MI:SS'),
TO_CHAR(LASTCHANGEDTTM,'YYYY-MM-DD HH24:MI:SS') FROM SYSADM.PSSTATUS
 1-7      01.54.56    0.040 Cur#1.3496.HR88 RC=0 Dur=0.000 Fetch
```

One step in the process of upgrading PeopleTools from one version to another is to run a "rel" script supplied by PeopleSoft. This script (see Listing 3-16) rebuilds the PeopleTools tables whose structures have changed and also updates the PeopleTools version number in the column TOOLSREL.

Listing 3-16. *An extract from* `rel844.dms`

```
UPDATE PSSTATUS SET TOOLSREL='8.44',
                    LASTREFRESHDTTM = SYSDATE
;
```

This test of the PeopleTools release is not sensitive to the patch level of PeopleTools. Any patch of release 8.44 (for example, 8.44.06) will successfully connect to the database.[1] PeopleSoft usually does not change the structure of PeopleTools in patch releases.

■ **Note** In PeopleTools 7.x this release test was only sensitive to major release number changes because only the major release number was stored in the database.

---

1. The version of the PIA servlet must exactly match that of the Application Server or the connection will fail. An 8.44.06 servlet cannot connect to an 8.44.05 application server. The same applies to Windows client programs, such as the Application Designer or nVision, in three-tier mode.

If the version numbers do not match during the signon process, an error dialog box will appear on the client PC (see Figure 3-5). It indicates that either the installation process has not been completed correctly or that the wrong executables are being used.

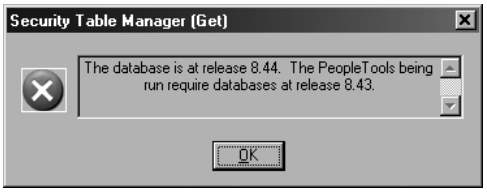


Figure 3-5. *PeopleTools version mismatch error message*

Within the application server, the PeopleTools version test is performed by each PeopleSoft application server process as it starts. The exception to this is the PSWATCHSRV server process that was introduced in PeopleTools 8.43, which detects and kills application server processes that have become zombies (see Chapter 11 for details). This process does not connect to the database at all.

As the application server processes start, they write messages to `appsrv.log`, an example of which is shown in Listing 3-17.

Listing 3-17. *An extract from* `appsrv.log`

```
PSADMIN.2504 [03/29/04 03:10:20](0) Begin boot attempt on domain HR88
PSWATCHSRV.1892 [03/29/04 03:10:26]  Checking process status every 120 seconds
PSWATCHSRV.1892 [03/29/04 03:10:26]  Starting
PSAPPSRV.3368 [03/29/04 03:10:42](0) PeopleTools Release 8.43.12 (WinX86) starting
PSAPPSRV.3368 [03/29/04 03:10:43](1) GenMessageBox(0, 0, M):
Security Table Manager (Get): The database is at release 8.44.
The PeopleTools being run require databases at release 8.43.
PSWATCHSRV.1892 [03/29/04 03:10:44]  Shutting down
PSADMIN.2504 [03/29/04 03:10:50](0) End boot attempt on domain HR88
```

Checking the Operator Password

The next stage in the signon process is to retrieve the PeopleSoft user's encrypted password stored on the operator definition table PSOPRDEFN, as shown in the next part of the trace file in Listing 3-18; the data retrieved is shown in Listing 3-19.

Listing 3-18. *An excerpt from a PeopleTools client trace showing the checking of the operator password*

```
1-8      01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000 COM
Stmt=SELECT VERSION, OPERPSWD, ENCRYPTED, SYMBOLICID, ACCTLOCK
FROM SYSADM.PSOPRDEFN WHERE OPRID = :1
1-9      01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=2 value=PS
 1-10     01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000 Fetch
```

Listing 3-19. *An encrypted operator password on PSOPRDEFN*

```
VERSION OPERPSWD                       ENCRYPTED SYMBOLIC   ACCTLOCK
------- ---------------------------- ---------- -------- ----------
      1 5iCGeTd2aRl/N+E3E8ZUz72qEe4=          1 SYSADM1           0
```

The query in Listing 3-18 shows that PeopleTools verifies the password entered by the operator or provided in the configuration file against the database.

Obtaining the Access Password

The operator's access profile, read from PSOPRDEFN.SYMBOLICID in the previous step, is used to look up the ACCESSID and ACCESSPSWD in PSACCESSPRFL, as shown in Listing 3-20.

Listing 3-20. *An excerpt from a PeopleTools client trace showing the access profile being looked up*

```
1-11     01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000 COM Stmt=SELECT ACCESSID,
ACCESSPSWD, ENCRYPTED FROM SYSADM.PSACCESSPRFL WHERE SYMBOLICID = :1
 1-12     01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=7 value=SYSADM1
 1-13     01.54.56    0.000 Cur#1.3496.HR88 RC=0 Dur=0.000 Fetch
```

The PSACCESSPRFL table existed in PeopleTools 7.5, but it was not checked during the signon process because the encrypted access password was also on PSOPRDEFN. In PeopleTools 8, the access password exists only on PSACCESSPRFL, as seen in Listing 3-21.

Listing 3-21. *Access profile*

```
SYMBOLIC ACCESSID         ACCESSPSWD        ENCRYPTED
-------- ---------------- ---------------- ----------
SYSYADM1 06DE6F1A406C9DAC 06DE6F1A406C9DAC          1
```

■ **Note** In this example, I am using the default Access ID SYSADM, whose password is also SYSADM. In general, though, it is not a good idea to have the same value for ACCESSID and ACCESSPSWD because their encrypted values will also be the same.

Reconnecting As Access ID

Having determined the Access ID schema and its password in the previous step, the process disconnects from the database and reconnects to the schema specified by the Access ID, which is SYSADM in Listing 3-22.

Listing 3-22. *An excerpt from a PeopleTools client trace showing the connection to the schema specified by Access ID*

```
 1-14     01.54.56    0.030 Cur#1.3496.HR88 RC=0 Dur=0.030 Disconnect
 1-15     01.54.56    0.200 Cur#1.3496.HR88 RC=0 Dur=0.100 Connect=Primary/HR88/SYSADM/
```

There are many more checks performed by the application to determine whether the operator is allowed to connect, and to determine how much they can do in either the application or the design tools, but from here on, everything is done with the process connected to the schema that contains the PeopleSoft database.

This can make it difficult to identify which operator is responsible for which piece of SQL, or to use Oracle's resource manager or resource profiles to set different resource usage thresholds for different classes of users. Since PeopleTools 7.53, PeopleSoft has used DBMS_APPLICATION_INFO to help identify operators' sessions (see Chapter 9 for more details).

■ **Note** The name of the Access ID schema does not come from PS.PSDBOWNER.OWNERID, but from PSACCESSPRFL.ACCESS_ID, although they are usually the same. The following sections show what happens when these tables have different values.

Using PeopleSoft Access Profiles and Oracle Resource Profiles

So far, I have shown how PeopleSoft processes all connect to the same Access ID. However, there is a way to make a process connect to the same PeopleSoft database via a different Oracle user. This could permit the use of Oracle resource profiles and other features.

At the beginning of the signon processing, the schema containing the PeopleSoft database was read from the column OWNERID on the table PS.PSDBOWNER. The signon process references PSSTATUS, PSOPRDEFN, and PSACCESSPRFL from that schema. But then, at the end, it connects to the schema specified in ACCESSID on the PSACCESSPRFL table.

During the default PeopleSoft installation process, a Data Mover script imports all the tables into the database and populates PS.PSDBOWNER. It then updates the single row on PSACCESSPRFL with a SYMBOLICID and the name and password of the PeopleSoft schema, as shown in Listing 3-23. All PeopleSoft operators are assigned to that SYMBOLICID. Hence, the value of ACCESSID on PSACCESSPRFL will be the same as OWNERID on PS.PSDBOWNER, and all PeopleSoft processes will connect to that schema, no matter which operator ID they use.

Listing 3-23. *An excerpt from the Data Mover script* `hr88ora.dms`, *which builds a PeopleSoft database*

```
UPDATE PSOPRDEFN SET SYMBOLICID = 'SYSADM1'
, OPERPSWD = OPRID, ENCRYPTED = 0;
UPDATE PSACCESSPRFL SET ACCESSID = 'SYSADM'
, SYMBOLICID = 'SYSADM1', ACCESSPSWD = 'SYSADM'
, VERSION = 0, ENCRYPTED = 0;
```

Having all user processes connected to the database as the same database user effectively prevents the use of Oracle's resource manager and of resource profiles to give different resource usage thresholds for different classes of users. However, it is possible to create another SYMBOLICID in Application Designer, as shown in Figure 3-6. You will have to create the corresponding database user yourself, though. Application Designer will not do it for you.

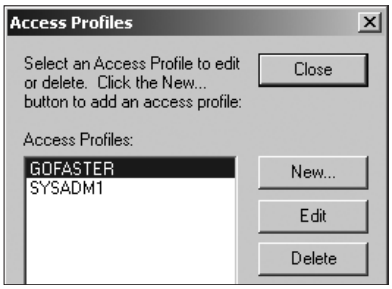


Figure 3-6. *Creating another symbolic ID in Application Designer*

The result is an additional row on PSACCESSPRFL, shown in Listing 3-24. The symbolic ID is specified as part of the user profile, as shown in Figure 3-7. Now when a PeopleSoft process signs in as operator PS, it will ultimately connect to the database as database user GOFASTER.

Listing 3-24. *Two access profiles in the PSACCESSPRFL table*

```
SYMBOLIC    VERSION ACCESSID         ACCESSPSWD        ENCRYPTED
-------- ---------- ---------------- ---------------- ----------
GOFASTER         32 FCB6E5E7B9C9E113 FCB6E5E7B9C9E113          1
SYSADM1         133 06DE6F1A406C9DAC 06DE6F1A406C9DAC          1
```

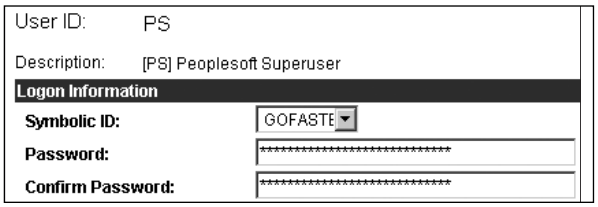


Figure 3-7. *Definiton of a user profile in PIA*

At this point, though, the schema GOFASTER is empty. I only have a single PeopleSoft database in my Oracle database, and it is in the SYSADM schema. So I need to create an after-logon trigger that changes the current schema to SYSADM for users connecting as GOFASTER (see Listing 3-25).

Listing 3-25. `connect.sql`, *an on-connect trigger to set* `CURRENT_SCHEMA`

```
CREATE OR REPLACE TRIGGER gofaster.current_schema_sysadm
AFTER LOGON ON gofaster.schema
BEGIN
   EXECUTE IMMEDIATE 'ALTER SESSION SET CURRENT_SCHEMA = SYSADM';
   EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

Database sessions connecting as GOFASTER will now reference objects in the SYSADM schema by default. All privileges on all tables in SYSADM will also have to be granted to GOFASTER, so that PeopleSoft processes connecting to GOFASTER will execute properly.

So now there is a way to make a particular PeopleSoft operator connect to the database as a particular database user. As an example, I set up an application server that connects with operator PSAPPS, whose SYMBOLICID is SYSADM1, which connects to the database as user SYSADM. I also set up a Process Scheduler that connects with operator PS, whose SYMBOLICID is GOFASTER, which connects to the database as user SYSADM.

This was verified by querying V$SESSION (see Listing 3-26). This proves that it is the ACCESSID that determines the schema to which PeopleSoft connects.

Listing 3-26. *PeopleSoft database sessions*

```
SELECT sid, program, username FROM v$session;

  SID PROGRAM              USERNAME
----- -------------------- -----------------
    7 PSPPMSRV.exe         SYSADM
    8 PSSAMSRV.exe         SYSADM
    9 PSMONITORSRV.exe     SYSADM
   11 PSAPPSRV.exe         SYSADM
   12 PSMSTPRC.exe         GOFASTER
   13 PSDSTSRV.exe         GOFASTER
   14 PSAESRV.exe          GOFASTER
   15 PSAESRV.exe          GOFASTER
   16 PSMONITORSRV.exe     GOFASTER
   18 PSPRCSRV.exe         GOFASTER
```

It is possible to have different Oracle users in operation within your PeopleSoft application, but there are some additional considerations:

- This technique can not be used for the Application Designer or Data Mover tools because they reference USER data dictionary views, such as USER_TABLES, to determine what database objects exist before creating them. This technique would cause them to do this while signed into a user that does not own the tables.
- In PeopleTools 8, only Query or nVision Windows clients make two-tier runtime connections to the database. Otherwise all connections will be via the application server or Process Scheduler. You can only set the operator ID for the whole of a particular application server or Process Scheduler, not just for some of the server processes. This means that all of the operators who connect via an application server will connect to the Access ID of the SYMBOLICID of the operator ID used to start the application server, and not the Access ID of their own SYMBOLICID.
- This technique should not be used to switch between different PeopleSoft databases in the same Oracle database (which you should not be creating anyway) as it would only lead to confusion.

However, the ability to redirect the final database connection to a different database user raises some tantalizing possibilities. I have had only limited opportunity to experiment with this feature, but here are some theoretical possibilities:

- If you use Query or nVision Windows clients in three-tier mode, they could be directed to a different application server with a particular operator ID. An Oracle resource profile could be applied to that database user to limit online report activity.
- An Oracle resource profile could be applied to all batch nVision and Crystal reporting by permitting them to run only on a particular Process Scheduler started with a particular PeopleSoft operator ID.
- The Global Payroll calculation process can sometimes err if HR changes are made while it is running. Tables that feed into the payroll process could be replicated to an alternative schema, and the calculation process could be run on a Process Scheduler connected to that schema. Instead of changing the session schema, synonyms would have to be created for other tables and views, although some views may also have to be created in the archive schema. The payroll would read the replicated table but would otherwise access tables in the live schema.
- Data could be archived to tables in a different schema and viewed via an application server connected to the archive schema.

■ **Caution** PeopleSoft does not support this technique of using multiple access profiles.

Signing On to a PeopleSoft 7.5 Database

In PeopleTools 7.x, the signon process was slightly different. Listings 3-27 through 3-31 show a successful two-tier PeopleTools 7.53 client trace in order to highlight the changes that were made in PeopleTools 8. These listings show the operator PS logging onto the database H75D.

First of all, as noted earlier, there was a database schema for every PeopleSoft operator in PeopleTools 7.x. The first line in the trace shows the client connecting to the schema specified in the signon dialog box. The connect string is `ps/ps@h75d`.

Listing 3-27. *An excerpt from a PeopleTools 7.5 client trace showing the client connecting to the schema*

```
1-1   Cur#1 RC=0 Dur=3.235 Connect=H75D/PS/
```

If an Oracle database contained multiple PeopleSoft 7.x (or earlier) databases, and if the same operator ID existed on more than one of those PeopleSoft databases, they had the same password across all the PeopleSoft databases. If an operator changed their own password on one PeopleSoft database, they would not be able to connect to another PeopleSoft database in two-tier mode, nor in three-tier if "Validate Signon with Database" was enabled in the application server configuration.

In Listing 3-28, the trace shows a call to a database procedure `SQLCQR_LOGINCHECK`. PeopleSoft added this to provide an interface to the Braintree SQL<>Secure product, which validated

passwords and detected break-in attempts. If the procedure were not present, the error produced by trying to call it was ignored. However, sometimes this caused spurious error messages. If a PeopleSoft COBOL program failed for a non-database reason, the Oracle error message structure would still be reported and it would return the error relating to the SQLCQR_LOGINCHECK during signon.

Listing 3-28. *An excerpt from a PeopleTools 7.5 client trace showing* SQLCQR_LOGINCHECK *being called*

```
1-2   0.000 Cur#1 RC=0 Dur=0.000 COM Stmt=EXECUTE :1 := SQLCQR_LOGINCHECK(:2)
1-3   0.000 Cur#1 RC=0 Dur=0.000 Bind-1 type=18 length=2 value=0
1-4   0.000 Cur#1 RC=0 Dur=0.000 Bind-2 type=2 length=254 value=
```

The signon then looks up the schema that contains the PeopleSoft database, as shown in Listing 3-29.

Listing 3-29. *An excerpt from a PeopleTools 7.5 client trace showing the database schema lookup*

```
1-5   1.993 Cur#1 RC=0 Dur=0.010 COM Stmt=SELECT OWNERID FROM PS.PSDBOWNER WHERE DBNAME = :1
1-6   0.000 Cur#1 RC=0 Dur=0.000 Bind-1 type=2 length=4 value=H75D
```

The next step was to check the PeopleTools version. The PeopleTools version was stored on the one-row PSLOCK table in PeopleTools 7.x. The table was also used to store the version numbers for PeopleSoft objects that controlled their caching on the client and application server. There was a column on the table for every version number. Listing 3-30 shows this information being retrieved.

In PeopleTools 8.x, the version numbers are still on PSLOCK, but now there is one row per version number. The PeopleTools version and the timestamp columns have moved to the PSSTATUS table.

Listing 3-30. *An excerpt from a PeopleTools 7.5 client trace showing version and timestamp information being retrieved*

```
1-8   0.000 Cur#1 RC=0 Dur=0.000 COM Stmt=SELECT OWNERID, TOOLSREL,
TO_CHAR(LASTREFRESHDTTM,'YYYY-MM-DD HH24:MI:SS'),
TO_CHAR(LASTCHANGEDTTM,'YYYY-MM-DD HH24:MI:SS'), SECURITY_OPTION
FROM SYSADM.PSLOCK
1-9   1.433 Cur#1 RC=0 Dur=1.433 Fetch
```

The Access ID password, the password to the database schema, was taken directly from PSOPRDEFN in PeopleTools 7.x, as shown in Listing 3-31. The same password had to be stored on every row in the operator definition table.

Listing 3-31. *An excerpt from a PeopleTools 7.5 client trace showing the Access ID password being retrieved*

```
1-10   0.000 Cur#1 RC=0 Dur=0.000 COM Stmt=SELECT VERSION, OPRTYPE, OPERPSWD, ACCESSID,
ACCESSPSWD FROM SYSADM.PSOPRDEFN WHERE OPRID = :1
1-11   0.000 Cur#1 RC=0 Dur=0.000 Bind-1 type=2 length=2 value=PS
0.490 Cur#1 RC=0 Dur=0.490 Fetch
```

This structure created some security weaknesses. Every user knew a user ID and password with which they could log on to the database directly. The user could then query all the rows on PSOPRDEFN and might be able to find rows where the passwords were not encrypted. If the Access ID password was changed, all the rows on PSOPRDEFN had to be updated and reencrypted. This encryption process was not completely stable.

Connecting Third-Party Applications

PeopleSoft also supplies reporting and batch functionality via two third-party reporting packages (Crystal Reports and SQR), as well as through its own COBOL and Application Engine programs.

Crystal Reports

PeopleSoft ships a standard, but rebranded, version of Crystal Reports. It is closely bound to the PeopleSoft Query tool, and will connect via the PeopleSoft ODBC driver. This provides logon and row-level security. The ODBC driver presents the queries, which are defined through the application, as stored procedures.

SQR Reports

SQR, originally from SQL Solutions, but now licensed by PeopleSoft by Hyperion, simply connects directly to the PeopleSoft schema (usually SYSADM) just like SQL*Plus or any other two-tier client. SQR is run via a PeopleTools process named pssqr. This process is a wrapper for SQR that passes other parameters to SQR to control report formatting.

In PeopleTools 7.x, the Process Scheduler supplied the user ID and password directly to SQR, via a short-lived file that could only be read by the Unix user who owned the Process Scheduler process.

Changing Database Passwords

It is good practice on any system to regularly change passwords. However, as the description of the PeopleSoft signon processing shows, encrypted passwords are stored by PeopleTools in ordinary tables. If these values are not synchronized with actual schema passwords, your application will fail to connect to the database.

Connect ID Password

Connect ID passwords are stored in two places:

- Windows registry: The password can be set and encrypted with the PeopleSoft configuration manager. The registry must be set on every PC that makes a two-tier connection to the database.
- Application Server and Process Scheduler configuration files: The password can be set by editing the configuration files directly, but only the interactive configuration dialog box will encrypt the Connect password (see Chapter 12).

The Connect ID is not stored in the database. The main challenge of changing the Connect ID password is to synchronize the PeopleSoft configuration change to the change of the database password. Realistically, it requires system downtime. However, bear in mind that it is a low-security account.

Owner ID Password

In PeopleTools 8, the password to the Owner ID schema should be changed with the `CHANGE_ACCESS_PASSWORD` command in Data Mover, as shown in Listing 3-32.

Listing 3-32. *Data Mover command to change access password*

```
CHANGE_ACCESS_PASSWORD <SymbolicID> <newAccessPswd>
```

This updates both the schema password and the encrypted value on PSACCESSPRFL. This password can be changed without system downtime.

If, for some reason, you cannot set the password in Data Mover, it is also possible, but not recommended, to set the Access ID and password on PSACCESSPRFL to the unencrypted values, as shown in Listing 3-33.

Listing 3-33. *SQL to manually set Access ID and password*

```
UPDATE psaccessprfl
SET    accessid = 'SYSADM'
,      accesspswd = 'SYSADM'
,      encrypted = 0
WHERE  symbolicid = 'SYSADM1'
/
```

The next PeopleSoft process that connects to the database will reencrypt them, but until then they are unencrypted and visible to the Connect ID.

Summary

PeopleSoft and Oracle use the term *database* differently. A PeopleSoft database exists in a schema within an Oracle database. That schema has certain privileges that permit PeopleSoft to function.

The signon processing securely authenticates an operator before permitting them to connect to the PeopleSoft database, but without using any database-specific security features and without revealing database passwords to the user.

CHAPTER 4

PeopleSoft Database Structure: A Tale of Two Data Dictionaries

One of the ways that PeopleSoft is able to deliver the same application to any platform is by having its own data dictionary. Different database platforms structure their data dictionaries differently, and platform independence is an overriding principle in PeopleSoft. Except when building DDL scripts and performing audits, PeopleTools never interrogates the database's catalogue, but instead relies upon its own. PeopleTools is also a development environment, so it is necessary to have a way of defining a table before it is actually created in the database—the PeopleSoft data dictionary serves this function.

As introduced briefly in Chapter 1, a PeopleSoft database on any platform is usually described as having the following three sections, which are shown in Figure 4-1:

- The database catalogue, which is sometimes referred to as the data dictionary, describes all the objects in the database. The structure of the database's catalogue varies from platform to platform, but the concept is common to all. In Oracle, the catalogue is exposed through the underlying fixed objects (tab$, col$, etc.) and is visible in a more user-friendly format through views (DBA_TABLES, DBA_TAB_COLUMNS, etc.)
- The PeopleTools tables contain most of the source code for the PeopleSoft application. This includes certain tables that hold information about the structure of all the PeopleTools and application tables and their indexes. This subset is PeopleSoft's data dictionary.
- Application tables contain the users' data. The PeopleTools tables and the Application tables all exist in the same database schema.

Note Oracle refers to its data dictionary as the "catalogue." When you create a database, you build the catalogue with `catalog.sql`. To avoid confusion, I shall refer to the Oracle *catalogue* and the PeopleSoft *data dictionary*.

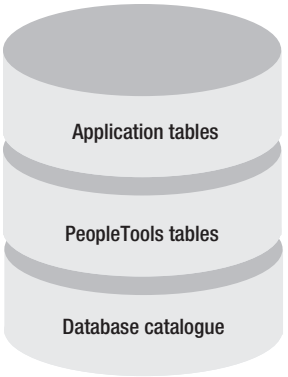


Figure 4-1. *The three sections of a PeopleSoft database*

APPLICATION AND PEOPLETOOLS TABLES

Because application and PeopleTools tables are mixed up together in the same database schema, DBAs and developers often want to know which is which.

In general, PeopleTools record names begin with "PS", and the SQLTABLENAME is explicitly set in PSRECDEFN to be the same as the record name. PeopleTools records can be of any type.

Application tables generally do not have their names overridden, the record name does not begin with "PS", and so the table name is the record name prefixed with "PS_".

However, there are some tables that do not follow this standard, and some tables are not quite PeopleTools tables and not quite application tables either. Table 4-1 shows some examples of different table names.

Table 4-1. *Application and PeopleTools Tables*

Record Name	Table Name	Comment
JOB	PS_JOB	Typical application data table.
JOB_VW	PS_JOB_VW	Typical application view.
PSRECDEFN	PSRECDEFN	Typical PeopleTools table.
PSRECFLDKEYSVW	PSRECFLDKEYSVW	Typical PeopleTools views.
PRCSDEFN	PS_PRCSDEFN	This is actually a PeopleTools table. It defines the processes that can be run on the Process Scheduler. It is one of seven delivered records where the SQLTABLENAME has been explicitly set to the name the table would have been given anyway.
XLATTABLE_VW	XLATTABLE_VW	This view is also a PeopleTools object, but the record name does not start with PS. It is a view of PSXLATITEM and PSXLATDEFN, which hold translate values used in drop-down lists in the PIA.

This chapter discusses the function and interrelationship of the Oracle and PeopleSoft dictionaries, focusing on how the PeopleTools tables specify the data model to PeopleSoft. Chapter 5 looks at how indexes are defined, and Chapter 6 explains how the Application Designer defines physical storage parameters and then combines all this information to generate the SQL to create the tables and indexes.

Two Data Dictionaries

In this section I will examine some of the PeopleTools tables that correspond to the Oracle catalogue. It is useful for both the DBA and developer to be familiar with these tables. While the Oracle catalogue tells you what actually exists in the database, the PeopleSoft data dictionary tells you what should be there. Table 4-2 lists the principal PeopleTools tables that correspond to the Oracle catalogue.

Table 4-2. *Corresponding PeopleTools and Oracle Catalogue Tables*

Table Description	Oracle Catalogue View	PeopleTools Table (version 7.x or earlier)	PeopleTools Table (from version 8)
1 row per table or view	DBA_TABLES DBA_VIEWS	PSRECDEFN	PSRECDEFN and PSRECTBLSPC
1 row per column	DBA_TAB_COLUMNS	PSRECFIELD	PSRECFIELDDB and PSRECFIELD
1 row per distinct column name	n/a	PSDBFIELD	
1 row per view	DBA_VIEWS	PSVIEWTEXT	PSSQLDEFN and PSSQLTEXTDEFN
1 row per index	DBA_INDEXES	PSINDEXDEFN	
1 row per index column	DBA_IND_COLUMNS	PSKEYDEFN	
1 row per user	DBA_USERS	PSOPRDEFN	

PeopleSoft does not create or use database privileges, synonyms, or referential constraints. All the PeopleTools and application tables are in the same database schema. All object- and row-level security issues are handled in the application. Only UNIQUE and NOT NULL constraints are enforced by the database. There are also length-checking constraints on character columns in a Unicode database. All other validation is handled by the application.

While there is good PeopleSoft documentation and training on how to develop and customize a PeopleSoft application, PeopleSoft does not officially explain the meaning and use of the PeopleTools tables. Their structure and meaning can change from release to release, as it has in the past. However, by investigating the contents of the PeopleTools tables, by trying to relate settings within the Application Designer to values on the PeopleTools tables, and by investigating the PeopleTools recursive SQL, it is possible to discover some of the meaning of these tables.

■ **Caution** One of the reasons that PeopleSoft does not provide detailed documentation for the PeopleTools tables is to discourage people from updating them directly. While these tables are a valuable source of information about a PeopleSoft system, you are cautioned not to alter them yourself. If you do, you are on your own.

The following sections describe these PeopleTools tables in more detail. Only some of the columns on these tables correspond to columns in the Oracle catalogue views. The other columns control the behavior of the PeopleSoft application; they are shown for completeness but are not described in this chapter. The primary key columns of the PeopleTools tables have been indicated with a key icon.

PSRECDEFN: Record Definition

This table contains one row for each PeopleSoft record, and it is described in Table 4-4. One of the attributes of a record set in the Application Designer (see Figure 4-2) is the record type. A record can correspond to a table or view, depending upon the record type, as set out in Table 4-3, although some record types do not correspond to any database object.

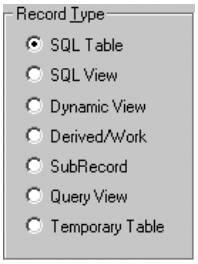


Figure 4-2. *Record types in the Application Designer*

Table 4-3. *Record Types in PeopleTools*

Record Type	Type Description	Comment
0	SQL Table	The record will be built as a table and there will be a corresponding row on DBA_TABLES.
1	SQL View	The record will be built as a view and there will be a corresponding row on DBA_VIEWS.
2	Derived/Work	This provides working storage in the component. It does not correspond to any database object.
3	SubRecord	This is a group of columns or other sub-records that is referenced by other records. It does not correspond to a database object.
4		Not used.
5	Dynamic View	This does not correspond to any database object. A database view is a query executed by the database; this is a query executed by PeopleTools to populate a page.

Record Type	Type Description	Comment
6	Query View	The record will also be built as a database view. The difference between this and record type 2 is that the query is defined via the PeopleTools Query utility, rather than with free format text in the Application Designer.
7	Temporary Table	This record type is new in PeopleTools 8. One temporary table record can correspond to a number of database tables. They are used by the Application Engine to temporarily hold data. If multiple instances of the same Application Engine process execute simultaneously, each instance can use a different temporary table. The number of instances of the table is defined both globally and on the Application Engine program. If, for example, there are 10 instances of record XXX, there will be 11 tables, PS_XXX and PS_XXX_1 through PS_XXX_10.

The PSRECDEFN table contains one row per PeopleTools record. The key icon indicates unique key columns on the PeopleTools tables.

Table 4-4. *Records vs. Tables and Views*

PeopleTools PSRECDEFN	Oracle DBA_TABLES	Description of PeopleTools Column
RECNAME		Unique name of record within PeopleSoft (see SQLTABLENAME below).
FIELDCOUNT		Number of fields on a record before expanding any sub-records; this does not therefore equate to the number of columns on a table.
INDEXCOUNT		Number of keys on a record, including those implied by the PeopleTools record key definition. Note that views can also have PeopleTools keys, because PeopleSoft will query a record by its primary key, although it is not possible to build an index on a view.
DDLCOUNT		Number of DDL override parameters specified.
VERSION		Internal PeopleTools version for controlling caching of object.
AUDITRECNAME		Name of audit record. The PIA will write audit information about this record to the audit record.
RECUSE		Audited actions (can be combined): 1 = Add 2 = Change 4 = Delete 8 = Selective
RECTYPE		See Table 4-3.
SETCNTRLFLD		Multi-company row-level security feature.
RELLANGRECNAME		Related language record that contains translated values.

(Continues)

PeopleTools PSRECDEFN	Oracle DBA_TABLES	Description of PeopleTools Column
RECDESCR		Short description of record.
OPTDELRECNAME		Optimization delete record (new in PeopleTools 8.4).
PARENTRECNAME		Name of parent record for use in Query utility.
QRYSECRECNAME		Name of query security record.
DDLSPACENAME	TABLESPACE_NAME	In PeopleTools 8.1, this column specifies the tablespace in which the table is to be built. In PeopleTools 8.4, the tablespace is stored on PSRECTBLSPC instead (see Chapter 6), and this column has been dropped.
SQLTABLENAME	TABLE_NAME or VIEW_NAME	Name of any table or view that is built in the database. If this column is blank, the table name will be "PS_" followed by the RECNAME. `DBA_TABLES.TABLE_NAME = DECODE(PSRECDEFN.SQLTABLENAME ,' ','PS_'||PSRECDEFN.RECNAME ,PSRECDEFN.SQLTABLENAME)`
BUILDSEQNO		Controls the order in which objects are built by the Application Designer. For instance, a view must be built before another view that references it, so the referencing view will have a higher build sequence number than the referenced view.
OPTTRIGFLAG		Set to "Y" if a trigger is to be built to maintain the system ID field (new in PeopleTools 8.4).
OBJECTOWNERID		Short code indicating the functional area that owns the record. There are over 250 codes, and the translations are on PSXLATITEM. For example: PPT: PeopleTools HCR: HR Core Objects HG: Global Payroll Core Application
LASTUPDDTTM		Timestamp of the last update in Application Designer.
LASTUPDOPRID		Operator (developer) who last updated this record.
SYSTEMIDFIELDNAME		Field on record to be updated by database trigger with sequence number obtained from record PSSYSTEMID (new in PeopleTools 8.4).
TIMESTAMPFIELDNAME		Field on record to be updated by database trigger with a timestamp (new in PeopleTools 8.4).
AUXFLAGMASK		Auxiliary flag mask (new in PeopleTools 8.4).
DESCRLONG		Long description of record.

PSRECFIELD: Record Field Definition

This table contains one row for each field, or sub-record, in each record. In PeopleSoft, a *sub-record* is a group of fields or other sub-records. For example, Figure 4-3 shows the NAMES record from HCM. It includes a sub-record called NAMEGBL_SBR.

In PeopleSoft, a sub-record defines a group of fields. It is used in many records, or even in other sub-records, so that the same group of fields is included in each record. For example, NAMEGBL_SBR contains all the fields required to hold a name.

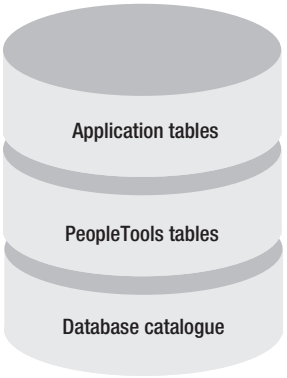


Figure 4-3. *The NAMES record with expanded sub-record*

Where a PeopleSoft record corresponds to a table or view in the database, each field in the record corresponds to a column in that object, as shown in Table 4-5.

Table 4-5. *Fields vs. Columns*

PeopleTools PSRECFIELD	Oracle DBA_TAB_COLUMNS	Description of PeopleTools Column
RECNAME	TABLE_NAME	Not an exact match (see PSRECDEFN.SQLTABLENAME).
FIELDNAME	COLUMN_NAME	Column or sub-record name.
FIELDNUM	COLUMN_ID	Controls order of field in record or sub-record.
DEFRECNAME		Default value from DEFFIELDNAME on record DEFRECNAME.
DEFFIELDNAME		Default value from DEFFIELDNAME on record DEFRECNAME.
CURCTLFIELDNAME		

(Continues)

PeopleTools PSRECFIELD	Oracle DBA_TAB_COLUMNS	Description of PeopleTools Column
EDITTABLE		Prompt table: a look-up table to provide a list of values and to validate entry.
USEEDIT		See Table 4-6. Contains 0 if the column is a sub-record.
USEEDIT2		New in PeopleTools 8.4.
SUBRECORD		Y/N: Is this column is a sub-record?
SUBRECORDVER		
SETCNTRLFLD		
DEFGUICONTROL		
LABEL_ID		
LASTUPDDTTM		Timestamp of last update in Application Designer.
LASTUPDOPRID		Operator (developer) who last updated this record.
TIMEZONEUSE		
TIMEZONEFIELDNAME		
RELTMDTFIELDNAME		
CURRCTLUSE		

The USEEDIT Field

The USEEDIT field on PSRECFIELD is an integer value. When converted to a binary number, each digit defines whether a certain attribute in PeopleTools is enabled for the field. Table 4-6 describes the meanings of these bits.

Table 4-6. *USEEDIT Bit Values*

Bit Number	USEEDIT Bit Value	Set (Not Set) Value	Comment
0	1	Key value	This column will be a part of the PeopleSoft key index (PS_<recname>).
1	2	Duplicate (unique) key	The PeopleSoft key index will be NONUNIQUE if any one field in the key is given duplicate attribute.
2	4	System maintained field	
3	8	Audit field—add	
4	16	Alternate search key	This field is not part of the primary key but can be used to query the data in a record locator. The field will appear in the alternate search key indexes.

Bit Number	USEEDIT Bit Value	Set (Not Set) Value	Comment
5	32	List box item	This column will appear in the result set of component record locator dialog or list of values. Up to PeopleTools 7.x, an index was created on all list columns.
6	64	Ascending (descending) key field	Controls order of the column in the ORDER BY clauses of SQL generated by the PIA. Up to PeopleTools 8.14, the indexes were also built with the order specified on the field.
7	128	Audit field—change	
8	256	Required field	Date and Long columns will have NOT NULL constraints. The PeopleSoft application will not allow 0 for Number columns or blank character fields.
9	512	X/Lat	Look-up value for field in table XLATTABLE.
10	1024	Audit field—delete	
13	8192	Y/N	Field will only accept a Y/N response that is mapped to specified database values.
14	16384	Table edit enabled	
18	262144	From search field	
19	514288	To search field	
21	2097152	Disable advanced search options	
23	8388608	Regular field (sub-record)	Not a sub-record.
24	16777216	Default search field	

PSRECFIELDDB: Expanded Record Field Definition

One of the difficulties of working with PSRECFIELD is that sub-records are not expanded, and in some places PeopleSoft delivers sub-records nested within sub-records. PSRECFIELDDB was introduced in PeopleTools 8 and contains the fully expanded record definitions. In PeopleTools 8 it was populated by an SQR, but in 8.4 it is maintained by Application Designer. Thus, this table is a much closer match for USER_TAB_COLUMNS.

As a result of the sub-record expansion, the field numbers (FIELDNUM) are not the same as in PSRECFIELD when a record contains a sub-record. They may not be the same as the COLUMN_ID on USER_TAB_COLUMNS either, because PeopleSoft moves long columns to be the last column in the table definition.

As you will see later in this chapter, PeopleSoft does not use PSRECFIELDDB during run time; it is maintained as a reporting table by the development tools.

The structure of PSRECFIELDDB is identical to PSRECFIELD except it has one extra field: RECNAME_PARENT. This column will usually contain the same value as RECNAME unless the field is defined via a sub-record, in which case it will contain the name of the sub-record.

PSDBFIELD: Field Definition

This table (see Table 4-7) contains one row for each distinct field name. As well as containing some descriptive and display information, it defines the data type of the fields. It doesn't correspond directly to any object in the Oracle catalogue, but when joined with PSRECFIELD, it provides additional information about how a column should be defined.

Each record in this table indicates how, in PeopleTools, fields with the same names are always created with the same attributes, regardless of which record or table they are created on. There is a single definition for each field in PeopleSoft that is held on a single row in PSDBFIELD.

Table 4-7. *Data Type Definitions*

PeopleTools PSDBFIELD	Oracle (DBA_TAB_COLUMNS)	Description of PeopleTools Column
FIELDNAME	COLUMN_NAME	Column or sub-record name.
VERSION		Internal PeopleTools version to control caching of object.
FIELDTYPE		See Table 4-8.
LENGTH	DATA_LENGTH or DATA_PRECISION	See comments in Table 4-8.
DECIMALPOS	DATA_SCALE	Only for FIELDTYPE 2 (Number) and FIELDTYPE 3 (Signed Number); see Table 4-8.
FORMAT		
FORMATLENGTH		
IMAGE_FMT		
FORMATFAMILY		
DISPFMTNAME		
DEFCNTRYYR		
IMEMODE		
KBLAYOUT		
OBJECTOWNERID		Short code indicating the functional area that owns the field (see PSRECDEFN.OBJECTOWNERID).
LASTUPDDTTM		Timestamp of the last update in Application Designer.
LASTUPDOPRID		Operator (developer) who last updated this record.
FLDNOTUSED		New in PeopleTools 8.4.
AUXFLAGMASK		New in PeopleTools 8.4.
DESCRLONG		Long description of record.

Field Types and Lengths

The FIELDTYPE column on PSDBFIELD specifies the data type that the field will have, and hence the data type of the column when the record is built. The field types are set out in Table 4-8. Some

PeopleSoft field types will correspond to more than one Oracle data type, depending upon the length and precision of the field. The LENGTH column on PSDBFIELD corresponds differently with the columns DATA_LENGTH and DATA_PRECISION on USER_TAB_COLUMNS, depending on the PeopleTools FIELDTYPE.

All character, numeric, and required fields in PeopleSoft will have a NOT NULL constraint.

Table 4-8. *Field Types and Column Definitions*

Value of FIELDTYPE	PeopleSoft Data Type	Oracle Data Type	Comment
0	Character	VARCHAR2	Maximum length 255 characters imposed by PeopleTools
1	Long Character	VARCHAR2	If LENGTH is between 1 and 2000
		LONG	Up to PeopleTools 8.1, if LENGTH = 0 or LENGTH > 2000
		LONG VARCHAR	From PeopleTools 8.4 on, if LENGTH = 0 or LENGTH > 2000
2	Number	DECIMAL (precision,scale)	If DECIMALPOS > 0 Where precision = LENGTH–1 and scale = DECIMALPOS
		SMALLINT	If DECIMALPOS = 0 and LENGTH <= 4
		INTEGER	If DECIMALPOS = 0 and LENGTH > 4
3	Signed Number	DECIMAL (precision,scale)	If DECIMALPOS > 0 Where precision = LENGTH–2 and scale = DECIMALPOS
		NUMBER	If DECIMALPOS = 0
4	Date	DATE	LENGTH = 10
5	Time	DATE	LENGTH = 15
6	DateTime	DATE	LENGTH = 26
8	Image	LONG RAW	
9	ImageRef	VARCHAR2(30)	LENGTH = 30 Contains a reference to a file

Unicode

PeopleSoft introduced support for Unicode in PeopleTools 8.1. There are some interesting implications for the way that columns are specified on tables created by PeopleTools.

On a database with a non-Unicode character set, character columns between 1 and 1999 characters in length have the same length as the field in PeopleSoft. EMPLID is an 11-character field in PeopleSoft, so it will be built as an 11-byte field in Oracle as shown in Listing 4-1.

Listing 4-1. *Creating the PS_JOB table in a database with a single-byte codepage*

```
CREATE TABLE PS_JOB
(EMPLID VARCHAR2(11) NOT NULL,...
```

However, if the database uses Unicode (indicated to PeopleTools by PSSTATUS.UNICODE_ENABLED=1)[1] the Application Designer builds character columns three times as long, to allow for the possibility of 3-byte Unicode characters, and then adds a constraint to enforce the original length in characters, as shown in Listing 4-2.

Listing 4-2. *Creating the PS_JOB table in a unicode database*

```
CREATE TABLE PS_JOB
(EMPLID VARCHAR2(33) NOT NULL CHECK(LENGTH(EMPLID)<=11),...
```

The result is something like 165,000 length-checking constraints in a vanilla HR database, and over 500,000 in a Financials database. There are two problems with this approach:

- The constraints have to be loaded into the library cache with the rest of the information about the table, with additional recursive SQL at parse time. Tests have indicated up to four times as much time is spent parsing, which consumes CPU resources on the database server.
- The constraints are not explicitly named and so are given system-generated names. It is possible that an Oracle import process could duplicate the constraints without warning.

From Oracle 9 on, it would be possible to define the column in terms of characters and omit the constraint, as shown in Listing 4-3. Unfortunately, PeopleSoft does not do this.

Listing 4-3. *New column definition syntax in Oracle 9i*

```
CREATE TABLE PS_JOB
(EMPLID VARCHAR2(11 CHAR) NOT NULL,...
```

If you create a long character field in Application Designer of between 1,334 and 1,999 characters in length, it will be always be built in Oracle as a 4,000 byte column with a maximum length of 1,333 characters, as shown in Listing 4-7. The maximum length for a VARCHAR2 column from Oracle 8 onwards is 4,000 bytes. The PeopleSoft application will permit you to enter more than 1,333 characters in the field without warning or error. Only when PeopleSoft attempts to save the data to the database will the resulting SQL INSERT or UPDATE statement then error.

EFFECTS OF LENGTH-CHECKING CONSTRAINTS ON PARSE TIME

To illustrate the magnitude of the effect of adding length-checking constraints to character columns in a Unicode database, I have constructed the following simple test.

1. Create two tables with 20 columns of type VARCHAR2, identical except that only one should have the length-checking constraints on the character columns.

1. If you want to see this yourself, you can temporarily set this parameter on a test database. Even if your Oracle database is not in Unicode, PeopleSoft will build the DDL for Unicode.

Listing 4-4. *SQL code for creating the two tables*

Without constraints

```
create table test_nocons
(idf     number
,field_01 varchar2(30)
...
,field_20 varchar2(30)
) STORAGE(INITIAL 256K);
```

Constraints

```
create table test_cons
(id          number
,field_01 varchar2(30) CHECK(LENGTH(field_01)<=30)
...
,field_20 varchar2(30) CHECK(LENGTH(field_20)<=30)
) STORAGE(INITIAL 256K);
```

2. Run the first test. It will insert the same data into each table with a PL/SQL loop, as shown in Listing 4-5. The statement in the loop has bind variables, so it will only be parsed once.

Listing 4-5. *Populate the tables (*`INSERT` *statements use bind variables)*

Without constraints

```
BEGIN
  FOR i IN 1..10000 LOOP
    INSERT INTO test_cons
    VALUES(i
          ,RPAD(TO_CHAR(i),11,'.')
...
          ,RPAD(TO_CHAR(i),30,'.')
           );
  END LOOP;
  COMMIT;
END;
/
```

Constraints

```
BEGIN
  FOR i IN 1..10000 LOOP
    INSERT INTO test_nocons
    VALUES(i
          ,RPAD(TO_CHAR(i),11,'.')
...
          ,RPAD(TO_CHAR(i),30,'.')
           );
  END LOOP;
  COMMIT;
END;
/
```

The results are shown in Table 4-9.

Table 4-9. *Results of the First Test*[2]

Results	No Constraints	With Constraints
Oracle 8.1.7.4	21.41s	21.80s
Oracle 9.2.0.1	1.05s	1.07s

The constraints only add a small overhead to the execution time of the SQL. Therefore, the overhead of actually executing the constraints is small.

3. Run the second test. It inserts exactly the same data, but the SQL statement is generated with literal values and so has to be reparsed every time. The test code is shown in Listing 4-6.

(Continues)

2. The test results were produced on 1.4 GHz laptop running Windows XP.

Listing 4-6. *Populate the tables (literal values in SQL force parsing)*

Without constraints

```
BEGIN
  FOR i IN 1..10000 LOOP
    EXECUTE IMMEDIATE 'INSERT INTO
      test_nocons VALUES ('||i
      ||',RPAD(TO_CHAR('||i
      ||'),11,''.'')'
...
      ||',RPAD(TO_CHAR('||i
      ||'),30,''.''))';
  END LOOP;
  COMMIT;
END;
/
```

Constraints

```
BEGIN

  FOR i IN 1..10000 LOOP
    EXECUTE IMMEDIATE 'INSERT INTO
test_cons VALUES ('||i
      ||',RPAD(TO_CHAR('||i||'),11,''.'')'
...
      ||',RPAD(TO_CHAR('||i||'),30,''.''))';
  END LOOP;
  COMMIT;
END;
/
```

The results are shown in Table 4-10.

Table 4-10. *Results of the Second Test*

Results	No Constraints	With Constraints
Oracle 8.1.7.4	22.62s	53.97s
Oracle 9.2.0.1	16.06s	66.03s

The presence of the length-checking constraints can double or even quadruple parse time. The situation is more severe on Oracle 9 because each constraint is read from the catalogue by a separate cursor.

All processing in PeopleSoft can be affected by this phenomenon because many statements are generated dynamically. Application Engine batch processes are particularly susceptible because bind variables are resolved to literals before the SQL is submitted.

I have administered these tests on various operating systems and database versions with varying results, but this effect is always present. I have also seen a number of production systems suffering from this effect. It especially affects Application Engine processes because they resolve bind variables to literal values in the SQL that is submitted to the database. This effectively guarantees that every statement will need to be parsed. Hence Financials, with its batch processing, is affected more than HCM and CRM, which are mainly online applications.

Listing 4-7. *Oracle restricts VARCHAR2 columns to 4,000 bytes*

```
CREATE TABLE PS_MYTABLE
(MY_CHAR VARCHAR2(4000) CHECK(LENGTH(MY_CHAR)<=1333),...
```

The general rule is: don't use Unicode unless you have a good business reason. However, PeopleSoft will only support certain fixed-length character sets, as shown in Table 4-11. So if, for example, you operate in Poland, Czech Republic, Slovakia, and Hungary, you are forced to use Unicode because PeopleSoft will not support the EE8ISO8859P1 codepage, which covers those countries.

Table 4-11. *PeopleSoft-Supported Character Sets*

Character Set	Description
WE8ISO8859P1	All Western European characters
WE8ISO8859P15	All Western European characters and Euro symbol (€)
WE8MSWIN1252	Microsoft Windows 8-bit code page 1252
US7ASCII	7-bit Roman characters (no accented characters)
JA16SJIS	16-bit Japanese characters
JA16SJISTILDE	Microsoft version of JA16SJIS

If you do find that you need to use Unicode, there seem to be no supported options to work around the constraint problem.

Probably the simplest option is just to drop the constraints. There is a risk that a 10-character field could end up with 30 Roman characters, but the PIA will not let the user type in more characters than are defined in PeopleSoft, so the main risk comes from batch processes, and few of those process strings.

PeopleSoft has stated that in PeopleTools 9 they will move to the character-based column definitions introduced in Oracle 9.

PSSQLDEFN: Definition of SQL Objects

This table (see Table 4-12) is the parent of PSSQLTEXTDEFN. It holds SQL object definitions, and these objects are used widely in PeopleTools. Only rows where SQLTYPE = 2 correspond to DBA_VIEWS.

Until version 7.x of PeopleTools, the definition of SQL Views in PeopleTools was held in PSVIEWTEXT, and that table corresponded directly with DBA_VIEWS.

Table 4-12. *PSSQLDEFN: SQL Definitions*

PeopleTools PSSQLDEFN	Oracle DBA_VIEWS	Description of PeopleTools Column
 SQLID	(VIEW_NAME)	SQL object identifier. Not an exact match. This column has different meanings depending upon the value of SQLTYPE. SQLTYPE = 0: SQL object name SQLTYPE = 1: Application Engine step identifier SQLTYPE = 2: RECNAME SQLTYPE = 6: Application Engine XSLT (XML definition)
 SQLTYPE		SQL object type. 0 = SQL object referenced from elsewhere 1 = Application Engine step 2 = SQL view 5 = Queries for DDDAUDIT and SYSAUDIT 6 = Application Engine step XSLT
VERSION		Internal PeopleTools version to control caching of object.
LASTUPDDTTM		Timestamp of last update in Application Designer.
LASTUPDOPRID		Operator (developer) who last updated this record.
ENABLEEFFDT		
OBJECTOWNERID		Short code of functional area that owns the SQL (see PSRECDEFN.OBJECTOWNERID).

PSSQLTEXTDEFN

This table (see Table 4-13) is the child of PSSQLDEFN. It specifies the SQL text for the object so that PeopleTools can specify different SQL strings for different platforms. This table corresponds to DBA_VIEWS for SQLTYPE = 2.

Table 4-13. *PSSQLTEXTDEFN: SQL Text Definitions*

PeopleTools PSSQLTEXTDEFN	Oracle DBA_VIEWS	Description of PeopleTools Column
SQLID		See PSSQLDEFN.SQLID.
SQLTYPE		See PSSQLDEFN.SQLTYPE.
MARKET		Market. For example: GBL: Global product USF: US Federal product
DBTYPE		0 = SQLBase 1=DB2 for z/OS (formerly known as DB2 for OS/390 and DB2 for MVS) 2 = Oracle 3 = Informix 4 = DB2/Unix 5 = AllBase (deprecated in PeopleTools 5) 6 = Sybase 7 = Microsoft 8 = DB2/400 (deprecated in PeopleTools 8)
EFFDT		Date from which the definition is effective. Only used on SQLTYPE 0 (SQL objects) and 1 (Application Engine steps).
SQLTEXT	TEXT	Text of the SQL query part of the view. However, SQLTEXT can contain PeopleSoft %functions that either reference other SQL objects or expand to database-specific functions.

PSINDEXDEFN: Index Definition

This table (see Table 4-14) corresponds to DBA_INDEXES. The specification of certain column attributes on PSRECFIELD implies the creation of certain indexes (see also Chapter 5 for more information on keys and indexing). Additional indexes can then be specified in the Application Designer.

Up to PeopleTools 7.x, PSINDEXDEFN and PSKEYDEFN were maintained for all record types, not just SQL tables, despite the fact that indexes cannot be built for the other object types, such as views!

Indexes can be suppressed or enabled on either all or certain platforms. Thus PeopleSoft can deliver different indexes on Oracle and DB2.

Table 4-14. *PSINDEXDEFN: Index Definitions*

PeopleTools PSINDEXDEFN	Oracle DBA_INDEXES	Description of PeopleTools Column
RECNAME	TABLE_NAME	Not an exact match (see PSRECDEFN.SQLTABLENAME).
INDEXID	INDEX_NAME	Not an exact match. INDEX_NAME = 'PS'||INDEXID||RECNAME _ = PeopleSoft key index. Implied from record definition. 1–9 = Alternate search key indexes. Implied from record definition. (# = List index. Implied from record definition. Only applicable in PeopleTools 7.5 and earlier; not used in PeopleTools 8.) A–Z = User-specified index.
INDEXTYPE		1 = PeopleSoft key index 2 = List index 3 = Alternate search key index 4 = User-specified index
UNIQUEFLAG	UNIQUENESS	1 = UNIQUE 0 = NONUNIQUE
CLUSTERFLAG		No meaning in Oracle. In Microsoft SQL Server and Sybase, a clustered index is more efficient. This is not the same thing as an Oracle clustered index.
ACTIVEFLAG		1 = Index to be built on at least one platform 0 = Index not to be built on any platform
CUSTKEYORDER		0 = Order of columns in PeopleTools-generated index is the same as in the record 1 = Developer-specified order of columns in PeopleTools-generated index Indexes with columns on a sub-record cannot have a custom key order.
KEYCOUNT		Number of columns in a PeopleTools index. A sub-record counts as one, so this does not correspond to the number of columns in the database index.
DDLCOUNT		Number of distinct DDL override parameters specified on this index across all platforms.
PLATFORM_SBS		1 = Index to be built on SQL Base 0 = Index not to be built on SQL Base Not used from PeopleTools 8 on. The database platform is no longer supported.
PLATFORM_DB2		1 = Index to be built on DB2 for z/OS 0 = Index not to be built on DB2 for z/OS
PLATFORM_ORA		1 = Index to be built on Oracle 0 = Index not to be built on Oracle
PLATFORM_INF		1 = Index to be built on Informix 0 = Index not to be built on Informix
PLATFORM_DBX		1 = Index to be built on DB2/Unix 0 = Index not to be built on DB2/Unix

(Continues)

PeopleTools PSINDEXDEFN	Oracle DBA_INDEXES	Description of PeopleTools Column
PLATFORM_ALB		1 = Index to be built on AllBase 0 = Index not to be built on AllBase Not used from PeopleTools 5 on. The database platform is no longer supported.
PLATFORM_SYB		1 = Index to be built on Sybase 0 = Index not to be built on Sybase
PLATFORM_MSS		1 = Index to be built on Microsoft SQL Server 0 = Index not to be built on Microsoft SQL Server
PLATFORM_DB4		1 = Index to be built on DB2/AS400 0 = Index not to be built on DB2/AS400 Not used from PeopleTools 8 on. The database platform is no longer supported.
IDXCOMMENTS		Holds comments added by developer on user indexes only. Added in PeopleTools 8.4.

PSKEYDEFN: Index Definition

This table (see Table 4-15) contains one row for each column in each index specified in PeopleTools, so it corresponds to DBA_IND_COLUMNS.

Table 4-15. *PSKEYDEFN: Index Key Definition*

PeopleTools PSKEYDEFN	Oracle DBA_IND_COLUMNS	Description of PeopleTools Column
RECNAME	TABLE_NAME	Not an exact match (see PSRECDEFN.SQLTABLENAME).
INDEXID	INDEX_NAME	Not an exact match (see PSINDEXDEFN.INDEXID).
KEYPOSN	COLUMN_POSITION	Order of column or sub-record in index.
FIELDNAME	COLUMN_NAME	Name of field or sub-record on PeopleTools record.
ASCDESC	DESCEND	1 = Ascending order 0 = Descending order From PeopleTools 8.15 on, this no longer causes a descending order index to be built. PeopleSoft eventually removed this functionality because of Oracle Bug #869177, which affected descending indexes on most versions of Oracle 8.1.x. (See Oracle support website and People-Tools installation guide for PeopleTools 8.15.)

Recursive PeopleTools SQL

When a SQL statement that is not in the library cache is submitted to Oracle, the instance must parse the SQL in order to determine whether the SQL is valid, and if so how to execute it. During the parse process the instance will query the catalogue in order to obtain information about the tables and columns in the statement. The SQL generated by the parsing that queries the catalogue is called *recursive SQL* and can be seen in an Oracle SQL*Trace.[3]

PeopleSoft uses its own data dictionary in a similar way. All of the application objects in a PeopleSoft application are stored in the PeopleTools tables. When a component is executed, the component processor in the PIA will have to load the objects, if they are not already cached and up to date, by querying the PeopleTools tables. Application Engine, Query, and nVision also query PeopleTools tables in order to execute.

COMPONENT PROCESSOR

Up to PeopleTools 7.x, the term "panel processor" referred to the code in the Windows client and application server programs that was responsible for parsing the content of the PeopleTools tables, executing the application, and referencing the application data.

In PeopleTools 8, panel groups became components and panels became pages (although the names of the underlying PeopleTools tables did not change). Now, PeopleSoft talks about the component processor, which is additionally responsible for rendering the HTML pages, graphics, and JavaScript that are sent to the browser.

In this section, I will step through some of the recursive SQL issued by PeopleTools to load a component in the PIA. The examples are taken from the PERSONAL_DATA component, where employee name and address information is entered into the HCM application. Figure 4-4 shows a page from this component.

When DDL is executed in Oracle, recursive SQL is issued to maintain the catalogue. Similarly, the PeopleTools tables are administered via the Application Designer, and when an object is saved in the Application Designer, rows are inserted into or deleted from the Tools tables.

Page definitions are held in various rows on various PeopleTools tables. There are tables for each of the various design elements within the component, including these:

- Components: PSPNLGRPDEFN, PSPNLGROUP
- Pages: PSPNLDEFN
- Objects and positions within the page (including fields, scroll bars, graphics, and push buttons): PSPNLFIELD, PSPNLCNTRLDATA, PSPNLBTNDATA
- The records within which the fields exist: PSRECFIELD
- The data types of the field: PSDBFIELD
- The PeopleCode that executes when certain events occur: PSPCMPROG

3. See *Optimizing Oracle Performance* by Cary Millsap with Jeff Holt for a detailed explanation of this trace.

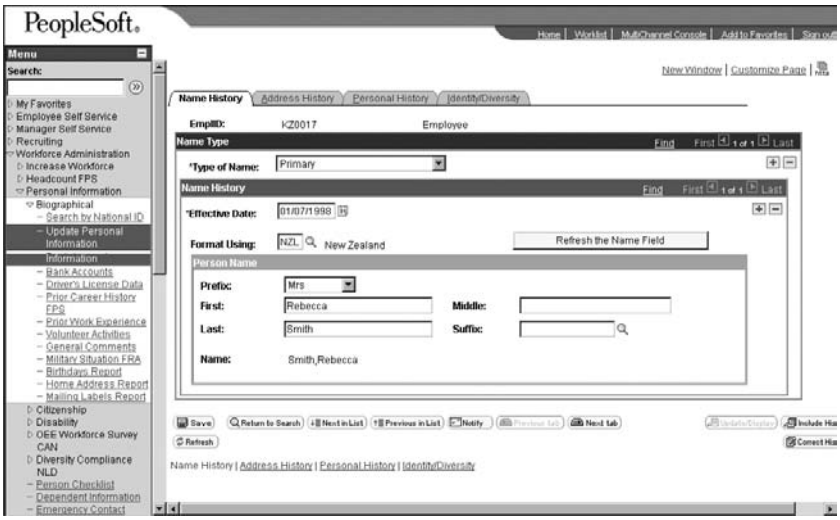


Figure 4-4. *Personal Data component of HCM*

The PeopleTools trace that is discussed over the following pages was obtained from an HCM 8.8 demonstration database.[4] The user entered the PERSONAL_DATA component, which had not previously been cached by PeopleTools. To load and open the component, the processor went through a number of stages:

- Check the PeopleTools object version numbers for caching
- Load the component definition
- Perform the component search dialog
- Identify and load pages in the component
- Load any component PeopleCode
- Load the page definitions
- Load the fields on each page
- Load the record definitions
- Load the field definitions
- Identify any sub-records
- Load the field labels
- Load the record DDL definitions
- Recursively load the sub-record definitions
- Load the application data

4. Enabling PIA trace is described in Chapter 9.

Some stages may have been executed more than once for different objects. In general, not all components require all these stages; for example, a record may not have sub-records.

Navigating from signon to the page generated 44,000 lines of trace, so I will only show some extracts.

Version Numbers and Caching

A system of version numbering is used on PeopleTools objects to enable the caching algorithm to determine whether a cached object is up to date. The table PSVERSION holds a global version number (object type SYS) and a version number for each of the PeopleTools object types (records, panels, etc.).

When an object is changed and saved by the Application Designer, the version number on PSVERSION for that object type is incremented and allocated to the object. The global version number is also incremented.[5] By comparing version numbers on the database and in the local caches, PeopleTools processes can determine whether the cached objects are up to date or have changed.

In a PeopleSoft trace, you will frequently see queries of the PSVERSION table like the one shown in bold in Listing 4-8.

Listing 4-8. *PeopleTools version-number check*

```
PSAPPSRV.3928   1-30702  16.12.40    0.010 Cur#1.3928.HR88 RC=0 Dur=0.000 COM
Stmt=SELECT VERSION FROM PSVERSION WHERE OBJECTTYPENAME = 'SYS'
```

PeopleTools uses this query to determine whether any new objects have been added to any Tools table. If the global version number on the database is higher than the cached version, then something must have been added, and PeopleTools will go on to examine version numbers for the PeopleTools object types. Where these version numbers have incremented, the Tools tables are queried, as shown next.

Component Definition

When the "Update Personal Information" link on the menu is clicked in the PIA, the application server starts to load the component, beginning with the component definition on PSPNLGRPDEFN, see Listing 4-9.

Listing 4-9. *Loading the panel group definition*

```
PSAPPSRV.3928   1-29830  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT DESCR, ACTIONS, VERSION, SEARCHRECNAME, ADDSRCHRECNAME,
SEARCHPNLNAME, LOADLOC, SAVELOC, DISABLESAVE, PRIMARYACTION, DFLTACTION,
DFLTSRCHTYPE,  DEFERPROC, EXPENTRYPROC, REQSECURESSL, INCLNAVIGATION,
FORCESEARCH, ALLOWACTMODESEL, PNLNAVFLAGS,  TBARBTNS, SHOWTBAR, ADDLINKMSGSET,
ADDLINKMSGNUM, SRCHLINKMSGSET, SRCHLINKMSGNUM,  SRCHTEXTMSGSET, SRCHTEXTMSGNUM,
OBJECTOWNERID, TO_CHAR(LASTUPDDTTM,'YYYY-MM-DD-HH24.MI.SS."000000"'),
LASTUPDOPRID,  DESCRLONG   FROM PSPNLGRPDEFN WHERE PNLGRPNAME = :1 AND MARKET = :2
```

5. PeopleSoft uses ordinary tables rather than Oracle sequences. This is discussed in Chapter 8.

```
PSAPPSRV.3928   1-29831  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=13 value=PERSONAL_DATA
PSAPPSRV.3928   1-29832  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-2 type=2 length=3 value=GBL
```

Search Record

A search record is specified for the component on the PeopleTools table PSPNLGRPDEFN in the column SEARCHRECNAME.

In Figure 4-5 (and Listing 4-10), the operator has searched for employees whose surnames begin with "SMITH". The layout of the search dialog is controlled by the specification of the component search record, PS_XX_GBL, which identifies the data to be retrieved into the component by primary key.

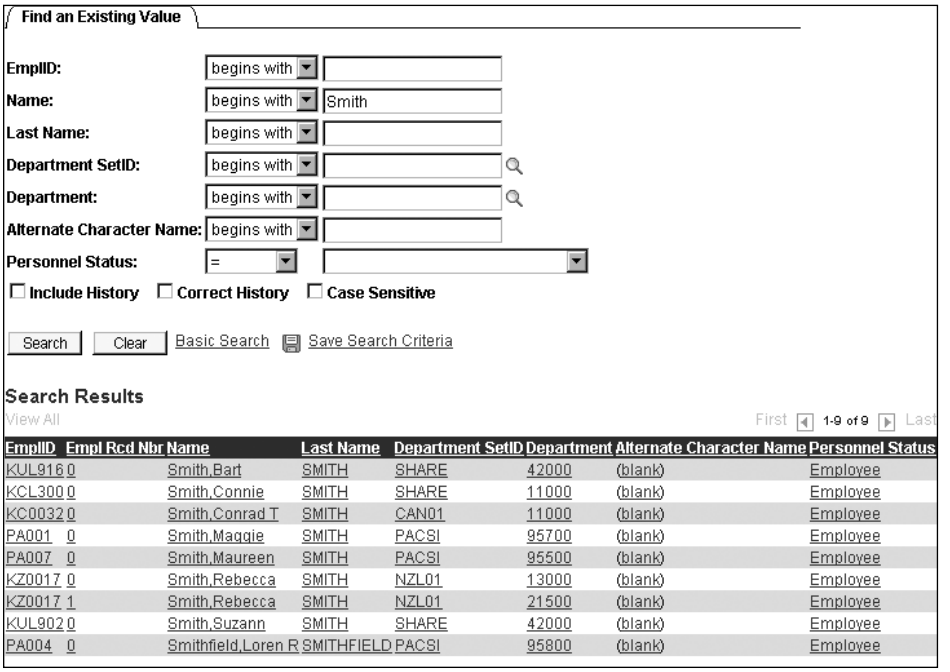


Figure 4-5. *Personal search dialog*

Search records also provide row-level security for the component by only retrieving data that the operator is entitled to see.

In HCM, operators are arranged in classes. The operator class is granted access to a point on the department security tree, and the security view is constructed so that it returns only employees in or below those departments to which the operator class is granted access. If, and only if, the column ROWSECCLASS is a column on the search record, as it is on PS_XX_GBL, the query is restricted to the current operator's class. In this example, the operator is in the class HCDPALL, so the search dialog only returns the employees that the operator is entitled to view.

Listing 4-10. *Component search dialog*

```
PSAPPSRV.3928   1-30561  16.12.01    0.021 Cur#1.3928.HR88 RC=0
Dur=0.000 COM Stmt=SELECT DISTINCT EMPLID, EMPL_RCD, NAME, LAST_NAME_SRCH,
SETID_DEPT, DEPTID, NAME_AC, PER_STATUS FROM PS_XX_GBL WHERE ROWSECCLASS=:1
AND UPPER(NAME) LIKE UPPER('Smith') || '%' ESCAPE '\' ORDER BY NAME, EMPLID
PSAPPSRV.3928   1-30562  16.12.01    0.000 Cur#1.3928.HR88 RC=0
Dur=0.000 Bind-1 type=2 length=7 value=HCDPALL
PSAPPSRV.3928   1-30563  16.12.02    1.602 Cur#1.3928.HR88 RC=0 Dur=0.000 Commit
```

Strictly speaking, the query generated by the search dialog is not a recursive statement because it is referencing an application record, but the search dialog is not explicitly coded by the developer. It is automatically generated by PeopleTools because a search record appears on the component definition (PSPNLGRPDEFN).

Note that while the ROWSECCLASS condition is on a bind variable, the content of the search field in the page appears as a literal string in the trace. Every time the operator enters a different search string, a different SQL statement will be generated. The database will treat it as a new SQL statement and will have to parse it. This is one reason why there is so much parsing on a PeopleSoft database.

Case-Insensitive Searching

PeopleTools makes the search for the surname case-insensitive by putting the UPPER() function around the name and the search string. This can have the effect of preventing any index on the NAME being used effectively, and it can cause performance problems. There are two ways to address this:

- Case-insensitive searching can be disabled globally in "PeopleTools Options."
- Function-based indexes can be created on the case-sensitive columns, but Application Designer is not capable of generating them automatically, so they have to be managed manually, as in Listing 4-11.

Listing 4-11. *Creating a function-based index*

```
CREATE INDEX PSZNAMES ON PS_NAMES(UPPER(NAME));
```

Component Pages

Pages exist within a component, and all of the pages within a component are loaded simultaneously. PSPNLGROUP specifies all the pages within the components, whether they appear to the user or not, the order in which they appear, and the details on the tabs.

Listing 4-12. *Determining pages in component*

```
PSAPPSRV.3928   1-29836  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT COUNT(*)  FROM PSPNLGROUP WHERE PNLGRPNAME = :1 AND MARKET = :2
PSAPPSRV.3928   1-29837  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=13 value=PERSONAL_DATA
PSAPPSRV.3928   1-29838  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-2 type=2 length=3 value=GBL
```

```
PSAPPSRV.3928   1-29839  16.11.15    0.020 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT PNLNAME, ITEMNAME, HIDDEN, ITEMLABEL, FOLDERTABLABEL, SUBITEMNUM
FROM PSPNLGROUP WHERE PNLGRPNAME = :1 AND MARKET = :2 ORDER BY SUBITEMNUM
PSAPPSRV.3928   1-29840  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=13 value=PERSONAL_DATA
PSAPPSRV.3928   1-29841  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-2 type=2 length=3 value=GBL
```

In Figure 4-4 there are four tabs visible; these are the pages in the PERSONAL_DATA component. Listing 4-13 shows the contents of PSPNLGROUP for this component.

Listing 4-13. *Contents of PSPNLGROUP*

```
PNLNAME           ITEMNAME          HIDDEN ITEMLABEL           SUBITEMNUM
----------------- ----------------- ------ ------------------- ----------
PERSONAL_DATA1    NAMES                  0 &Name History       1
PERSONAL_DATA1B   ADDRESS_HISTORY        0 &Address History    2
PERSONAL_DATA2    PERSONAL_DATA2         0 &Personal History   3
PERSONAL_DATA3    PERSONAL_DATA_3        0 &Identity/Diversity 4
PERSONAL_DATA_WRK PERSONAL_DATA_WRK      1 Personal Data Wrk   5
```

The query actually returns five rows. PERSONAL_DATA_WRK is a hidden page that contains working storage fields referenced by PeopleCode.

Component PeopleCode

The last stage in loading the component is to find any component-level PeopleCode (Listing 4-14).

Listing 4-14. *Loading component-level PeopleCode*

```
PSAPPSRV.3928   1-29897  16.11.15    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000 COM
Stmt=SELECT OBJECTID1,OBJECTVALUE1, OBJECTID2,OBJECTVALUE2, OBJECTID3,
OBJECTVALUE3, OBJECTID4,OBJECTVALUE4, OBJECTID5,OBJECTVALUE5, OBJECTID6,OBJECTVALUE6,
OBJECTID7,OBJECTVALUE7  FROM PSPCMPROG WHERE OBJECTID1 = :1
AND OBJECTVALUE1 = :2 AND  OBJECTID2 = :3 AND OBJECTVALUE2 = :4 ORDER BY
OBJECTID1,OBJECTVALUE1, OBJECTID2,OBJECTVALUE2, OBJECTID3,OBJECTVALUE3,
OBJECTID4,OBJECTVALUE4, OBJECTID5,OBJECTVALUE5, OBJECTID6,OBJECTVALUE6,
OBJECTID7,OBJECTVALUE7
PSAPPSRV.3928   1-29898  16.11.15    0.000 Cur#1.3928.HR88 RC=0
Dur=0.000 Bind-1 type=8 length=4 value=10
PSAPPSRV.3928   1-29899  16.11.15    0.000 Cur#1.3928.HR88 RC=0
Dur=0.000 Bind-2 type=2 length=13 value=PERSONAL_DATA
PSAPPSRV.3928   1-29900  16.11.15    0.000 Cur#1.3928.HR88 RC=0
Dur=0.000 Bind-3 type=8 length=4 value=39
PSAPPSRV.3928   1-29901  16.11.15    0.000 Cur#1.3928.HR88 RC=0
Dur=0.000 Bind-4 type=2 length=3 value=GBL
```

Listing 4-15 shows that there are three pieces of PeopleCode on the PERSONAL_DATA component, and one on the "Refresh the Name Field" button on the Name History page (see Figure 4-4).

Listing 4-15. *Component-level PeopleCode*

```
OBJECTID1 OBJECTVALUE1  OBJECTID2 OBJ OBJECTID3 OBJECTVALUE3   OBJECTID4
--------- ------------- --------- --- --------- -------------- ---------
OBJECTVALUE4    OBJECTID5 OBJECTVALUE5 OBJECTID6 OBJECTVALUE6  OBJECTID7
--------------- --------- ------------ --------- ------------- ---------
       10 PERSONAL_DATA        39 GBL          1 DERIVED_NAME           2
UPDATE_NAME_BTN        12 FieldChange           0                        0

       10 PERSONAL_DATA        39 GBL         12 PostBuild              0
                      0                        0                        0

       10 PERSONAL_DATA        39 GBL         12 PreBuild               0
                      0                        0                        0

       10 PERSONAL_DATA        39 GBL         12 SavePostChange         0
                      0                        0                        0
```

Page Definition

A component can consist of a number of pages. Each page within the component is loaded, starting with the page definition stored on the table PSPNLDEFN (Listing 4-16).

Listing 4-16. *Loading page definition*

```
PSAPPSRV.3928   1-30703  16.12.40    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT VERSION, PNLTYPE, GRIDHORZ, GRIDVERT, FIELDCOUNT, MAXPNLFLDID,
HELPCONTEXTNUM, PANELLEFT, PANELTOP, PANELRIGHT, PANELBOTTOM, PNLSTYLE,
STYLESHEETNAME, PNLUSE, DEFERPROC, DESCR, POPUPMENU, LICENSE_CODE,
TO_CHAR(LASTUPDDTTM,'YYYY-MM-DD-HH24.MI.SS."000000"'), LASTUPDOPRID,
OBJECTOWNERID, DESCRLONG FROM PSPNLDEFN WHERE PNLNAME = :1
PSAPPSRV.3928   1-30704  16.12.40    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=14 value=PERSONAL_DATA1
```

Page Fields

Each field on the page is then loaded (Listing 4-17).

Listing 4-17. *Loading fields on a page*

```
PSAPPSRV.3928   1-30710  16.12.41    0.010 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT PNLFLDID, FIELDTYPE, EDITSIZE, FIELDLEFT, FIELDTOP, FIELDRIGHT,
FIELDBOTTOM, EDITLBLLEFT, EDITLBLTOP, EDITLBLRIGHT, EDITLBLBOTTOM, DSPLFORMAT, DSPLFILL,
LBLTYPE, LBLLOC, LBLPADSIZE, LABEL_ID, LBLTEXT, FIELDUSE, FIELDUSETMP, DEFERPROC,
OCCURSLEVEL, OCCURSCOUNT1, OCCURSCOUNT2, OCCURSCOUNT3, OCCURSOFFSET1,
OCCURSOFFSET2, OCCURSOFFSET3, PNLFIELDNAME, RECNAME, FIELDNAME, SUBPNLNAME, ONVALUE,
OFFVALUE, ASSOCFIELDNUM, FIELDSTYLE, LABELSTYLE, FIELDSIZETYPE, LABELSIZETYPE, PRCSNAME,
PRCSTYPE, FORMATFAMILY, DISPFMTNAME, PROMPTFIELD, POPUPMENU, TREECTRLID,
TREECTRLTYPE, MULTIRECTREE, NODECOUNT, GRDCOLUMNCOUNT, GRDSHOWCOLHDG,
```

```
GRDSHOWROWHDG, GRDODDROWSTYLE, GRDEVENROWSTYLE, GRDACTIVETABSTYLE,
GRDINACTIVETABSTYL, GRDNAVBARSTYLE, GRDLABELSTYLE, GRDLBLMSGSET, GRDLBLMSGNUM,
GRDLBLALIGN, GRDACTTYPE, TABENABLE, PBDISPLAYTYPE, OPENNEWWINDOW, URLDYNAMIC,  URL_ID,
GOTOPORTALNAME, GOTONODENAME, GOTOMENUNAME, GOTOPNLGRPNAME,  GOTOMKTNAME,
GOTOPNLNAME, GOTOPNLACTION, SRCHBYPNLDATA, SCROLLACTION, TOOLACTION, CONTNAME,
CONTNAMEOVER, CONTNAMEDISABLE, PTLBLIMGCOLLAPSE, PTLBLIMGEXPAND,
SELINDICATORTYPE, PTADJHIDDENFIELDS, PTCOLLAPSEDATAAREA, PTDFLTVIEWEXPANDED,
PTHIDEFIELDS, SHOWCOLHIDEROWS, PTLEBEXPANDFIELD, SHOWTABCNTLBTN,
SECUREINVISIBLE, ENABLEASANCHOR, URLENCODEDBYAPP, USEDEFAULTLABEL,
GRDALLOWCOLSORT, PNLNAME, FIELDNUM FROM PSPNLFIELD WHERE PNLNAME = :1 ORDER BY FIELDNUM
PSAPPSRV.3928   1-30711  16.12.41    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=14 value=PERSONAL_DATA1
```

Record Definition

The record definitions for all of the records referenced on all of the pages in the component are loaded (Listing 4-18).

Listing 4-18. *Loading record definition*

```
PSAPPSRV.3928   1-30783  16.12.41    0.010 Cur#2.3928.HR88 RC=0 Dur=0.010
COM Stmt=SELECT VERSION, FIELDCOUNT, RECTYPE, RECUSE, OPTTRIGFLAG, AUDITRECNAME,
SETCNTRLFLD, RELLANGRECNAME, OPTDELRECNAME, PARENTRECNAME, QRYSECRECNAME,
SQLTABLENAME, BUILDSEQNO, OBJECTOWNERID, TO_CHAR(LASTUPDDTTM,'YYYY-MM-DD-
HH24.MI.SS."000000"'), LASTUPDOPRID, SYSTEMIDFIELDNAME, TIMESTAMPFIELDNAME, RECDESCR,
DESCRLONG  FROM PSRECDEFN WHERE RECNAME = :1
PSAPPSRV.3928   1-30784  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=6 value=PERSON
PSAPPSRV.3928   1-30785  16.12.41    0.010 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT COUNT(*) FROM PSRECDEFNLANG WHERE RECNAME = :1
PSAPPSRV.3928   1-30786  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=6 value=PERSON
```

The record definition includes details such as the underlying SQLTABLENAME, which is the name of the table in the database that corresponds to the record.

Field Definitions

For each record that is loaded, all of the fields are also loaded. PeopleTools has the concept of sub-records, which are groups of fields or other sub-records that, for ease and consistency, can be included in a number of different records. Sub-records can contain other sub-records. The concept is similar to a part explosion, or to a structure in a C program.

In this example, the record NAMES has a sub-record NAMEGBL_SBR. This sub-record is used in several other records in the HCM application.

PeopleTools starts by loading fields on a record that are not part of a sub-record.

Listing 4-19. *Load fields that are not sub-records*

```
PSAPPSRV.3928   1-30819  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT VERSION, A.FIELDNAME, FIELDTYPE, LENGTH, DECIMALPOS, FORMAT,
FORMATLENGTH, IMAGE_FMT, FORMATFAMILY, DISPFMTNAME,
DEFCNTRYYR,IMEMODE,KBLAYOUT,OBJECTOWNERID, DEFRECNAME, DEFFIELDNAME,
CURCTLFIELDNAME, USEEDIT, USEEDIT2, EDITTABLE, DEFGUICONTROL, SETCNTRLFLD, LABEL_ID,
TIMEZONEUSE, TIMEZONEFIELDNAME, CURRCTLUSE, RELTMDTFIELDNAME,
TO_CHAR(B.LASTUPDDTTM,'YYYY-MM-DD-HH24.MI.SS."000000"'), B.LASTUPDOPRID,
B.FIELDNUM, A.FLDNOTUSED, A.AUXFLAGMASK, B.RECNAME FROM PSDBFIELD A, PSRECFIELD B
WHERE B.RECNAME = :1 AND A.FIELDNAME = B.FIELDNAME AND B.SUBRECORD = 'N'
ORDER BY B.RECNAME, B.FIELDNUM
PSAPPSRV.3928   1-30820  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
```

Sub-Records

Columns that are not sub-records are loaded first (Listing 4-20) with information about their type, length, and how they are to be displayed. Then the sub-records on NAMES are loaded. PeopleSoft has not changed this algorithm in PeopleTools 8. It still loads the fields from PSRECFIELD and not PSRECFIELDDB.

Listing 4-20. *Identifying sub-records on the record*

```
PSAPPSRV.3928   1-30821  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT FIELDNUM, FIELDNAME, TO_CHAR(LASTUPDDTTM,'YYYY-MM-DD-
HH24.MI.SS."000000"'), LASTUPDOPRID, RECNAME FROM PSRECFIELD WHERE RECNAME = :1 AND
SUBRECORD = 'Y' ORDER BY RECNAME, FIELDNUM
PSAPPSRV.3928   1-30822  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
```

There is just one sub-record on the NAMES record (Listing 4-21). It will be loaded a little later on, after all the records are loaded.

Listing 4-21. *Sub-records on NAMEGBL_SBR*

```
  FIELDNUM FIELDNAME          TO_CHAR(LASTUPDDTTM,'YYYY-
---------- ------------------ -------------------------
LASTUPDOPRID                   RECNAME
------------------------------ ---------------
         4 NAMEGBL_SBR        2000-03-14-09.40.22.000000
PPLSOFT                        NAMES
```

Field Labels

Field labels are the prompts that appear by a field in a page. By default, whenever fields of the same name appear in pages, they will have the same label next to them, although the developer can override this on the page when the field is added. This approach tends to make the labeling of a field consistent wherever it appears in the system, and simplifies translation of field prompts into other languages.

Field descriptions are loaded after the fields are loaded (Listing 4-22).

Listing 4-22. *Loading field labels*

```
PSAPPSRV.3928   1-30823  16.12.41    0.010 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT FIELDNAME, LABEL_ID, LONGNAME, SHORTNAME, DEFAULT_LABEL FROM
PSDBFLDLABL WHERE FIELDNAME IN (SELECT A.FIELDNAME FROM PSDBFIELD A, PSRECFIELD B
WHERE B.RECNAME = :1 AND A.FIELDNAME = B.FIELDNAME) ORDER BY FIELDNAME, LABEL_ID
PSAPPSRV.3928   1-30824  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
```

Record DDL

Part of the record definition describes how the Application Designer should build the table and index or view on the database. Although the PIA will never do that, this information is part of the record definition, and it is all loaded (Listing 4-23) and cached at the same time.

Listing 4-23. *Load index definitions*

```
PSAPPSRV.3928   1-30825  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT INDEXID, INDEXTYPE, UNIQUEFLAG, CLUSTERFLAG, ACTIVEFLAG,
CUSTKEYORDER, IDXCOMMENTS, PLATFORM_SBS, PLATFORM_DB2, PLATFORM_ORA,
PLATFORM_INF, PLATFORM_DBX, PLATFORM_ALB, PLATFORM_SYB, PLATFORM_MSS,
PLATFORM_DB4, RECNAME FROM PSINDEXDEFN WHERE RECNAME = :1 ORDER BY RECNAME, INDEXTYPE,
INDEXID
PSAPPSRV.3928   1-30826  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
PSAPPSRV.3928   1-30827  16.12.41    0.030 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT INDEXID, KEYPOSN, FIELDNAME, ASCDESC, RECNAME FROM PSKEYDEFN
WHERE RECNAME = :1 ORDER BY RECNAME, INDEXID, KEYPOSN
PSAPPSRV.3928   1-30828  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
PSAPPSRV.3928   1-30829  16.12.41    0.030 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT INDEXID, PLATFORMID, SIZINGSET, PARMNAME, PARMVALUE, RECNAME
FROM PSIDXDDLPARM WHERE RECNAME = :1 ORDER BY RECNAME, INDEXID, PLATFORMID, SIZINGSET,
PARMNAME
PSAPPSRV.3928   1-30830  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
PSAPPSRV.3928   1-30831  16.12.41    0.030 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT PLATFORMID, SIZINGSET, PARMNAME, PARMVALUE, RECNAME FROM
PSRECDDLPARM WHERE RECNAME = :1 ORDER BY RECNAME, PLATFORMID, SIZINGSET, PARMNAME
PSAPPSRV.3928   1-30832  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=5 value=NAMES
PSAPPSRV.3928   1-30833  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT DDLSPACENAME, DBNAME, DBTYPE FROM PSRECTBLSPC WHERE RECNAME = :1
ORDER BY DBTYPE
```

Sub-Record Definition

After the rest of the record is loaded, the sub-records are then loaded (Listing 4-24). The whole of the record-loading process is repeated recursively. This includes the DDL definition in the previous section, although it is meaningless for a sub-record.

Listing 4-24. *Loading sub-record definition*

```
PSAPPSRV.3928   1-30837  16.12.41    0.010 Cur#2.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT VERSION, FIELDCOUNT, RECTYPE, RECUSE, OPTTRIGFLAG, AUDITRECNAME,
SETCNTRLFLD, RELLANGRECNAME, OPTDELRECNAME, PARENTRECNAME, QRYSECRECNAME,
SQLTABLENAME, BUILDSEQNO, OBJECTOWNERID, TO_CHAR(LASTUPDDTTM,'YYYY-MM-DD-
HH24.MI.SS."000000"'), LASTUPDOPRID, SYSTEMIDFIELDNAME, TIMESTAMPFIELDNAME,
RECDESCR, DESCRLONG  FROM PSRECDEFN WHERE RECNAME = :1
PSAPPSRV.3928   1-30838  16.12.41    0.000 Cur#2.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=11 value=NAMEGBL_SBR
```

Application SQL

Eventually, after all of the application components have been loaded, the application data is queried into the page. In Listing 4-25 the NAMES record for employee KZ0017 is queried from the database, loaded into the component buffer, and displayed on the screen.

Listing 4-25. *Load the application data for employee KZ001*

```
PSAPPSRV.3928   1-44210  16.13.10    0.371 Cur#1.3928.HR88 RC=0 Dur=0.000
COM Stmt=SELECT EMPLID, NAME_TYPE, EFFDT, TO_CHAR(EFFDT,'YYYY-MM-DD'),
COUNTRY_NM_FORMAT, NAME, NAME_INITIALS, NAME_PREFIX, NAME_SUFFIX,
NAME_ROYAL_PREFIX, NAME_ROYAL_SUFFIX, NAME_TITLE, LAST_NAME_SRCH,
FIRST_NAME_SRCH, LAST_NAME, FIRST_NAME, MIDDLE_NAME, SECOND_LAST_NAME,
SECOND_LAST_SRCH, NAME_AC, PREF_FIRST_NAME, PARTNER_LAST_NAME,
PARTNER_ROY_PREFIX, LAST_NAME_PREF_NLD FROM PS_NAMES WHERE EMPLID=:1 ORDER BY
EMPLID, NAME_TYPE, EFFDT DESC
PSAPPSRV.3928   1-44211  16.13.10    0.000 Cur#1.3928.HR88 RC=0 Dur=0.000
Bind-1 type=2 length=6 value=KZ0017
```

There are a few things to note in the SQL in Listing 4-25:

- This query has been dynamically constructed by PeopleTools from the information loaded from the PeopleTools tables.
- The columns in the SELECT list were derived from the columns on the NAMES record and its sub-record. The field types are stored on PSDBFIELD. This indicates that EFFDT is a date, and the TO_CHAR function is added to convert it to a date string.
- The query on the record is by the primary key columns in common with the search record, so this query is only on EMPLID. The search key value was taken from the row selected on the search dialog. Whether a field is part of the unique key, the order in which it should be queried, and whether a field appears in a search dialog are all attributes of the field on that record and were queried from PSRECFIELD.
- The query is ordered by all the primary key columns in descending order where the key is specified as descending.

We have now seen how PeopleSoft retrieves the application and the data structures from the PeopleTools tables and then queries the application data. Queries on the Tools tables are effectively recursive queries, just as an Oracle database may recursively query its catalogue when a SQL statement is parsed.

Data Dictionary Synchronization

I have demonstrated how PeopleTools dynamically generates SQL to reference the application data from information in the PeopleTools tables, in much the same way as a database recursively queries its own catalogue. I have also described the relationship between certain PeopleTools tables and the Oracle database catalogue.

It is essential that, wherever possible, all DDL changes are specified via the Application Designer, which will then also be used to generate all of the DDL scripts. If any changes to the data model are not implemented in this way, they will introduce discrepancies between the two data dictionaries. They are likely to cause errors because PeopleSoft will build invalid SQL or the changes will be lost in a subsequent application upgrade or patch. If other parts of PeopleTools tables are not referentially integral, the application server or even the Application Designer may simply crash.

Here are two examples:

- If a developer removes a character or numeric column from a table and does not implement the ALTER TABLE script generated by Application Designer, the application will fail when it tries to insert data into the table because the column is no longer in the generated insert SQL, the default value of the column is NULL, and the column has a NOT NULL constraint. The application will fail with "ORA-1400: cannot insert NULL into ..."
- If a DBA decides to add an index to a table, and then a developer makes a subsequent change, that record may get rebuilt. PeopleTools usually alters tables by building a copy, copying the data from the original table to the copy, renaming the tables, rebuilding the indexes, and dropping the original table. If PeopleTools doesn't know about the index that the DBA added, it will not rebuild it, and that index will be lost.

There are, however, some Oracle-specific object types for which the Application Designer cannot construct DDL. It is not possible to build Partitioned, Global Temporary, or Index Organized Tables (see Chapter 6). The columns must still be specified in Application Designer, but the actual creation of the objects must be handled manually.[6]

PeopleSoft provides two diagnostic SQR reports to help identify problems in the PeopleTools tables so that they can be repaired:

- DDDAUDIT compares the two data dictionaries and reports on discrepancies between them.
- SYSAUDIT checks relational integrity within the PeopleTools tables.

Both reports run a series of queries that generate an exception report. The possible errors from this report and the recommended remedies are set out in the "Administration Tools PeopleBook."

6. Some of these Oracle object types are discussed in Chapter 6. For Global Payroll, a PL/SQL script was developed to build partitioned result tables and the global temporary working storage tables according to the specifications in the PeopleTools tables.

Column Difference Audit Script

One significant limitation of PeopleSoft's DDDAUDIT is that this report does not check that columns are defined consistently on both data dictionaries. If a developer adds a column to a table and forgets to rebuild that table, it will not be reported by DDDAUDIT, but PeopleTools will add that column to the SQL that it generates. The result will be Oracle error "ORA-00904: <column name>: invalid identifier."

The `colaudit.sql` script was originally written to detect this and similar problems.[7] The script should simply be run in SQL*Plus while connected to the database schema in which the PeopleSoft database resides. It generates a report of exceptions, set out in Table 4-16, to the screen and a spool file. It also creates a project in Application Designer from which DDL scripts can be built to rebuild objects if necessary.

Table 4-16. `colaudit` *Report Sections*

Error Code	Error Description	Comment
COL-01	Object in PeopleSoft data dictionary, but not in Oracle.	DDDAUDIT also performs this test. It is included here for convenience.
COL-02	Object in Oracle catalogue, but not in PeopleSoft.	DDDAUDIT also performs this test. It is included here for convenience.
COL-03	More instances of PS temporary table exist in Oracle than are defined in PeopleSoft.	
COL-04	Corresponding PeopleSoft and Oracle tables and views with different numbers of columns.	This section summarizes COL-05 and COL-06.
COL-05	Columns in PeopleSoft data dictionary, but not in Oracle catalogue.	
COL-06	Columns in Oracle catalogue, but not in PeopleSoft.	
COL-07	Columns in both Oracle and PeopleSoft, but with different definitions.	There are separate versions of this test for PeopleTools 8.1x and PeopleTools 8.4x to handle the new SYSTEMIDFIELD.
COL-08	Records referenced as but no longer defined as sub-records.	This doesn't cause a runtime error but can produce spurious messages in COL-01 and in DDDAUDIT.
COL-09	Tables or views with the same number of columns in both Oracle and PeopleSoft, but in different positions.	This can be a problem if INSERT INTO statements do not explicitly list columns.
COL-10	Corresponding PeopleSoft and Oracle indexes with different numbers of columns.	
COL-11	Indexes in both Oracle and PeopleSoft, but with columns in different positions.	
COL-12	Warning: key columns not at top of record definition.	This is not an error, but it is contrary to PeopleTools development advice.

7. `colaudit.sql` is available for download from the Go-Faster Consultancy web site (www.go-faster.co.uk).

Mostly, the problems reported by `colaudit.sql` can be resolved by rebuilding the objects.

Summary

The chapter has described the PeopleTools data dictionary and its relationship to the Oracle catalogue, and it has provided a reference for the PeopleTools tables that correspond to table structures.

On all database platforms, the catalogue tells you what actually exists in the database, and it is used by the database to validate and execute SQL statements. The PeopleSoft data dictionary tells you what should exist in the database, and it is used by PeopleTools to dynamically generate SQL statements at runtime.

Discrepancies in corresponding parts of the catalogues can cause PeopleTools to generate invalid SQL, leading to runtime errors. Hence, all changes to the database should, whenever possible, be implemented via the scripts built by the Application Designer.

CHAPTER 5

Keys and Indexing

This chapter discusses the use of keys and indexes in PeopleSoft, and how they translate into indexes and implicit constraints in Oracle (there is an almost total absence of explicit constraints in a PeopleSoft database). In the previous chapter, we looked at how the two data dictionaries are related. Here, we will look at how PeopleSoft uses the "keys"[1] defined in its data dictionary to generate SQL in the application and to create the indexes in the database to support that SQL.

We will also examine the role of the Application Designer in more detail. This Windows client program is a complete development environment for PeopleSoft online applications. In addition, it is used to specify Application Engine batch programs, to migrate functionality between PeopleSoft environments, and to apply PeopleSoft patches.

The Application Designer is also a tool for the DBA. It is used to specify all the database objects, and will then generate and optionally execute the DDL scripts to build or alter them. I will explain how the definition of PeopleSoft records controls which indexes are built, upon which columns, and whether the indexes are unique. The derivation of the generated DDL is explained in the next chapter.

First, I will recap some basic principles related to indexes, constraints, and keys.

What Is the Purpose of an Index?

This is the most succinct definition of the purpose of indexes that I have seen:

> *Indexes exist to reduce the cost of data retrieval.*[2]

I will start by restating the obvious. Databases store data in tables. That data can be queried back via a SQL statement.

```
SELECT column1
FROM   table1
WHERE  column2 = 'VALUE';
```

1. "Keys" is another word that means one thing in database terms and a several different things when used by PeopleSoft.
2. Jonathan Lewis, *Practical Oracle 8*i (Boston, MA: Addison-Wesley Professional, 2001).

If the table has no indexes, when this statement is submitted, the database will scan through the entire table and identify all the rows where the condition is met. Every row in the table will have to be read from disk or, if you are lucky, the block buffer cache. Data that does not match the query will be discarded. The time taken to perform this operation will, of course, increase roughly in proportion to the number of rows on the table.

This is obviously not efficient for even moderately sized tables. Therefore, indexes exist to help the database find the data quickly. Indexes are really tables that, for each row in the table, contain the data for the indexed columns and the physical address of that row in the table.

If we go back to the example and assume that there is an index on COLUMN2 of TABLE1, then when the query is submitted, the database will identify COLUMN2 as an indexed column when it parses the SQL statement. It might use that index to find the rows for which the data matches the condition.

What Is a Constraint?

An *integrity* constraint restricts the values in one or more columns such that they conform to a condition. A *referential* or *foreign key* constraint verifies that data in the database is referentially integral, ensuring a child record does not exist without a corresponding parent record.

If a constraint is not met, it prevents a database update from taking place. Constraints can be created deferrable and deferred to commit time, in which case should the constraint be violated the whole transaction will roll back.

The benefit of using constraints is that all of the processing occurs within the database engine and so does not have to be coded into every program that updates the database. In a client/server model, this will also reduce SQL traffic across the network. However, the cost is some additional overhead on the database server.

PeopleSoft never uses explicitly defined constraints (except for the length-checking constraints on character columns in a Unicode database, as discussed in Chapter 4) because each database implements them differently. In PeopleTools, the validation of field values and referential integrity is defined in the PeopleSoft data dictionary in terms of the data model, and is executed by the PIA. Wherever the same record is used in the online system, the same validation is employed.

CAN I ADD EXPLICIT CONSTRAINTS TO PEOPLESOFT?

It has occasionally been suggested that it would be possible to use the information in the PeopleTools tables to generate a set of explicit constraints. I do not believe that this is either possible or advisable, for the following reasons.

- Some of the referential validations involve effective dated logic that could not be coded within standard constraints.
- Parent records are not always defined in PeopleSoft where they could be.
- Since there is no way to turn off the PeopleSoft validation without significantly customizing the application, you would be doubling up on the work done to validate data.

- Some of the parent/child relationships and relationships to edit tables could be constructed as referential constraints. However, there is no guarantee that PeopleSoft will insert parents before children, but delete children before parents. Thus, the constraints would have to be deferred.
- There is no guarantee that batch programs only perform intermediate commits at times when the data is referentially integral. This could generate additional failures.
- PeopleSoft does not handle Oracle errors well. The end user would see an Oracle error rather than a meaningful message in their own language.

What Is the Purpose of a Unique Constraint?

A *unique* or *primary key* constraint specifies the columns whose values will uniquely describe a single row on the table, and prevents more than one row from having that combination of values. A primary key constraint implies NOT NULL constraints on the primary key columns.

There are three ways to create a unique constraint on a table in Oracle:

- Explicitly create a unique index in a separate command.
- Specify a unique constraint within the `CREATE TABLE` command.
- Add a constraint to a table that uses a pre-existing nonunique index.[3]

The three constructions are illustrated in Table 5-1.

Table 5-1. *Unique Indexes vs. Unique Constraints*

`CREATE TABLE tab1`
`(a NUMBER`
`,b NUMBER`
`,c NUMBER`
`,d NUMBER`
`);`
`CREATE UNIQUE INDEX idx1`
`ON tab1(a,b,c);`
	`CREATE TABLE tab1`
	`(a NUMBER`
	`,b NUMBER`
	`,c NUMBER`
	`,d NUMBER`
	`,CONSTRAINT idx1`
	`UNIQUE (a,b,c));`
		`CREATE TABLE tab1`
		`(a NUMBER`
		`,b NUMBER`
		`,c NUMBER`
		`,d NUMBER`
		`);`
		
		`CREATE INDEX idx1`
		`ON tab3(a,b,c,d);`
		`ALTER TABLE tab1`
		`ADD CONSTRAINT idx1`
		`UNIQUE(a,b,c)`
		`USING INDEX idx1;`

3. A unique index on the same columns as a nonunique index will be slightly smaller and slightly more efficient because there is no need to provide for multiple rows with the same key values. However, this construction has the advantage of permitting additional columns in the index other than the unique key columns that might allow the index to satisfy a query without the need to visit the table. This could save an entire index.

The functional effect of these three methods is identical. It is not possible to insert duplicate rows into the table. A unique constraint violation error is produced.

```
ORA-00001: unique constraint (SYSADM.IDX1) violated
```

There is one minor difference. The unique constraint, unless it uses a pre-existing nonunique index, implies a unique index, and you can see entries on both DBA_INDEXES and DBA_CONSTRAINTS. The index will have the same name as the constraint. In contrast, the unique index enforces uniqueness without creating a unique constraint, so there is no entry on DBA_CONSTRAINTS.

This excursion into unique constraints and their enforcement through unique or nonunique indexes is purely theoretical as far as this book goes. PeopleSoft never explicitly specifies any unique or referential constraints; it uses only the unique index syntax.

Record Field Key Attributes

In the Application Designer, the fields in a PeopleSoft record can be given certain key attributes (see Figure 5-1).

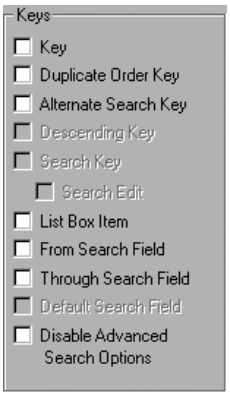


Figure 5-1. *Key attributes*

These key attributes have two main purposes:

- They control the behavior of the PeopleSoft application, and therefore determine some of the SQL that is generated by the PIA and submitted to the database.
- The attributes with *key* in their names cause the Application Designer to generate certain indexes to assist with the performance of the application. I will refer to these as *system-generated* indexes to distinguish them from the additional *custom* or *user-defined* indexes that can be defined by developers.

Field Attributes and Application Behavior

In this section, I am going to look at how key attributes on records control the behavior of the PeopleSoft application. The example used is the JOB_DATA component, which is one of the most heavily used parts of the HCM product.

Record Locator Dialog

In some places in the application, you may simply be able to insert a new key value, but usually you need to find existing data. In this case, before you enter a page in the PIA, you are presented with the record locator dialog (see Figure 5-2), also known as the search dialog.

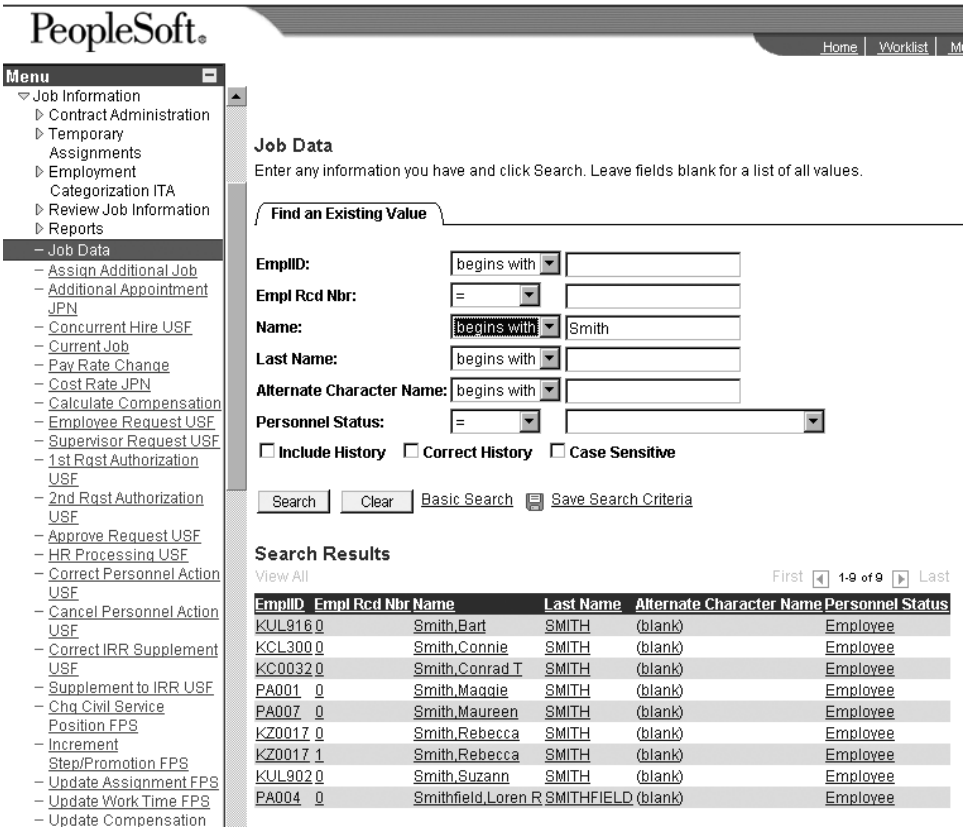


Figure 5-2. *Record locator or search dialog for the JOB_DATA component*

The purpose of the search record is to permit the operator to identify the primary key values for the data to be retrieved into the component. Figure 5-3 shows that EMPLMNT_SRCH_COR is specified in the Application Designer as the search record for the JOB_DATA component.

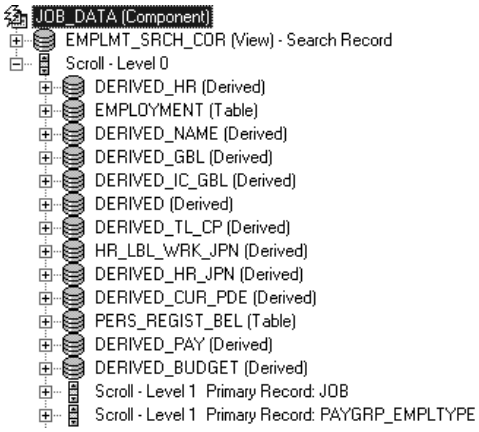


Figure 5-3. *JOB_DATA component structure from the Application Designer*[4]

Figure 5-4 shows the definition of the EMPLMT_SRCH_COR record. EMPLID and EMPL_RCD are key fields. These two fields are also the first two key fields for each of the hierarchy of records, headed by EMPLOYMENT, that are queried by the component. It is the application developer's responsibility to ensure that these keys match; otherwise, the application does not function correctly.

The format of the search dialog page is determined by the key field attributes of the record specified as the component search record. The user can specify criteria for the fields in this dialog to restrict the search results, before eventually selecting one of the key values in the list.

The fields that appear under the Find an Existing Value tab in the search dialog (in Figure 5-3) are:

- Key or duplicate key fields that also have the search attribute
- Alternate search fields

Only fields with key or alternate key attributes can be given the search attribute. ROWSECLASS does not appear in the search dialog shown in Figure 5-2 because although it is also a key field, it does not have the search attribute.

In this example, the search record EMPLMT_SRCH_COR contains a sub-record EMPLMT_SGBL_SBR. A *sub-record* in PeopleTools is simply a group of fields that often appear together on various records, or even other sub-records. They can be added to a record as a group by adding the sub-record. The attributes of the fields in the sub-record are inherited by the record in which the sub-record is used. The record definition shown in Figure 5-4 has been expanded to show the sub-record EMPLMT_SGBL_SBR. EMPLID is a key field in sub-record EMPLMT_SGBL_SBR, hence it is a key on record EMPLMT_SRCH_COR.

4. The "derived" records are effectively groups of working storage variables used by the component.

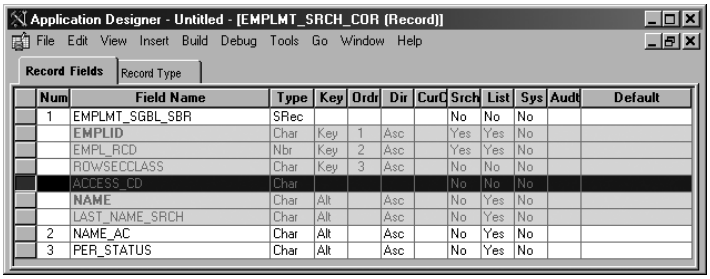


Figure 5-4. *Search record for the JOB_DATA component*

Behind the scenes, the search results were produced from the SQL in Listing 5-1. The SELECT clause was determined by the list of Key and List fields.

Listing 5-1. *SQL trace of the search dialog*

```
PSAPPSRV.3608   1-31132  14.55.58    0.000 Cur#1.3608.HR88 RC=0 Dur=0.000
COM Stmt=
SELECT DISTINCT EMPLID, EMPL_RCD, NAME, LAST_NAME_SRCH, NAME_AC, PER_STATUS
FROM   PS_EMPLMT_SRCH_COR
WHERE  ROWSECCLASS=:1
AND    UPPER(NAME) LIKE UPPER('Smith') || '%' ESCAPE '\'
ORDER BY NAME, EMPLID, EMPL_RCD
PSAPPSRV.3608   1-31133  14.55.58    0.000 Cur#1.3608.HR88 RC=0
Dur=0.000 Bind-1 type=2 length=7 value=HCDPALL
```

There are several points of interest in the search dialog SQL.

- ROWSECCLASS is a field name that triggers special processing by PeopleTools. It is common for search records to be views that join application data with security data. Operators are allocated to security classes. In the preceding example, the primary security class for the user was HCDPALL. So when a search record with a column called ROWSECCLASS with the search attribute was encountered in the search record, the condition on RECSECCLASS in the WHERE clause was automatically added. The query on PS_EMPLMT_SRCH_COR returns the employees that an operator with the specified security class is permitted to see in the application.
- The DISTINCT keyword is always added to the search dialog. The use of DISTINCT makes Oracle retrieve and sort the entire result set before a single row is fetched. Even though the search dialog will only fetch up to the first 300 rows, all the data will be fed to the SQL sort operation. This can be a cause of poor search dialog performance.
- Search records, as in this example, are often views. The DISTINCT keyword can alter the execution plan of the query, taking the tables in the view in the order encountered in the SELECT clause of the view. An ordered hint in the view can suppress undesirable execution plans. However, the scope of hints in views is not limited to the view. If the view is used elsewhere, the hint may cause undesirable effects.

- Note also that when the user typed **Smith** into the name field, the literal `'Smith'` appeared in the generated SQL statement. Every time a user searches for something different, or searches by a different attribute, a different SQL statement is generated and parsed.
- The condition in the where clause on the column NAME uses the LIKE operator because the string entered by the operator was shorter than the field as defined in PeopleTools. If the string entered had been the same length as the field, it would have used `'='`. It is almost inevitable that name searches will use the LIKE operator. This is why PeopleSoft recommends that customers use five-character values for SETID and BUSINESS_UNIT, which are both five-character fields.
- This query was case-insensitive, so PeopleSoft searched for `UPPER(NAME) LIKE UPPER('Smith')`. This disables the use of the index on NAME on the underlying table and can be another cause of poor performance. There are two options in this situation: either globally disable the use of case-insensitive searching throughout PeopleSoft (set in PeopleTools ➤ Utilities ➤ PeopleTools Options) or create a function-based index on NAME. However, this could conflict with use of Oracle Materialized Views (see Chapter 6).

Component Queries

The search dialog discussed in the previous section provides EMPLID and EMPL_RCD values as a key to query the component.

One of the records queried by the component is JOB. Figure 5-5 shows that this record has four fields defined with the key attribute. These key fields tell PeopleSoft what columns should be specified to uniquely identify a row in the record. With only two fields specified from the search dialog, the PIA expects to retrieve many rows.

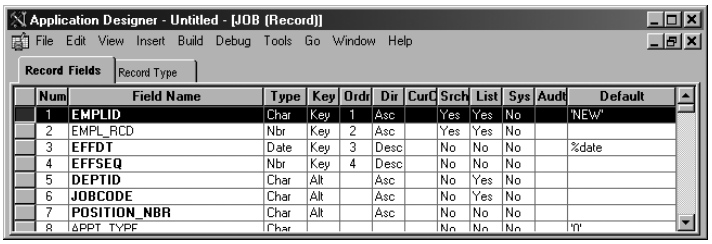


Figure 5-5. *The keys view of the Application Designer, showing a record with a unique key*

On entering the JOB component, the PIA generated the following SQL to retrieve the job history for a particular employee. The two known keys are specified in the where clause, but all the keys were used to generate the ORDER BY clause. EFFDT and EFFSEQ are descending keys on the JOB record, so the query is in descending order on those columns.

```
SELECT EMPLID,
...
FROM PS_JOB
WHERE EMPLID=:1
AND EMPL_RCD=:2
ORDER BY EMPLID, EMPL_RCD, EFFDT DESC, EFFSEQ DESC
```

EFFECTIVE DATE/SEQUENCE PROCESSING

Columns called EFFDT and EFFSEQ automatically trigger special effective date and sequence processing in PeopleTools. Effective dated rows are active from the EFFDT until superceded by a later row. PeopleSoft handles multiple effective dated records on the same day by having an effective sequence in the column EFFSEQ. The maximum effective sequence number is considered to supercede the other records for the same effective date.

As shown in Figure 5-5, the fields have a descending key attribute. When data is loaded into a page, the PIA queries the rows in descending order, and it will only fetch rows until EFFDT is less than or equal to the current date. The last row fetched will be the current effective dated row. The PIA will only query the historical rows if the Include History check box is selected.

In some places, such as the Query tool, PeopleTools will automatically build the subqueries to get the current effective row, or the maximum sequenced rows, as shown in the following example (see also Chapter 11).

```
SELECT A.EMPLID
  FROM PS_JOB B, PS_EMPLMT_SRCH_QRY B1
  WHERE A.EMPLID = A1.EMPLID
    AND A.EMPL_RCD = A1.EMPL_RCD
    AND A1.ROWSECCLASS = 'HCDPALL'
    AND ( A.EFFDT =
        (SELECT MAX(A_ED.EFFDT) FROM PS_JOB A_ED
        WHERE A.EMPLID = A_ED.EMPLID
          AND A.EMPL_RCD = A_ED.EMPL_RCD
          AND A_ED.EFFDT <= SYSDATE)
    AND A.EFFSEQ =
        (SELECT MAX(A_ES.EFFSEQ) FROM PS_JOB A_ES
        WHERE A.EMPLID = A_ES.EMPLID
          AND A.EMPL_RCD = A_ES.EMPL_RCD
          AND A.EFFDT = A_ES.EFFDT) )
```

Field Attributes and System Index Definitions

We have seen so far how the key attributes on a record control some of the functionality of PeopleSoft applications, and hence the SQL that is created by the application. This approach produces indexes that generally result in good performance of the SQL generated by the online application. However, it also produces a large number of indexes. Typically in a PeopleSoft 8.*x* database, the volume of the index extents will be about 5% larger than the volume of the table extents.

The Application Designer can build SQL scripts to build the tables, indexes, views, and triggers in a PeopleSoft database, and it can also build these objects directly. The exact format of the create table and index commands can be customized to include some additional physical parameters (see Chapter 6).

Data Mover, another Windows client program, is PeopleSoft's import/export utility. Therefore, it also creates tables and indexes in the database. The DDL generated during import is written to the export file when it is exported. Data Mover provides a platform-independent environment in which to run scripts and perform certain administrative tasks during installation or upgrade.

The next sections examine how the same information is used to determine what indexes are necessary. I have created a record (see Figure 5-6) and will gradually add various key field attributes to illustrate what indexes are built.

No Keys

Some tables have no key attributes defined on any fields (see Figure 5-6). Therefore, they are created without any system-defined indexes. Typically, these tables either contain only a single row or are used for batch working storage.

There is, of course, nothing stopping the developer from specifying a user index, which is described later in this chapter.

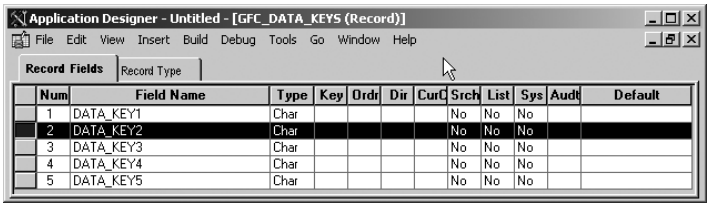


Figure 5-6. *A record with no keys*

Key

The key fields on a record are those that are used to uniquely identify a single row. Therefore, the Application Designer builds a unique index for the key fields.

If a row in a table is queried by the primary key, then that particular key can be found in the primary key index. The row can be found with a single fetch operation, and no further searching is required. This is the most efficient possible use of an index.

Figure 5-7 shows a record in the Application Designer with only two key fields.

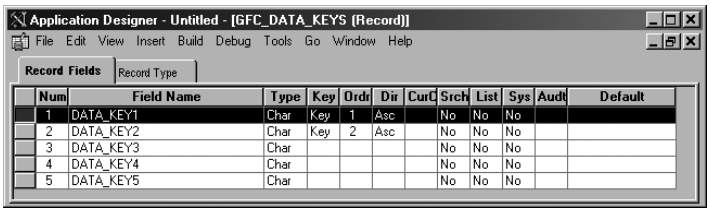


Figure 5-7. *A record with a unique key*

The key fields are used to build a unique index, which can then be seen in the Change Record Indexes dialog (Tools ➤ Data Administration ➤ Indexes) shown in Figure 5-8.

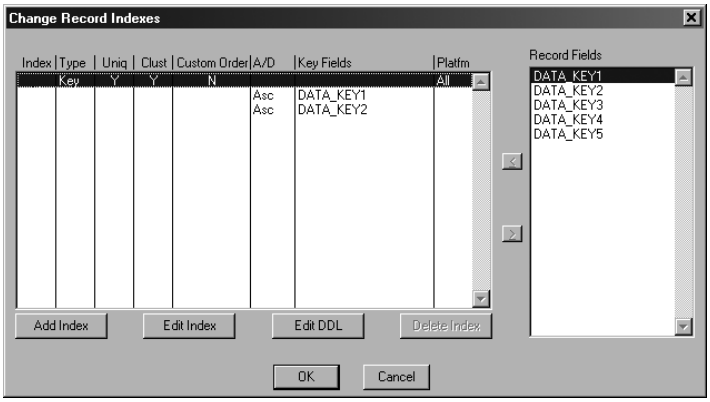


Figure 5-8. *Index definition*

As you can see in Figure 5-8, the Delete Index button is grayed out. System-generated indexes cannot be deleted other than by changing the field attributes. However, changing field attributes changes the application.

The indexes generated by the Application Designer have the following naming convention:

```
INDEX_NAME = 'PS '||<index_id>||<peoplesoft record name>
```

The index ID, described in Table 5-2, indicates which kind of index it is.

Table 5-2. *Index IDs*

Index ID	Description
_	PeopleSoft key index. Implied from the record definition.
1–9	Alternate search key indexes. Implied from the record definition.
#	List index. Implied from the record definition. Only applicable in PeopleTools 7.5 and earlier; not used in PeopleTools 8.
A–Z	User-specified index.

Thus, if the SQLTABLE name has not been set on the record definition, the primary key index generated by the Application Designer has the same name as the table upon which it is built.

The Edit DDL button on the Change Record Indexes dialog leads to the Maintain DDL dialog (not shown), where you can click the View DDL button to see the DDL that will be generated by the Application Designer, as shown in Figure 5-9.

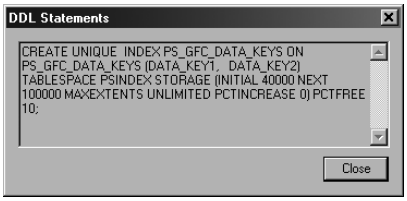


Figure 5-9. *Index DDL*

Descending Key

We have already seen how PeopleSoft uses descending keys to control the order in which data is queried into search dialogs and panels. In Figure 5-10, the first key field is descending.

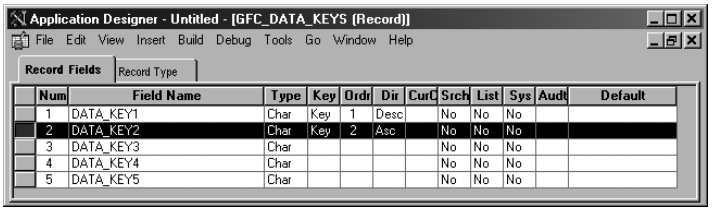


Figure 5-10. *A record with a descending key field*

This appears to be carried forward into the index that is created (see Figure 5-11).

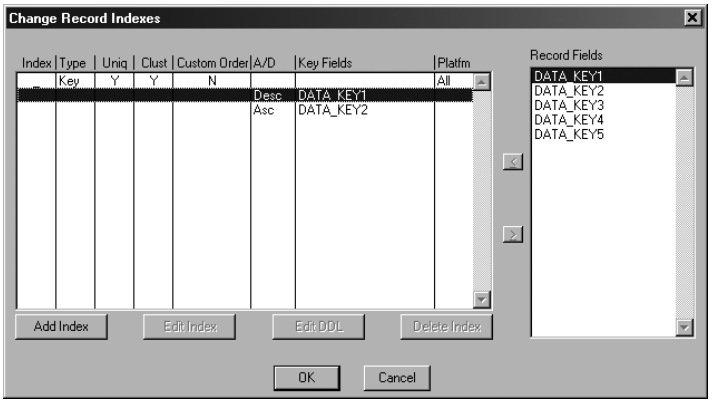


Figure 5-11. *Change Record Indexes dialog for a record with a descending key field*

However, since PeopleTools 8.14, there is no longer a DESC keyword in the index DDL (see Figure 5-12).[5]

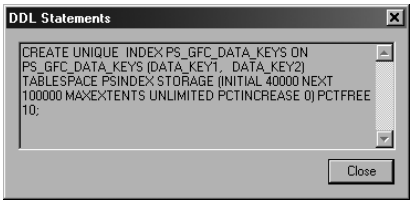


Figure 5-12. *Index DDL for a record with a descending key field*

Duplicate Order Key

Even if a record does not have a field or group of fields that uniquely identify each row, PeopleTools still needs to know by which fields it should query the record. In this case, a duplicate key should be defined, as shown in Figure 5-13.

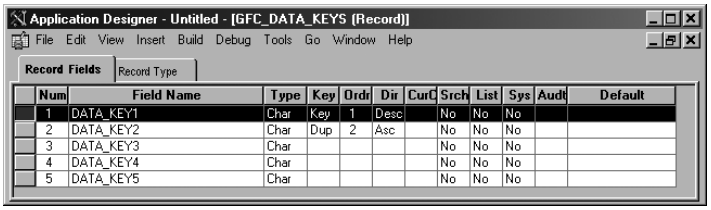


Figure 5-13. *A record with a duplicate key*

Either Key or Duplicate Order Key can be specified in the field properties (see Figure 5-14), but not both.

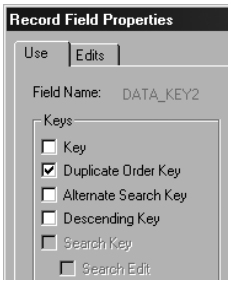


Figure 5-14. *Duplicate Order Key field attribute*

5. Up to PeopleTools 8.14, a descending keyword was included in the create index statement for descending key fields. PeopleSoft eventually removed this functionality because of Oracle Bug #869177, which in most versions of Oracle 8.1.*x* could cause the error ORA-3113 when accessing a descending index.

If any one of the key fields is specified as a Duplicate Order Key, then the key index is not built as a unique index. In Figure 5-15, the unique flag is set to N.

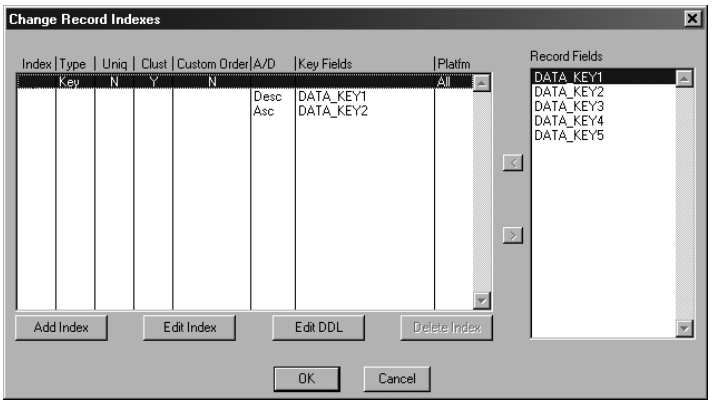


Figure 5-15. *A nonunique key index*

The unique keyword has disappeared from the index DDL in Figure 5-16 (but you might notice that there are two spaces between CREATE and INDEX).

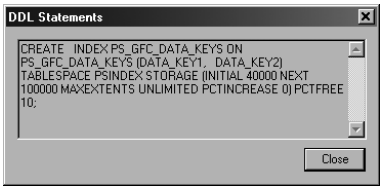


Figure 5-16. *DDL to create a nonunique key index*

Alternate Search Key

Alternate search keys are columns by which the data can be queried from the search dialog, but which are not part of the primary key. Figure 5-17 shows there are two alternate key fields in addition to the two key fields.

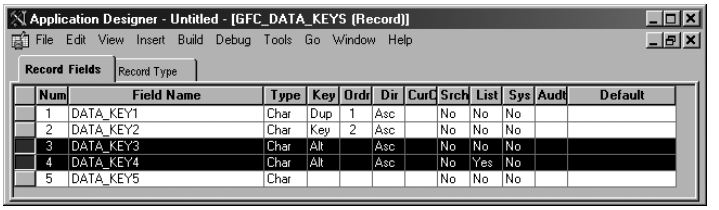


Figure 5-17. *A record with two alternate search keys*

To support search dialog queries on alternate search keys, PeopleSoft automatically creates an index for each alternate search field that is composed of that field followed by the primary (but not the duplicate) key fields (see Figure 5-18).

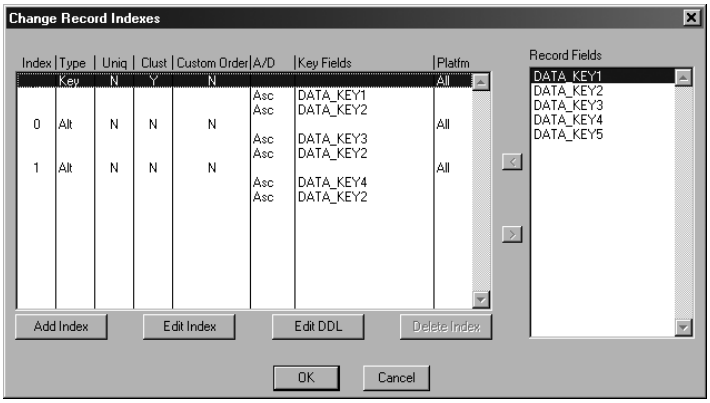


Figure 5-18. *Alternate search key indexes*

Uniqueness of Alternate Search Key Indexes

It follows that if a record has a unique key index, the unique key fields will be included within the columns of each of the alternate search key indexes. Therefore, the composite alternate search keys are themselves unique.

For example, Figure 5-19 shows the indexes on the JOB record. The four key fields also appear in each of the alternate search key indexes.

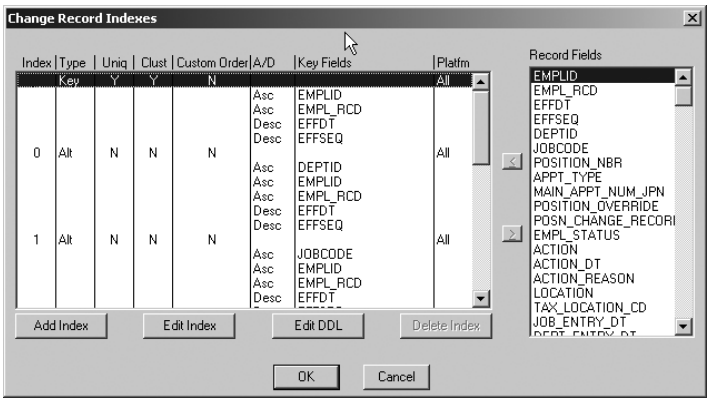


Figure 5-19. *Key and alternate search key indexes*

Unfortunately, PeopleSoft does not create these indexes as unique indexes. This nugget of information can be very valuable to the cost-based optimizer, which has special optimization when it is able to use a unique index.

Even more unfortunate is that the Unique check box in the Edit Index dialog is grayed out on all system-generated indexes, including alternate search key indexes (see Figure 5-20).

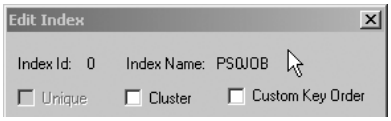


Figure 5-20. *Edit Index dialog*

The DBA could simply edit the index creation scripts generated by the Application Designer to build the indexes as unique. However, the next time the Application Designer is asked to build the create index DDL for that table, it will detect that the index is unique when it should not be, and it will generate a script to drop and re-create the index.[6] There is a significant risk that the unique attribute may get lost in subsequent object build operations.

The only way to make PeopleSoft generate these indexes as unique indexes is to update the PeopleTools table directly as follows:

```
UPDATE psindexdefn a
SET    a.uniqueflag = 1
WHERE  a.uniqueflag = 0
AND    a.indextype = 3
AND    EXISTS(
          SELECT 'x'
          FROM   psindexdefn k
          WHERE  k.recname = a.recname
          AND    k.indexid = '_'
          AND    k.indextype = 1
          AND    k.uniqueflag = 1)
AND    a.recname = '<record name>'
;
```

The Unique check box is then checked, albeit still grayed out (see Figure 5-21), and the index build process creates the desired unique indexes.

6. The same problem occurs if the **BITMAP** variable in the DDL model is set to UNIQUE. The Application Designer will build the correct DDL but will keep rebuilding it because the index is unique and PSINDEXDEFN.UNIQUEFLAG = 0.

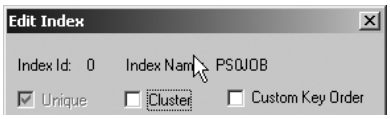


Figure 5-21. *A unique alternate search key index*

■ **Caution** Updating the PeopleTools tables as described in this section is, of course, not supported by PeopleSoft. Nor does this section imply that you should make all the alternate search key indexes unique. As always, follow the principle of minimum intrusion, and make this change only where it serves a demonstrable purpose.

Alternate Search Key Indexes in PeopleTools 7.x

Up to PeopleTools 7.*x*, the alternate search key indexes consisted of the alternate search key, followed by the primary keys, followed by any other alternate search keys or list box fields specified for the record. Thus, all of the alternate search key indexes contained the same columns, but in a different order.

The effect of removing the additional columns and the additional list index (see Figure 5-22) has reduced the volume of indexes in a typical PeopleSoft database by approximately 15%.

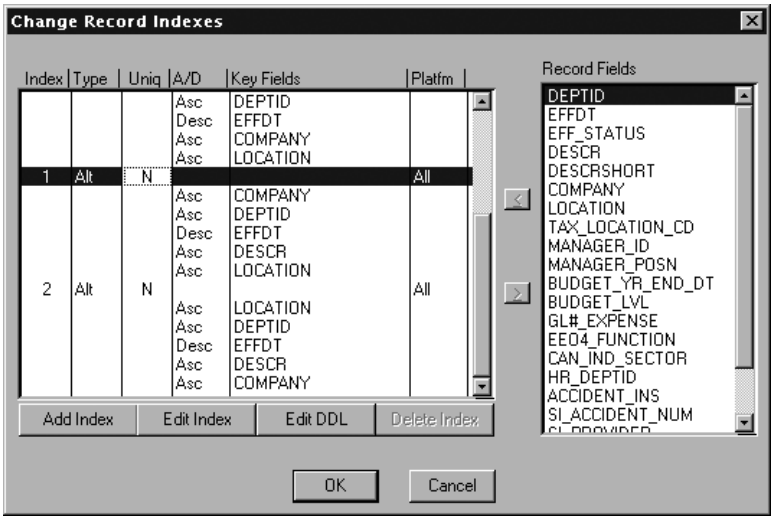


Figure 5-22. *Alternate search keys in PeopleTools 7.x*

List

The list box attribute controls which fields from the component search record appear in the result set (see Figure 5-23).

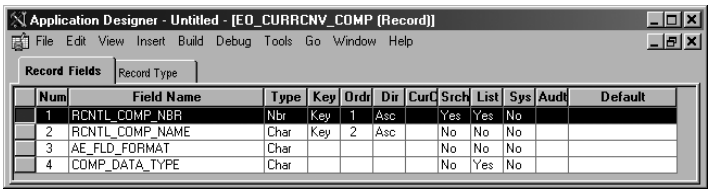


Figure 5-23. *A record with a list field*

In PeopleTools 7.*x* and earlier, a separate index was built that contained each of the list box columns (see Figure 5-24). The intention appeared to be that the list index would supply all the columns for the list box without visiting the table.

The list indexes consumed space in the database, generated additional redo logging, and degraded DML for very little benefit. Most list indexes were hardly ever used. Most of the panel search records are actually views, which cannot be indexed. The rule-based optimizer, which was more likely to be used at this time, is predisposed to use the unique indexes. On the rare occasions when the list index was used, it did not significantly improve performance.

This index is no longer built from PeopleTools 8.

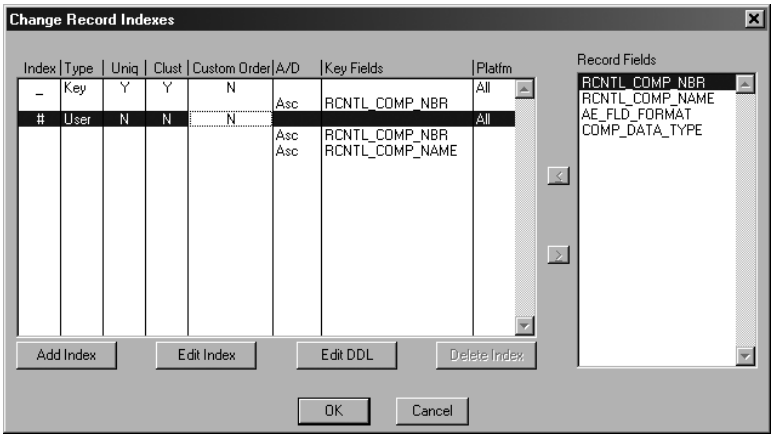


Figure 5-24. *List index in PeopleTools 7.5x*[7]

7. This screenshot is actually from PeopleTools 8.18.10 running on the HR8 SP1 demo database. There was a step in the upgrade process from PeopleTools 7.5 to 8 to remove existing list indexes, because you can't delete them in the Application Designer. Two of them have slipped through to this demo database: EO_CNV_CMP_LANG and EO_CURRCNV_COMP still have list (#) indexes, and you can still build them with the Application Designer!

Custom Key Order

The order of the columns in the system-generated indexes follows the order of columns in the table. From PeopleTools 8, this can be overridden on specific indexes with the Custom Key Order attribute (see Figure 5-25).

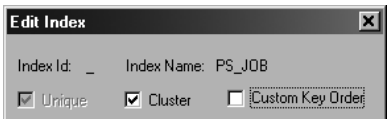


Figure 5-25. *Custom Key Order attribute*

Checking the Custom Key Order box enables drag and drop of columns in the Change Record Indexes dialog (see Figure 5-26) for system-generated indexes in exactly the same way as it has always been possible for user-defined indexes.

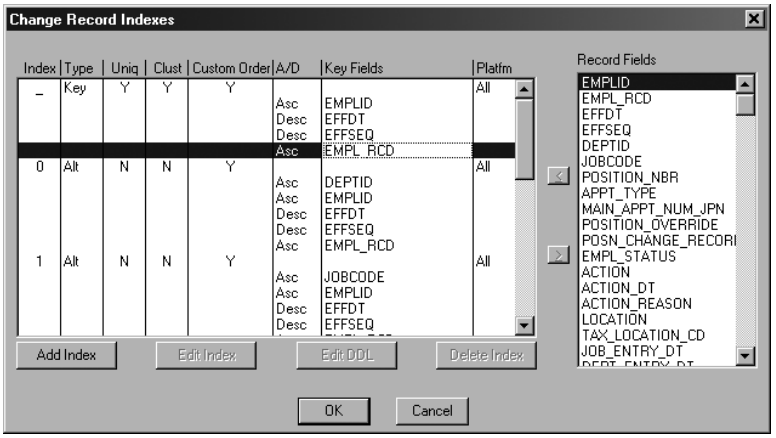


Figure 5-26. *Custom Key Order attribute on a primary key index*

On the JOB record, EMPL_RCD is delivered as the second key column. This column is used to handle the situation that occurs in some companies in which an employee can have two concurrent jobs or may be temporarily posted to another job, perhaps overseas. In companies that do not use this feature, all rows will have an EMPL_RCD of zero. Even when concurrent job processing is used, only a very small proportion of employees will normally have rows with a nonzero EMPL_RCD.

In this example (see Figure 5-26), I have demoted EMPL_RCD to be the last column in every index. I can't remove the column from the index without changing the field attributes, and that will change the application functionality. While the column is still part of the key, it will still be referenced in the queries generated by PeopleTools, so the column still needs to be in the index. Otherwise, there will be occasions when SQL queries will reference the table where previously they would have been fully satisfied from the index alone. So I have relegated the field EMPL_RCD

to the bottom of the index because it does not help the selectivity of the queries. In this particular case, it made sense to do the same to all of the indexes. Custom key order has been separately specified for each system-generated index.

In previous versions, the same effect was achieved by suppressing index creation for a PeopleTools-defined index and adding a user index to replace it. However, this created the potential for problems during upgrades and manual error when specifying the alternate index.

Custom key order cannot be enabled on indexes where the key fields are on sub-records. In this case, the only option is to suppress and replace the index as described previously.

User-Defined Indexes

The automatically generated indexes derived from the field attributes are usually suitable for most of the SQL generated by PeopleTools. However, it is inevitable that there will be some processing—particularly batch processing—where additional indexes will be required.

Additional indexes can be added by the developer via the Add Index dialog shown in Figure 5-27.

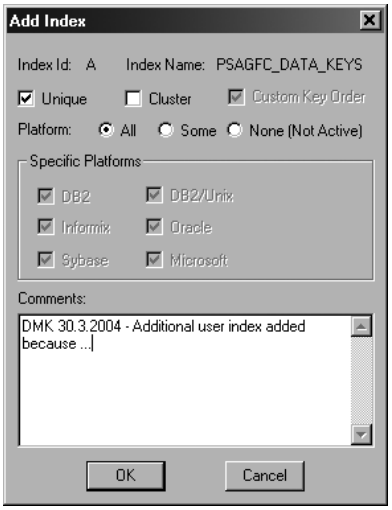


Figure 5-27. *Add Index dialog*

A user index can also be unique, as is the case in Figure 5-27. There is nothing to prevent the same table from having more than one unique index on different columns.

The Cluster option has no meaning when PeopleSoft is installed on an Oracle database. It adds the CLUSTER keyword to the create index DDL for DB2, Microsoft SQL Server, and Sybase.

The Custom Key Order option is grayed out on user indexes (see Figure 5-27) because the choice and order of the columns is already entirely at the developer's discretion.

From PeopleTools 8.4*x*, separate comments can be recorded for user indexes only. User indexes can also be changed without changing the record, can be separately upgraded, and are recorded separately in an Application Designer upgrade project. Since system-generated indexes can only be changed by changing the record, comments can only be put in the comments for the record.

The developer can add columns to the index by selecting from the Record Fields list on the right-hand side of the Change Records Indexes dialog, as shown in Figure 5-28.

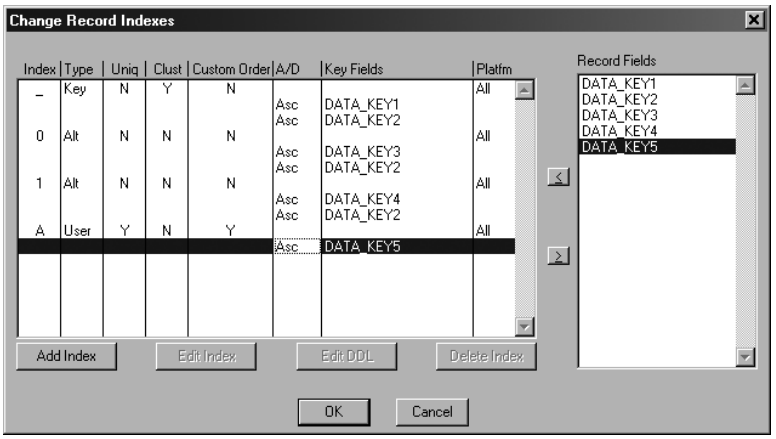


Figure 5-28. *Adding columns to a user index*

Other Index Issues

A number of other issues in PeopleSoft relate to the use of indexes. I detail some of these issues in the following sections.

Views, Keys, and Indexing

The various key attributes can be applied to fields on any type of record in PeopleTools. It follows that keys can be allocated to records that are views (see Figure 5-29), and that PeopleSoft will generate SQL to reference them at run time, much as it does records that are tables.

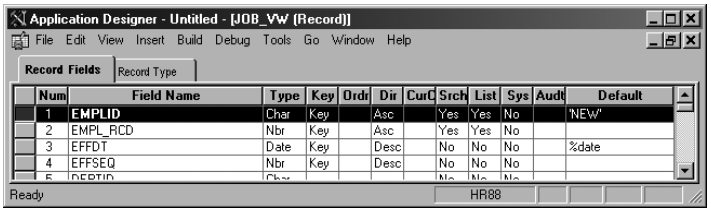


Figure 5-29. *A view with a key*

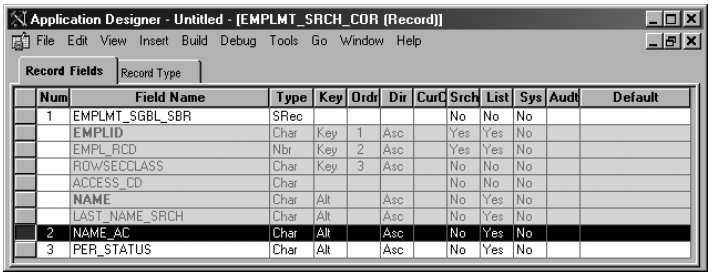
For records that do not correspond to tables, the Tools ➤ Data Administration ➤ Indexes option is not available[8] in PeopleTools 8.

8. In PeopleTools 7.*x* and earlier this option was available, and when a column on a view was given a key attribute, then PeopleTools 7.5 would appear to specify indexes in the Change Record Indexes dialog, although they could not be built.

It is usually beneficial if the columns on the tables referenced by views have indexes to support the allocation of key attributes on the view. If a column of a view is given a particular attribute in PeopleTools, then often the attribute should also exist on the underlying table. PeopleTools will not do this automatically; the developer must do this manually.

The default indexes tend to enable the queries generated by the record locator dialog to obtain data from the index without a further need to read the table. The performance benefit of this is considerable.

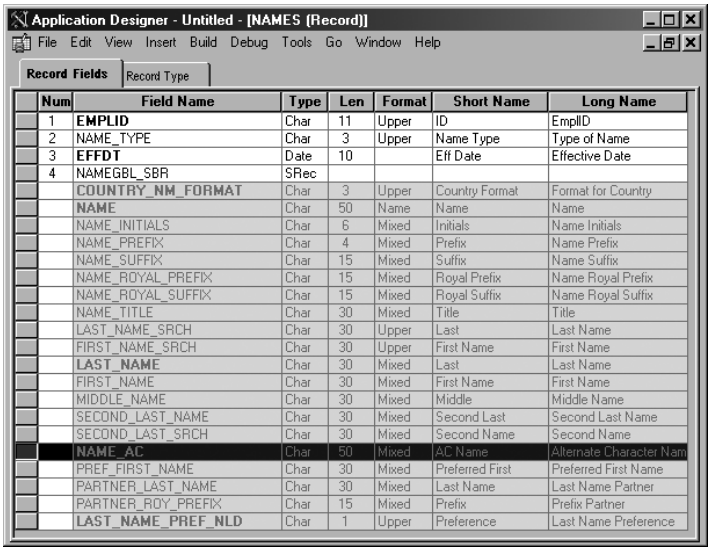
Consider the record EMPLMT_SRCH_COR (see Figure 5-30). It is one of a family of almost identical records that are used in an HCM 8.8 application as component search records. Different countries use different, but similar, views.



Num	Field Name	Type	Key	Ordr	Dir	CurC	Srch	List	Sys	Audt	Default
1	EMPLMT_SGBL_SBR	SRec					No	No	No		
	EMPLID	Char	Key	1	Asc		Yes	Yes	No		
	EMPL_RCD	Nbr	Key	2	Asc		Yes	Yes	No		
	ROWSECCLASS	Char	Key	3	Asc		No	No	No		
	ACCESS_CD	Char					No	No	No		
	NAME	Char	Alt		Asc		No	Yes	No		
	LAST_NAME_SRCH	Char	Alt		Asc		No	Yes	No		
2	NAME_AC	Char	Alt		Asc		No	Yes	No		
3	PER_STATUS	Char	Alt		Asc		No	Yes	No		

Figure 5-30. *Component search record*

In the view EMPLMT_SRCH_COR, the field NAME_AC is an alternate serarch key. Ultimately, this column in the view is derived from the NAME_AC column on the table PS_NAMES (see Figure 5-31).



Num	Field Name	Type	Len	Format	Short Name	Long Name
1	**EMPLID**	Char	11	Upper	ID	EmplID
2	NAME_TYPE	Char	3	Upper	Name Type	Type of Name
3	**EFFDT**	Date	10		Eff Date	Effective Date
4	NAMEGBL_SBR	SRec				
	COUNTRY_NM_FORMAT	Char	3	Upper	Country Format	Format for Country
	NAME	Char	50	Name	Name	Name
	NAME_INITIALS	Char	6	Mixed	Initials	Name Initials
	NAME_PREFIX	Char	4	Mixed	Prefix	Name Prefix
	NAME_SUFFIX	Char	15	Mixed	Suffix	Name Suffix
	NAME_ROYAL_PREFIX	Char	15	Mixed	Royal Prefix	Name Royal Prefix
	NAME_ROYAL_SUFFIX	Char	15	Mixed	Royal Suffix	Name Royal Suffix
	NAME_TITLE	Char	30	Mixed	Title	Title
	LAST_NAME_SRCH	Char	30	Upper	Last	Last Name
	FIRST_NAME_SRCH	Char	30	Upper	First Name	First Name
	LAST_NAME	Char	30	Mixed	Last	Last Name
	FIRST_NAME	Char	30	Mixed	First Name	First Name
	MIDDLE_NAME	Char	30	Mixed	Middle	Middle Name
	SECOND_LAST_NAME	Char	30	Mixed	Second Last	Second Last Name
	SECOND_LAST_SRCH	Char	30	Mixed	Second Name	Second Name
	NAME_AC	Char	50	Mixed	AC Name	Alternate Character Nam
	PREF_FIRST_NAME	Char	30	Mixed	Preferred First	Preferred First Name
	PARTNER_LAST_NAME	Char	30	Mixed	Last Name	Last Name Partner
	PARTNER_ROY_PREFIX	Char	15	Mixed	Prefix	Prefix Partner
	LAST_NAME_PREF_NLD	Char	1	Upper	Preference	Last Name Preference

Figure 5-31. *The NAMES record*

However, on the NAMES record, NAME_AC is not an alternate search key, nor does it appear in a user index, and so the column is not indexed. The column is used to hold a name in an alternate character set. For example, a multinational company operating in Russia has a legal requirement to hold some employee data, including names, in Cyrillic characters. Most HCM customers do not populate this field and frequently remove the alternate search attribute from the view. However, if this column is going to be populated and frequently searched, then it would probably be advantageous to index it on PS_NAMES.

In this situation, you have two customization options:

- The NAME_AC field is on the sub-record NAMEGBL_SBR, which complicates the picture. It could be given the alternate search key attribute, which would then be inherited by EMPLMT_SRCH_COR. However, this would also affect every other record where this sub-record is referenced, and it could change the functionality of the application if any of those records are used as search records on any components. That may or may not be desirable, depending on the circumstances.
- A user index could be explicitly specified on the NAMES record.

Suppressing Index Creation

It is possible to suppress creation of a particular index by the Application Designer for some or all platforms via the Add/Edit Index dialog.

There may be situations in which a field is made an alternate search key for functional reasons, so that the field appears in a search dialog, but it is not appropriate to build an index. An example of this is the CUST_MICR_TBL table in the Financials product (see Figure 5-32).

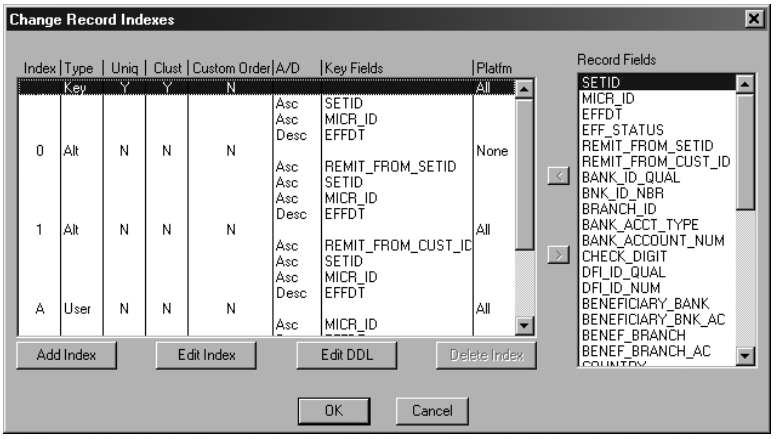


Figure 5-32. *Alternate search index 0 is suppressed.*

PeopleSoft uses this feature to deliver different indexes to different database platforms. In most cases they specify platforms on user indexes, but the feature could be applied to any index. In PeopleTools 8.44, index E on PSPMTRANHIST has been suppressed on DB2 (see the left-hand image in Figure 5-33) and replaced with index H, which is only built on DB2 (see the right-hand image in Figure 5-33).

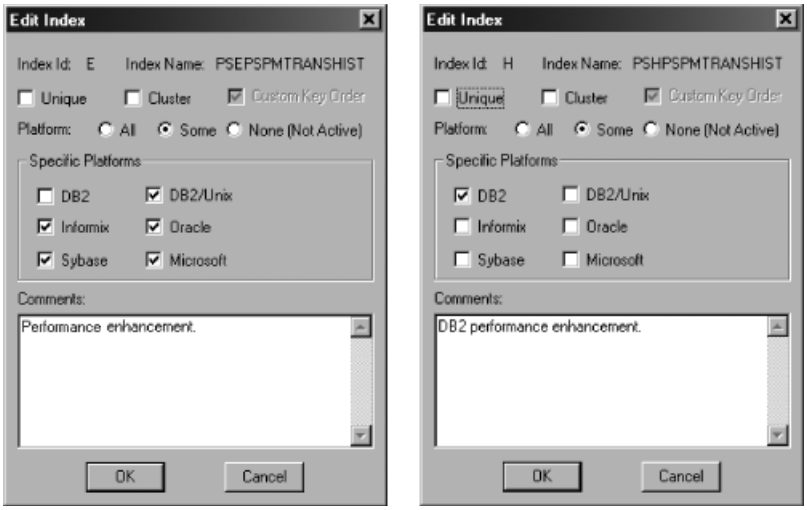


Figure 5-33. *Indexes for only some database platforms*

During performance tuning, you may decide that a particular index is not beneficial or perhaps is actually degrading performance. Instead of deleting the index definition from PeopleTools, you can suppress index creation and add an explanatory comment (see Figure 5-34). Thus, you can easily reverse the change.

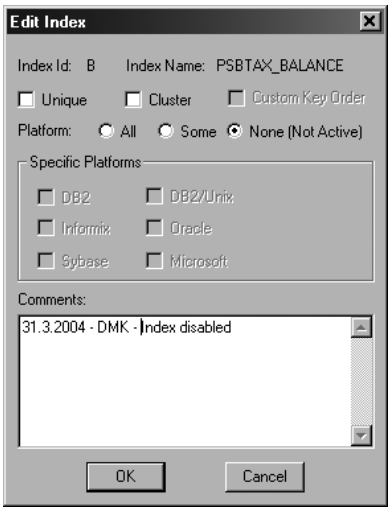


Figure 5-34. *User index disabled on all platforms*

The index will always appear in the Change Record Indexes panel. The setting of the Platform radio button is also reported on this screen.

Indexes and Histograms

In the Financials product in particular, much of the batch processing is status driven. Statuses are usually held in single-character columns. It is often useful to have an index on the status column only in order to enable the batches to find the rows for processing efficiently. There are usually only a few distinct values for each status column. The cost-based optimizer will assume that the distribution of data values is uniform, and so will not use the index.

However, the usual scenario in a system that has been running for a while is that most of the rows will have a completed status, and only a few will have live statuses. Most batch processes will be looking for the live statuses. Oracle will, almost certainly, use indexes on statuses only if there is also a histogram on the column.

The column PROCESS_INSTANCE is also used in some Application Engine processes to select rows for subsequent processing by that process. When the process is initiated by the Process Scheduler, it is given a unique instance number, and it writes that number on the rows to be processed. When the process completes, it sets the PROCESS_INSTANCE column back to zero.[9] An index on the PROCESS_INSTANCE column is often helpful, but again, the column is extremely skewed, and a histogram will help Oracle decide when not to use the index.

Problems can occasionally occur in General Ledger processing on the ACCOUNTING_PERIOD column. This column has values in the range 0 to 999. In a company with 12 monthly periods, there will be only 15 distinct values. Most of the data will be in the range 1 to 12. Values 13 to 997 are not used at all. Values 0, 998, and 999 are used in year-end processing.

On numeric columns, the cost-based optimizer assumes that there are data values evenly distributed between the minimum and maximum values for a column. The optimizer will assume that each value has 1/1,000 of the data and will be likely to use an index where a full scan might be more appropriate. A histogram will help the optimizer make a more informed choice.

Null Columns in PeopleSoft

All character and numeric columns in a PeopleSoft database are defined as NOT NULL columns—PeopleSoft sets them to a single space or zero, respectively, rather than leave them as null.[10] Only date and long columns that are not defined as "required" in the Application Designer are nullable. In the application, the operator must put an entry into a required field.

Table 5-3 shows a comparison of field validation in PeopleSoft and column constraints in Oracle.

9. What about the additional overhead? What about the additional redo logging? What happens if the process crashes before setting the values back to zero? Good questions, but that's what PeopleSoft delivers!
10. From PeopleTools 8.4*x*, if a column is specified as the SYSTEMIDFIELD, its value is populated with a database trigger that is generated by the Application Designer, and the column on the database *is* nullable.

Table 5-3. *Field Validation in PeopleSoft vs. Column Constraints in Oracle*

Data Type	Not Required		Required	
	PeopleSoft	Oracle	PeopleSoft	Oracle
CHARACTER	Blank (single space in database)	Not null	No blanks	Not null
NUMERIC	Zero	Not null	No zeros	Not null
DATE	Blank	Null	Valid date	Not null
LONG	Blank	Null	No blanks	Not null

Prohibiting NULL data values has various implications:

- Table row lengths will be longer than if nulls were permitted, and so tables will be larger. However, chained rows will be less common because the row does not expand as much as numeric data values are filled in.
- All columns need to be specified in INSERT statements because the default column values will default to null, which would result in an error.
- Status and process instance columns cannot be set to null when the row reaches its final status. Thus, the row remains in the index. This makes sparse indexing difficult to implement in PeopleSoft.

SPARSE INDEXING

Oracle indexes do not include rows in which all the indexed columns are null.

```
CREATE TABLE dmk(a number,b number,c number,d number);
CREATE UNIQUE INDEX dmk ON dmk(a,b,c);
INSERT INTO dmk VALUES (null,null,0,1);
INSERT INTO dmk VALUES (null,null,0,2) /*this insert fails, no surprises*/;
INSERT INTO dmk VALUES (null,null,null,3);
INSERT INTO dmk VALUES (null,null,null,4) /*this insert succeeds!*/;
ANALYZE TABLE dmk COMPUTE STATISTICS;

SELECT * FROM dmk;

         A          B          C          D
---------- ---------- ---------- ----------
                              0          1
                                         3
                                         4

SELECT num_rows FROM user_tables WHERE table_name = 'DMK';
```

```
  NUM_ROWS
----------
         3

SELECT num_rows FROM user_indexes WHERE index_name = 'DMK';

  NUM_ROWS
----------
         1
```

There are three rows on the table, but only one in the index. Notice also that there are two duplicate unindexed rows in the table. These features enable a technique sometimes called *sparse indexing* to be used.

Suppose that a table contains many rows, that a process requires only a very few rows, and that there is no easy way to query those few rows without the database performing a full table scan. A possible approach would be to add a column to the table set to 1 for the rows that require to be processed and set to null as they are processed. The column is effectively a "request to process" flag. If an index were built on that column, then it would be used by a query looking for rows due to be processed. The index would remain small because only the rows with the flag set would ever be indexed. As the rows are processed and the flag is set to null, the rows drop out of the index.

However, all numeric and character columns in PeopleSoft are not nullable; hence, the only way to implement a sparse index on a PeopleSoft system is to use a nonrequired date type column.

Summary

The majority of indexes in a PeopleSoft system will be system generated. Their definition is derived directly from the definition of the application. As you learned in this chapter, using PeopleSoft's Application Designer, you can add, remove, or adjust index specifications. You use the same tool to generate and issue DDL to build those tables and indexes.

It is part of the DBA's role to resolve database performance issues, and this may well involve changes to indexing. These changes must be specified and should be implemented via the Application Designer, or they are likely to be lost in a subsequent upgrade.

CHAPTER 6

PeopleSoft DDL

So far I have described how the PeopleSoft and Oracle data dictionaries are related, and how the specification of keys on PeopleSoft records controls both the SQL that is generated by the panel processor and the indexes that are built.

This chapter describes how the Application Designer dynamically builds DDL statements, and in particular where some of the physical attributes are defined and how these models can be enhanced to handle some, but not all, Oracle features. This chapter also covers additional PeopleTools tables that correspond to parts of the Oracle catalogue.

PeopleSoft DDL and the DBA

DBAs will usually have a range of responsibilities, including managing growth and change (capacity planning) and performance monitoring and tuning of the database and application. In an application development environment, the DBA's job may also include responsibility for the design or enhancement of the data model.

This may legitimately lead the DBA to want to adjust the physical properties of existing objects. Common reasons include the following:

- Add, remove, or change an index.
- Move an object to a different tablespace.
- Change the physical parameters (PCTUSED, PCTFREE, etc.).

Consider what happens if the DBA adjusts a table that is then altered by a PeopleSoft patch or some other customization. The upgrade process will include generating and applying a script to alter the table structure, usually by re-creating it.

If the new parameters introduced by the DBA are not placed in the Application Designer, altering the table will effectively erase the work performed by the DBA. Re-creating the table with the original parameters may reintroduce the original problem, or it may cause an error during the upgrade.

If the DBA moves a growing table to a larger tablespace, and that table continues to grow beyond the capacity of the old tablespace, then if that table is re-created by a PeopleSoft ALTER script, it will be re-created in the original tablespace. It will not be possible to repopulate the upgraded table with the application data, and the ALTER script will fail. Worse still, if that space

management error is not caught immediately, it is possible to lose data if the script goes on to drop the original table. If the new tablespace was specified in PeopleTools, then not only would the error not occur during the upgrade, but the DBA wouldn't have had to write the script to move the table in the first place. It could have been generated by the Application Designer.

Therefore, it is very important that object-level changes that the DBA makes to tables and indexes should at least be reflected in the Application Designer, and preferably they will be made in the Application Designer in the first place.

At some PeopleSoft customer sites, the scripts that are produced by the Application Designer must be reviewed by a DBA before they are applied. The DBA then edits them to conform to a local "standard." While it may be appropriate to have the DBA apply the DDL scripts, the sheer number of PeopleSoft tables and views makes this approach virtually impossible and introduces further opportunity for human error. I would suggest that it is better to edit the PeopleSoft DDL models, and set appropriate DDL model defaults and overrides, and restrict manual editing to Oracle features that the Application Designer cannot handle.

DDL Models

The Application Designer and Data Mover utilities build the DDL statements to create the objects referenced by the PeopleSoft application. The formats for these statements are defined as *DDL models* (see Figure 6-1; this window is accessed by selecting PeopleTools ➤ Utilities ➤ Administration ➤ DDL Model Defaults), which are stored in the PeopleTools tables and maintained via the PeopleSoft Internet Architecture (PIA). Each database platform builds objects in a slightly different way. Therefore, PeopleSoft maintains a separate DDL model for each of the database platforms that it supports.

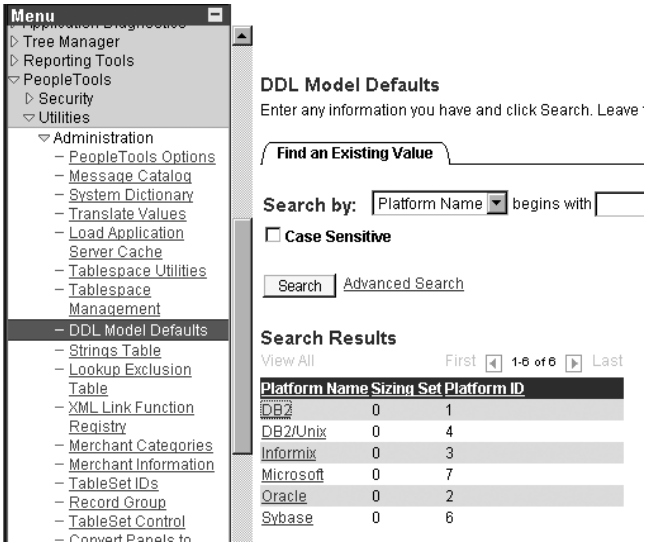


Figure 6-1. *DDL Model Defaults window*

PeopleSoft defines five DDL models:

- Create Table
- Create Index
- Create Tablespace
- Analyze Table Estimate Statistics (new in PeopleTools 8.1*x*)
- Analyze Table Compute Statistics (new in PeopleTools 8.1*x*)

Create Table

Each DDL model consists of a string containing an outline of a SQL command, shown in Figure 6-2, the `CREATE TABLE` command. Various parts of the command are built up using two classes of substitution variables.[1]

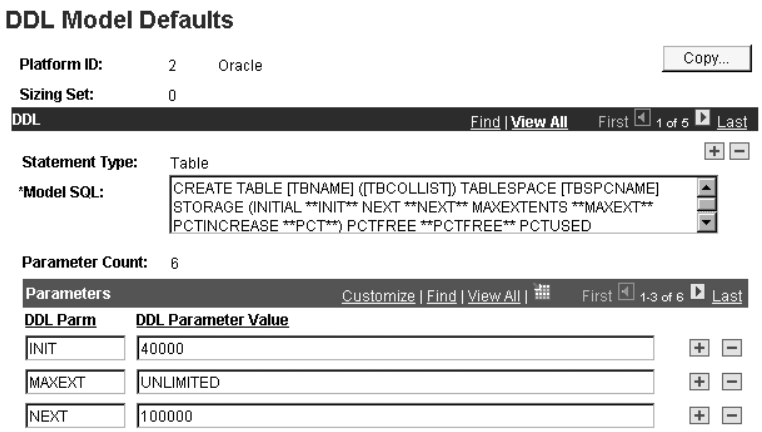


Figure 6-2. *Create Table DDL model*

Variables delimited with square brackets ([]) are PeopleTools internal variables. The values of these variables are determined by code inside the Application Designer and Data Mover utilities, which is not revealed to the user and cannot be changed, except by PeopleSoft's Tools development.

Variables delimited with double asterisks (**) are explicitly declared additional variables. They are defined independently for each DDL model on each RDBMS. Additional variables can be defined through these pages to enable additional features, or existing variables can be removed if they are no longer required. For example, if you are using locally managed tablespaces, the INITIAL and NEXT storage parameters are overridden. The samples shown in this section are the PeopleSoft vanilla definitions. Each of these additional variables has a default value in the model that can be overridden in the object definition in the Application Designer. Listing 6-1 shows the DDL model for creating a table.

1. PeopleSoft doesn't have official terms for these variables. I refer to them as internal and additional DDL variables.

Listing 6-1. *Create Table DDL model*

```
CREATE TABLE [TBNAME] ([TBCOLLIST])
TABLESPACE [TBSPCNAME]
STORAGE (
    INITIAL **INIT**
    NEXT **NEXT**
    MAXEXTENTS **MAXEXT**
    PCTINCREASE **PCT**)
PCTFREE **PCTFREE**
PCTUSED **PCTUSED**;
```

There are four internal variables in the Create Table DDL model that are described in Table 6-1.

Table 6-1. *Create Table Internal Variables*

DDL Model Internal Variable	Description
[DBNAME]	Name of the PeopleSoft database. Not used in any Oracle DDL model. Evaluates to NULL on Oracle.
[TBNAME]	Name of the table: DECODE(PSRECDEFN.SQLTABLENAME, ' ', PS_'||PSRECDEFN.RECNAME, SQLTABLENAME)
[TBCOLLIST]	Column list. Includes the data type of the column, whether it is NOT NULL, and a length-checking constraint if this is a Unicode database.
[TBSPCNAME]	Tablespace in which table is to be built, an attribute of the record (PSRECTBLSPC.DDLSPACENAME). This parameter does not appear in the DDL models for Microsoft SQL Server and Sybase because it has no meaning on those platforms.

PeopleSoft delivers a number of additional variables that correspond to the various physical table parameters (see Table 6-2).

Table 6-2. *Create Table Additional Variables*

Oracle Keyword	Variable	Default Value
INITIAL	**INIT**	40000
NEXT	**NEXT**	100000
MAXEXTENTS	**MAXEXT**	UNLIMITED
PCTINCREASE	**PCT**	0
PCTFREE	**PCTFREE**	10
PCTUSED	**PCTUSED**	80

Column Definitions

The variable [TBCOLLIST] in the Create Table DDL model specifies

- The columns that make up the table
- How the columns are defined
- The length-checking constraints if the database is set to use Unicode

This variable will evaluate to different values on different database platforms. This is coded inside PeopleTools and cannot be altered by the user. It's one of the few examples of platform sensitive coding in PeopleSoft.

Listing 6-2 shows the start of the [TBCOLLIST] variable for the JOB record on a Unicode database on Oracle.

Listing 6-2. *Value of* [TBCOLLIST] *for the JOB record*

```
 EMPLID VARCHAR2(33) NOT NULL CHECK(LENGTH(EMPLID)<=11)
,EMPL_RCD SMALLINT NOT NULL
,EFFDT DATE NOT NULL
,EFFSEQ SMALLINT NOT NULL
,DEPTID VARCHAR2(30) NOT NULL CHECK(LENGTH(DEPTID)<=10)
...
```

The column definitions are based on the field definitions on PSDBFIELD, as described in Chapter 4.

Create Index

From PeopleTools 8, there is only a single DDL model, the Create Index DDL model (see Listing 6-3), that combines the separate DDL models for unique and nonunique indexes in previous versions.

Listing 6-3. *Create Index DDL model*

```
CREATE [UNIQUE] **BITMAP** INDEX [IDXNAME] ON [TBNAME] ([IDXCOLLIST])
TABLESPACE **INDEXSPC**
STORAGE (
    INITIAL **INIT**
    NEXT **NEXT**
    MAXEXTENTS **MAXEXT**
    PCTINCREASE **PCT**)
PCTFREE **PCTFREE**;
```

There are four internal variables in the Create Index DDL model, as described in Table 6-3.

Table 6-3. *Internal Index Variables*

Internal DDL Model Variable	Description
[TBNAME]	Name of the table: DECODE(PSRECDEFN.SQLTABLENAME, ' ', PS_'||PSRECDEFN.RECNAME, SQLTABLENAME)
[IDXNAME]	Name of the index: 'PS'||INDEX_ID||RECNAME
[IDXCOLLIST]	Comma-separated list of columns in the index
[UNIQUE]	Set to UNIQUE for a unique index; otherwise NULL

As with the table DDL, the additional variables (shown in Table 6-4) handle the physical parameters.

Table 6-4. *Create Index Additional Variables*

Oracle Keyword	Variable	Default Value
BITMAP	**BITMAP**	NULL
TABLESPACE	**INDEXSPC**	PSINDEX
INITIAL	**INIT**	40000
NEXT	**NEXT**	100000
MAXEXTENTS	**MAXEXT**	UNLIMITED
PCTINCREASE	**PCT**	0
PCTFREE	**PCTFREE**	10

It is interesting to note that the BITMAP keyword is handled with an additional variable, whereas the UNIQUE keyword is from an internal variable. Both of these were introduced in PeopleTools 8. The reason for the difference in handling is that the UNIQUE keyword is common to `CREATE INDEX` commands on all database platforms, while the BITMAP keyword is specific to Oracle. Hence, the `**BITMAP**` parameter appears only in the DDL model for Oracle.

The specification of tablespace for tables and indexes is handled differently. For tables, it is an internal variable defined on the record until PeopleTools 8.1 or PSRECTBLSPC from PeopleTools 8.4. For indexes, there has always been an additional variable in the DDL model.

Create Tablespace

The Create Tablespace DDL model is only used by Data Mover when it creates an entire database including the tablespaces, either at installation time or if migrating a PeopleSoft database from another RDBMS to Oracle. However, not even the PeopleSoft documented installation process uses this feature. Instead, PeopleSoft supplies SQL scripts (`utlspace.sql` and *xx*`ddl.sql`, where *xx* is the product code; so it is `hrddl.sql` for HCM) to create the default tablespaces, and the installation instructions require you to run these scripts before the Data Mover import. The database import script, generated by Data Mover itself (see Listing 6-4), suppresses tablespace creation with the `SET NO SPACE` command. So, Data Mover normally never gets the opportunity to build a tablespace.

Listing 6-4. *Script to import Data Mover dump during installation*

```
REM - hr88ora.dms
REM - Created by Data Mover 8.44.05 Wed Mar 17 01:24:22 2004
REM -
REM - Database Platform: Oracle
REM - Non-Unicode Database
REM - Selected Character Set: WE8ISO8859P15 - Western European ISO 8859-15 (with Euro sign)
REM - Generate Latin-1 Code
REM -
/
REM - PeopleSoft HRMS Database - US English
/
SET LOG c:\temp\hcengs.log;
SET INPUT D:\ps\hr88\data\hcengs.db;
```

```
SET COMMIT 30000;
SET NO VIEW;
SET NO SPACE;
SET NO TRACE;
SET UNICODE OFF;
SET IGNORE_DUPS;
IMPORT *;
```

Even if PeopleSoft didn't do this, as a DBA I would insist upon creating the tablespaces manually so that all the relevant options can be specified. For example, I would encourage you to implement locally managed tablespaces.

The Create Tablespace DDL model (see Listing 6-5) has only a single internal variable, which is the tablespace name. The tablespace name is also used to make up part of the data file name.

Listing 6-5. *Create Tablespace DDL model*

```
CREATE TABLESPACE [TBSPCNAME]
DATAFILE '**DIR**[TBSPCNAME].DBF'
SIZE **SIZE**
DEFAULT STORAGE (
    INITIAL **INIT**
    NEXT **NEXT**
    MAXEXTENTS **MAXEXT**
    PCTINCREASE **PCT**);
```

The additional variables specify the physical location of the data file and the default storage options (see Table 6-5).

Table 6-5. *Create Tablespace Additional Variables*

Oracle Keyword	Variable	Default Value
DATAFILE	**DIR**	/dir/
INITIAL	**INIT**	40000
NEXT	**NEXT**	100000
MAXEXTENTS	**MAXEXT**	UNLIMITED
PCTINCREASE	**PCT**	0
PCTFREE	**PCTFREE**	10

Analyze Table Estimate Statistics

The two Analyze Table DDL models are new in PeopleTools 8, but they are only supplied for Oracle and DB2 (but not DB2/Unix). They are used to support the `%UpdateStats` meta-SQL in the Application Engine that enables batch processes to regenerate cost-based optimizer statistics during execution. Some COBOL processes also make use of them.

The Analyze Table Estimate Statistics DDL model is as follows:

```
ANALYZE TABLE [TBNAME] ESTIMATE STATISTICS;
```

There are no additional parameters specified.

Collecting Statistics with DBMS_STATS

Oracle has, since at least Oracle8*i*, been recommending the use of the DBMS_STATS package to collect statistics, instead of the `ANALYZE` command. It is possible to get this into the DDL model.

It is permissable to put a PL/SQL block into the DDL model. Unfortunately, `%UpdateStats` in the Application Engine prepends the table name with the PeopleSoft access ID, so the default SQL submitted becomes

```
ANALYZE TABLE SYSADM.PSLOCK ESTIMATE STATISTICS;
```

The workaround is to create a PL/SQL package (see Listing 6-6) that separates the table owner and name, and then calls DBMS_STATS.

Listing 6-6. `wrapper.sql`

```
CREATE OR REPLACE PACKAGE wrapper AS
   PROCEDURE ps_stats (p_full_table_name VARCHAR2, p_estimate NUMBER);
END wrapper;
/

CREATE OR REPLACE PACKAGE BODY wrapper AS
   PROCEDURE ps_stats(p_full_table_name VARCHAR2, p_estimate NUMBER) IS
      l_sep NUMBER;
      l_ownname VARCHAR2(20);
      l_tabname VARCHAR2(30);
   BEGIN
      l_sep := INSTR(p_full_table_name,'.');
      IF l_sep > 0 THEN
         l_ownname := SUBSTR(p_full_table_name,1,l_sep-1);
         l_tabname := SUBSTR(p_full_table_name,l_sep+1);
      ELSE
         l_ownname := 'SYSADM';
         l_tabname := p_full_table_name;
      END IF;

      IF p_estimate = 0 THEN
         sys.dbms_stats.gather_table_stats
            (ownname=>l_ownname
            ,tabname=>l_tabname
            ,estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE
            ,method_opt=>'FOR ALL COLUMNS SIZE REPEAT'
            ,cascade=>TRUE);
      ELSE
         sys.dbms_stats.gather_table_stats
            (ownname=>l_ownname
            ,tabname=>l_tabname
            ,estimate_percent=>p_estimate
            ,method_opt=>'FOR ALL COLUMNS SIZE REPEAT'
            ,cascade=>TRUE);
      END IF;
```

```
   EXCEPTION WHEN OTHERS THEN NULL;[2]
   END ps_stats;
END wrapper;
/
```

■ **Note** The DBMS_STATS.AUTO_SAMPLE_SIZE pseudo-variable was introduced in Oracle9*i*. It allows the database to calculate an appropriate sample size.

The REPEAT option for histograms was also introduced in Oracle9*i*. It collects histograms only on columns that already have histograms.

The packaged procedure can thus be used in the DDL model:

```
BEGIN wrapper.ps_stats('[TBNAME]',0); END;;
```

■ **Note** The PL/SQL block in the DDL model is terminated by the second semicolon.

Analyze Table Compute Statistics

PeopleSoft chose to implement different DDL models for computing and estimating statistics, rather than parameterizing the keyword (as they did with bitmap indexes).

The Analyze Table Compute Statistics DDL model is as follows:

```
ANALYZE TABLE [TBNAME] COMPUTE STATISTICS;
```

Alternatively, the same PL/SQL wrapper can be used to invoke DBMS_STATS.

```
BEGIN wrapper.ps_stats('[TBNAME]',NULL); END;;
```

%UpdateStats

The `%UpdateStats` macro is used within the Application Engine and some COBOL programs in which it is necessary to reanalyze a table in the middle of a batch having loaded a significant amount of data into it.

The format of the macro is `%UpdateStats(record name ,[HIGH/LOW])`. The default is `LOW`. The SQL generated by this macro is sourced from the DDL models, as shown in Table 6-6.

Table 6-6. `%UpdateStats` *Examples on Oracle*

Application Engine Code	Issued SQL (Default DDL Models)
`%UpdateStats(JOB)`	`ANALYZE TABLE PS_JOB ESTIMATE STATISTICS`
`%UpdateStats(JOB,LOW)`	`ANALYZE TABLE PS_JOB ESTIMATE STATISTICS`
`%UpdateStats(JOB,HIGH)`	`ANALYZE TABLE PS_JOB COMPUTE STATISTICS`

2. The exception is required on Oracle8i. If GTTs are analyzed, DBMS_STATS raises an error. This is discussed later in this chapter.

However, commits are suppressed in Application Engine programs when they are called from PeopleCode, because the PIA that executes the PeopleCode will be in the middle of a transaction. When inside a SELECT/FETCH loop, and when the program is restartable, they can be deferred until the loop ends. PeopleSoft knows that the `ANALYZE TABLE` command is DDL and so implies a commit. If commits are deferred, the `%UpdateStats` macro is ignored and the table is not analyzed, as shown in Listing 6-7, which is an extract of an Application Engine trace. A warning is written to the Application Engine trace file if the step trace is enabled (although this trace does have a run-time overhead).

Listing 6-7. *Part of an Application Engine trace*

```
11.30.18 ...(FS_VATUPDFS.BB000.BB000-3) (SQL)
RECSTATS PS_VAT_UPD_BU_TAO LOW
/
11.30.18 UpdateStats ignored - COMMIT required
```

A commit in the wrong place could jeopardize both data integrity and the ability to restart Application Engine programs from the last commit point.

Restricting the implied commit by placing DBMS_STATS inside an autonomous transaction is not an effective workaround. Some of the statistics are not correctly calculated, because the autonomous transaction cannot "see" the uncommitted data.

From Oracle 9.2, dynamic sampling could be used to gather statistics on these tables.

Sizing Sets

It is possible to define different sets of DDL models for the same platform. These are called *sizing sets*. However, they are used only by Data Mover. For example, a particular sizing set can be used to build and import a development database with one sizing set, and a production database with another. The DDL models in different sizing sets can be completely different.

I have never seen this feature used, because most databases are created by cloning other databases, and most objects are created by DDL scripts generated by the Application Designer. PeopleSoft does not deliver multiple sizing sets.

Overriding DDL Model Defaults

The default DDL model parameters can be overridden for specific objects in both the Application Designer and Data Mover. I explain how to do so in the following sections.

Application Designer

The DDL model and additional variables with their default values can be viewed in the Application Designer. The values of the additional variables can be overridden in the Application Designer on each record and index. Figure 6-3 shows the DDL parameter values for the CREATE TABLE statement for the JOB record. The initial extent size of 40000 has been overridden with 957440. This particular override is delivered in the vanilla database.

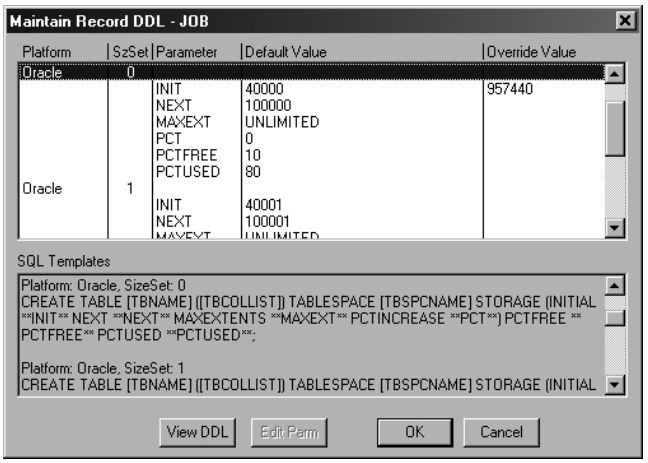


Figure 6-3. *DDL override*

In Oracle, the units for the INIT parameter are bytes unless explicitly specified otherwise. Since the DDL default and override values are character strings, there is nothing to stop these overrides from being specified in K (kilobytes) or M (megabytes) where appropriate, as shown in Figure 6-4.

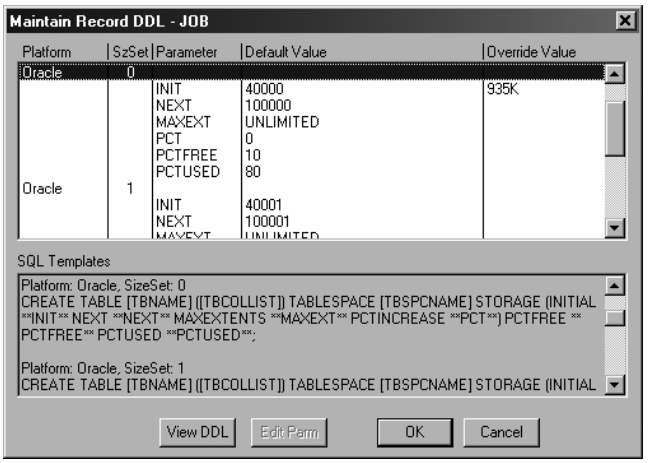


Figure 6-4. *DDL override specified in kilobytes*

The resultant CREATE statement can be viewed in the Application Designer (see Figure 6-5), after the record has been saved. Both sizing sets can be viewed depending on which one is selected, but the Application Designer will only build DDL scripts for sizing set 0.

The variables in the DDL models are simply replaced with their string values to produce a DDL statement.

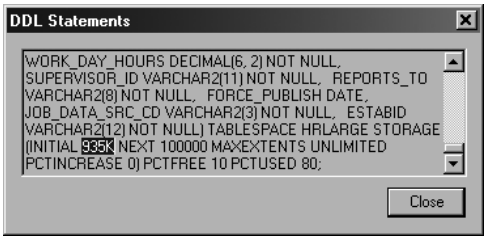


Figure 6-5. *Record DDL with DDL override*

Data Mover

The DDL models and their defaults are encoded into the top of Data Mover export files, and so are read in during an import. This is similar to Oracle's export and import utilities. Although you cannot override the DDL model during a Data Mover import, it is possible to override individual DDL model variables for one or more tables.

```
SET DDL {RECORD | INDEX | UNIQUE INDEX | SPACE} {object_name | *} INPUT parm AS
value;
```

For example, if you are installing a PeopleSoft database in which all the tablespaces are locally managed, the number of extents that will be used by the table when it is created but before any rows are inserted is still determined by the INITIAL, NEXT, and MINEXTENTS parameters in the STORAGE clause (although by default PeopleSoft does not specify MINEXTENTS). The table will have as many extents as are required to allocate at least the same volume as would have been allocated if the table were dictionary managed. But you might decide that you only ever want one extent allocated when the table is built, and then let it expand as it is populated. To do so, you would add the following commands to the top of the Data Mover import:

```
SET DDL RECORD * INPUT INIT AS 2K;
SET DDL INDEX * INPUT INIT AS 2K;
SET DDL UNIQUE INDEX * INPUT INIT AS 2K;
```

PeopleTools Tables

The following PeopleTools tables are used to store the DDL statement models, storage parameters, and override values. Only the additional parameters, delimited by double asterisks (**) in the DDL model, are stored on these tables. The internal variables, delimited by square brackets ([]), are placed in the DDL model, but their values are generated code inside the Application Designer and Data Mover.

PSDDLMODEL (or PS_DDLMODEL_VW): DDL Model Statement

The PSDDLMODEL table, shown in Table 6-7, defines the SQL statement for the DDL model.

Table 6-7. *PSDDLMODEL*

Field Name	Description
STATEMENT_TYPE	Statement type 1 = Table 2 = Index 3 = Tablespace 4 = Analyze Table Estimate 5 = Analyze Table Compute
PLATFORMID	Platform ID 0 = SQLBase 1 = DB2 2 = Oracle 3 = Informix 4 = DB2/Unix 5 = ALLBASE (not supported from PeopleTools 8.*x*) 6 = Sybase 7 = Microsoft 8 = DB2/400 (not supported from PeopleTools 8)
SIZING_SET	Sizing set
PARMCOUNT	The number of parameters defined in this model
MODEL_STATEMENT	A long column that holds the model SQL statement as seen in the panels

PSDDLDEFPARMS (or PS_DDLDEFPARMS_VW): DDL Model Parameter

This PSDDLDEFPARMS table, shown in Table 6-8, defines the default values for the additional parameters in the DDL model. It contains one row for each parameter in each DDL model.

Table 6-8. *PSDDLDEFPARMS*

Field Name	Description
STATEMENT_TYPE	Statement type (see PSDDLMODEL)
PLATFORMID	Platform ID (see PSDDLMODEL)
SIZING_SET	Sizing set (see PSDDLMODEL)
PARMNAME	DDL parameter name
PARMVALUE	DDL parameter value

PSRECDDLPARM: Record DDL Parameter

The PSRECDDLPARM table, shown in Table 6-9, holds the DDL parameter overrides for CREATE TABLE statements. It contains one row for each override on a table entered in the Application Designer. If a parameter is not overridden, there will not be a row on this table.

Table 6-9. *PSRECDDLPARM*

Field Name	Description
RECNAME	PeopleTools record name
PLATFORMID	Platform ID (see PSDDLMODEL)
SIZINGSET	Sizing set (see PSDDLMODEL)
PARMNAME	DDL parameter name (see PSDDLDEFPARMS)
PARMVALUE	DDL parameter value

PSTBLSPCCAT: Tablespace Catalogue

The PSTBLSPCCAT table, shown in Table 6-10, was introduced in PeopleTools 8.4. It provides a list of available tablespaces, and it contains at least one row for every tablespace referenced by PeopleTools.

Table 6-10. *PSTBLSPCCAT*

Field Name	Description
DDLSPACENAME	Name of the tablespace.
DBNAME	Name of the database in which the tablespace occurs. If blank (single space), then the tablespace is in all databases.
TSTYPE	Tablespace type. A = Audit G = Large L = LOB R = Regular S = Small
DBXTSTYPE	DB2 Unix tablespace type.
COMMENTS	Comment.

PSRECTBLSPC: Record Tablespace Allocation

From PeopleTools 8.4, the tablespace for a table is no longer defined on the record definition on PSRECDEFN—it has been moved to PSRECTBLSPC (see Table 6-11). This table contains at least one row for every table. A record can be assigned to different tablespaces on different databases.

Table 6-11. *PSRECTBLSPC*

Field Name	Description
DDLSPACENAME	Name of the tablespace.
DBNAME	Name of the database. If blank (single space), then the record appears in this tablespace in all databases.
RECNAME	PeopleTools record name.
DBTYPE	Database platform. It has the same values and meanings as PLATFORMID (see PSDDLMODEL).
TEMPTBLINST	

PSIDXDDLPARM: Index DDL Parameters

This table (see Table 6-12) holds the DDL parameter overrides for CREATE INDEX statements. It contains one row for each override on an index entered in the Application Designer.

Table 6-12. *PSIDXDDLPARM*

Field Name	Description
RECNAME	PeopleTools record name
INDEXID	Index identifier _ = Primary key index # = List columns index (no longer used in PeopleTools 8) 0–9 = Alternate search key indexes A–Z = User-specified indexes
PLATFORMID	Platform ID (see PSDDLMODEL)
SIZINGSET	Sizing set (see PSDDLMODEL)
PARMNAME	DDL parameter name (see PSDDLDEFPARMS)
PARMVALUE	DDL parameter value

DDL Model Enhancements

The flexibility of the DDL model provides many possibilities. The overriding principle is that PeopleSoft should generate the correct DDL in the first place, although there are some exceptional situations in which this cannot be done.

In this section, I will cover a number of possible DDL model enhancements, such as the ability to add physical parameters, to put multiple commands into a DDL model, and to implement Global Temporary Tables.

Additional DDL Parameters

Not only can you alter the values of the existing DDL parameters, but you can also add additional variables. PeopleSoft added PCTFREE and PCTUSED to the table and index models from PeopleTools 8.

The following could also be specified, although they are, perhaps, less likely to be used. I am not recommending that you add any or all of these, but if you need to, you can.

- MINEXTENTS
- FREELISTS
- BUFFER_POOL
- CACHE/NOCACHE
- LOGGING/NOLOGGING (RECOVERABLE/UNRECOVERABLE prior to Oracle8*i*)
- INITRANS
- PARALLEL/NOPARALLEL

Figure 6-6 shows how the DDL model and default could be set up.

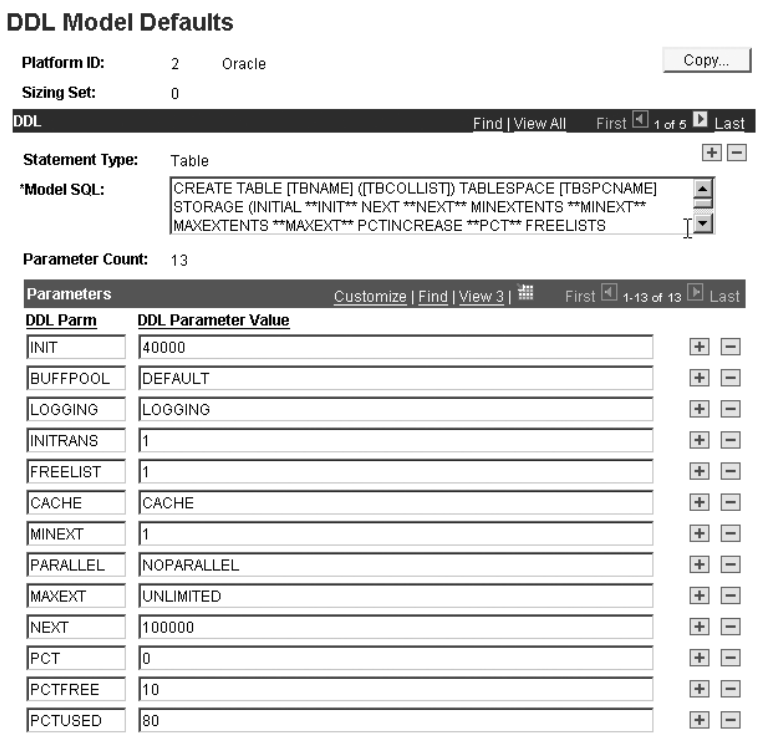


Figure 6-6. *Enhanced DDL model default*

All that is needed is to edit the DDL statement in the DDL Model Defaults component for the relevant statement and to insert the extra parameters in the scroll, as shown in Listing 6-8.

Listing 6-8. *Enhanced Create Table DDL model*

```
CREATE TABLE [TBNAME] ([TBCOLLIST]) TABLESPACE [TBSPCNAME]
STORAGE (INITIAL **INIT** NEXT **NEXT** MINEXTENTS **MINEXT** MAXEXTENTS
**MAXEXT** PCTINCREASE **PCT** FREELISTS **FREELIST** BUFFER_POOL **BUFFPOOL**
PCTFREE **PCTFREE** PCTUSED **PCTUSED**
;
```

If any variables in the DDL model do not match any of either the internal or additional variables, you will get the following error message:

```
Error: Inconsistent statement type Create Table, platform 2, sizing set 0,
reason code 1. (76,39)
```

If you do create additional variables, you may wish to customize `settable.sqr` and `setindex.sqr`. These SQR processes feed the real values of the Oracle physical storage variables back from USER_TABLES and USER_INDEXES into PeopleTools. I discuss the issues surrounding synchronizing physical parameters in the data dictionaries later in this chapter.

■ **Note** The number of extents allocated when the table is created in a locally managed tablespace will be the number that provides the same space as if the table had been created in a dictionary managed tablespace. If you choose to add MINEXTENTS to the DDL models for tables and indexes, and if you are using locally managed tablespaces, then set the default values for INITIAL and NEXT in the DDL model to the uniform extent size of the tablespace. Thus, the actual number of extents will match the number specified in the STORAGE clause.

At the very least, following this procedure will make space calculations easier. It will be helpful if you also feed back the MIN_EXTENTS from USER_TABLES into PeopleTools. This requires customization of `settable.sqr` and `setindex.sqr`, or an alternative script.

If the locally managed tablespace uses automatic extent sizing, set INITIAL and NEXT to the minimum allocation size (usually 64KB), but the number of extents may not match.

You can see the effect of the changes in the Maintain Record DDL dialog (see Figure 6-7) in the Application Designer (after you have restarted it and the application server also if you are developing in three-tier mode).

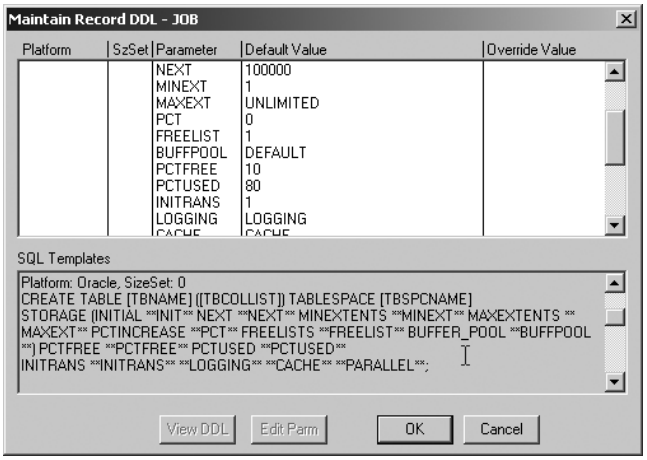


Figure 6-7. *Additional parameters can be overridden.*

You can also see the effect on the DDL statement in the DDL window shown in Figure 6-8.

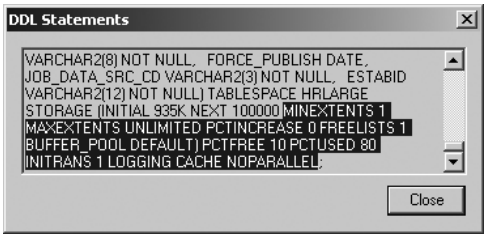


Figure 6-8. *Enhanced DDL statement*

■ **Caution** If you do customize the DDL models, be aware that during the PeopleTools upgrade process on an Oracle database is an instruction to run the Data Mover script `ddlora.dms` (there is a similar script for each database platform), which sets up the default DDL models. This will delete all sizing sets and reinstate the vanilla configuration. Any changes you make to the DDL model or any additional sizing sets will be lost. Therefore, you may choose not to run this script.

Fewer DDL Parameters

Locally managed tablespaces were introduced in Oracle8*i* and have become the default in Oracle9*i*, although PeopleSoft's delivered scripts only create dictionary managed tablespaces.

If you choose to use locally managed tablespaces, you might decide it would be better if the PeopleSoft DDL did not include INITIAL, NEXT, or MAXEXTENTS. It may look simple to just delete them from the DDL model in the PIA, but this will cause problems. If you completely delete the variables and the default values, then the processes to feed sizing information back into PeopleTools (`settable.sqr` or `setindex.sqr`) will create overrides for which there are not defaults, and so cause errors to be generated in the Application Designer when viewing or generating DDL. It appears that all the additional variables that have defined defaults in the model must also appear in the DDL model; otherwise, this will also cause errors in the Application Designer.

I would suggest commenting out the unwanted parameters in the DDL model string, as shown in Listing 6-9.

Listing 6-9. *Create Table DDL model with physical storage parameters commented out*

```
CREATE TABLE [TBNAME] ([TBCOLLIST]) TABLESPACE [TBSPCNAME]
/*STORAGE (INITIAL **INIT** NEXT **NEXT** MINEXTENTS **MINEXT** MAXEXTENTS
**MAXEXT** PCTINCREASE **PCT**)*/
PCTFREE **PCTFREE** PCTUSED **PCTUSED**;
```

Multiple Commands

It is possible to put multiple commands into a DDL model, and all of the commands will be executed by the Application Designer or will be generated in the build script. The sections that follow present two examples.

Cost-Based Optimizer Statistics

If an upgrade changes a record, the ALTER script built by the Application Designer will re-create the table and its indexes. Statistics will need to be generated on the new objects. Listing 6-10 shows how an ANALYZE command can be added to the Create Index DDL model.

Listing 6-10. *Create Index DDL model with an additional* ANALYZE *command*

```
CREATE [UNIQUE] **BITMAP** INDEX [IDXNAME] ON [TBNAME] ([IDXCOLLIST])
TABLESPACE **INDEXSPC** STORAGE (
    INITIAL **INIT** NEXT **NEXT**
    MAXEXTENTS **MAXEXT** PCTINCREASE **PCT**)
PCTFREE **PCTFREE**;
ANALYZE TABLE [TBNAME] **SAMPLE** STATISTICS;
```

The sample size for estimating the statistics has also been coded as an additional variable in the DDL model (see Figure 6-9).

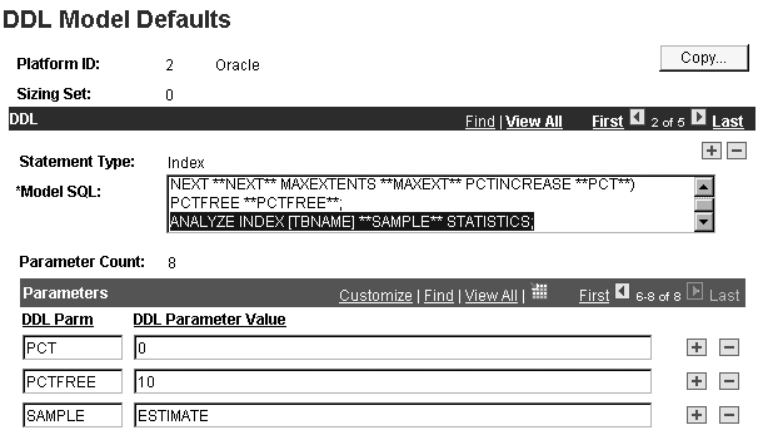


Figure 6-9. *Additional* ANALYZE *command*

There is little point in adding an ANALYZE command to the Create Table model. If the table has just been created, then it is also empty. If it is being altered by re-creation, only the first command in the DDL model, the CREATE TABLE command, will be generated. However, the entire Create Index model is generated for each active index. Therefore, I have added a command to the Create Index DDL model to analyze the table. This will also analyze all of its indexes on the table.

It does mean that the table and previously built indexes will be analyzed repeatedly after each index is built. This does waste some time, but at least the table is analyzed. So long as the statistics are estimated on a small sample, this is unlikely to be an unacceptable overhead.

Grants

Sometimes it is necessary to allow other Oracle users to access tables in the PeopleSoft schema. This might be to allow access to an interface process or to developers. Usual Oracle practice would be to create database roles, grant the privileges to the roles, and grant the roles to the users (see Listing 6-11). The roles could even be parameterized, with different tables having different roles.

Listing 6-11. *Create Table DDL model with additional grants*

```
CREATE TABLE [TBNAME] ([TBCOLLIST])
TABLESPACE [TBSPCNAME]
STORAGE (
    INITIAL **INIT** NEXT **NEXT**
    MAXEXTENTS **MAXEXT** PCTINCREASE **PCT**)
PCTFREE **PCTFREE** PCTUSED **PCTUSED**;
GRANT SELECT ON [TBNAME] TO **READROLE**;
GRANT SELECT, INSERT, UPDATE, DELETE ON [TBNAME] TO **WRITROLL**;
```

If the grants were not built into the DDL model, they would have to be created for new tables, as shown in Figure 6-10.

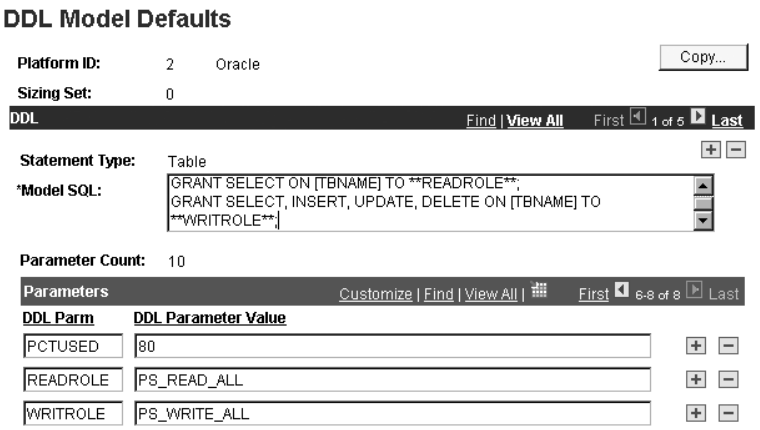


Figure 6-10. *Additional commands in the DDL model*

There is a limitation with this technique. When altering a table by re-creation, only the first command in the Create Table model is generated in the `psbuild.sql` script. When the table is altered by re-creation, the grants will be lost. The DBA will still have to check that all the grants exist.

Global Temporary Tables

It is possible to make PeopleTools implement Global Temporary Tables, although the solution is a bit untidy.

The problem is that you cannot specify any physical parameters in the DDL statements, but in the Application Designer you cannot add or remove parameters from the DDL model for some records and not others, nor can you specify a different sizing set for a particular record. Therefore, the existing physical parameters in the PeopleSoft DDL model must be hidden inside comments.

The DDL model could be amended as shown in Listing 6-12 with three new parameters. The default values for the new parameters are all blank.

Listing 6-12. *DDL model to create global temporary tables and indexes*

```
CREATE **GLOBTEMP** TABLE [TBNAME] ([TBCOLLIST])
**GTSPCOM1**
TABLESPACE [TBSPCNAME]
STORAGE (
    INITIAL **INIT** NEXT **NEXT**
    MAXEXTENTS **MAXEXT** PCTINCREASE **PCT**) PCTFREE **PCTFREE** PCTUSED
**PCTUSED**
**GTSPCOM2**;

CREATE [UNIQUE] **BITMAP** INDEX [IDXNAME] ON [TBNAME] ([IDXCOLLIST])
**GTSPCOM1**
TABLESPACE **INDEXSPC**
STORAGE (
    INITIAL **INIT** NEXT **NEXT**
    MAXEXTENTS **MAXEXT** PCTINCREASE **PCT**)
PCTFREE **PCTFREE**
**GTSPCOM2**;
```

Three new additional variables are set out in Table 6-13 that are needed in both the Create Table and Create Index DDL models.

Table 6-13. *New Variables to Handle Global Temporary Objects in the DDL Model*

DDL Model Parameter	DDL Override (for Global Temporary Table)	Description
GLOBTEMP	`GLOBAL TEMPORARY`	Introduce GLOBAL TEMPORARY keyword in the CREATE TABLE statement

NOSPCOM1	For tables:
	`ON COMMIT PRESERVE ROWS /*`
	For indexes:
	`/*`
		Variable to start comment at the beginning of the STORAGE clause

NOSPCOM2	`*/`	Variable to close comment at the end of the STORAGE clause

The Application Designer generates the SQL shown in Listing 6-13. The storage parameters are still generated, but they are now within multiline comments.

Listing 6-13. *DDL to create a global temporary table and index*

```
--
-- WARNING:
--
-- This script should not be run in Data Mover. It may contain platform
-- specific syntax that Data Mover is unable to comprehend. Please use the
-- SQL query tool included with your database engine to process this script.
--
DROP TABLE PS_GP_CANC_WRK
/
CREATE GLOBAL TEMPORARY TABLE PS_GP_CANC_WRK (EMPLID VARCHAR2(11) NOT
 NULL,
   CAL_RUN_ID VARCHAR2(18) NOT NULL,
   GP_PAYGROUP VARCHAR2(10) NOT NULL,
   CAL_ID VARCHAR2(18) NOT NULL) ON COMMIT PRESERVE ROWS /* TABLESPACE
 GPAPP STORAGE (INITIAL 40000 NEXT 100000 MAXEXTENTS UNLIMITED
 PCTINCREASE 0) PCTFREE 10 PCTUSED 80 */
/

COMMIT
/
CREATE UNIQUE  INDEX PS_GP_CANC_WRK ON PS_GP_CANC_WRK (EMPLID,
   CAL_RUN_ID,
   GP_PAYGROUP,
   CAL_ID) /* TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10 */
/

COMMIT
/
```

Although you can implement a Global Temporary Table in the Application Designer, I would recommend handling this type of table manually because of the potential for making errors in the override values.

Other Issues with Global Temporary Tables

Attractive as Global Temporary Tables are, there are some problems to consider before implementing them.

It is not possible to analyze Global Temporary Tables in Oracle8*i*. The `ANALYZE` command completes without raising an error message, but it does not collect any statistics. However, it is possible to set statistics with specific values with the DBMS_STATS package.

In Oracle8*i*, DBMS_STATS.GATHER_STATISTICS will raise an error if it attempts to use a Global Temporary Table.

```
ORA-20000: Gathering statistics for a temporary table is not supported
ORA-06512: at "SYS.DBMS_STATS", line 4481
ORA-06512: at line 2
```

From Oracle9*i*, it is possible to collect optimizer statistics for a Global Temporary Table with either the `ANALYZE` command or DBMS_STATS, but there is only a single place for statistics in the Oracle catalogue, so the statistics are common to all sessions that reference a table. Therefore, analyzing a Global Temporary Table in one session could impact another.

From PeopleTools 8.4, the Application Engine is a server within a Tuxedo Process Scheduler domain (see Chapter 14). Like other PeopleSoft application server processes, it makes a persistent connection to the database. Thus, one session may run several Application Engine programs. Global Temporary Tables must be defined to preserve rows on commit. When they are referenced by a session, they will consume physical space in the temporary segment, unless truncated, until the database session terminates, which will happen only when the Process Scheduler domain is shut down.

Preserve on commit Global Temporary Tables cannot be dropped until all the database sessions that have referenced them have terminated. Particularly in a development environment, you may have to shut down the Process Scheduler in order to alter Global Temporary Tables.

Some PeopleSoft Application Engine programs truncate temporary working storage tables at the start and/or the end of a process. If you truncate a Global Temporary Table with the REUSE STORAGE option, the command will complete without raising an error, but it will not truncate the table—it doesn't do anything at all.[3]

```
TRUNCATE TABLE my_gtt REUSE STORAGE
```

Limitations of PeopleSoft DDL Models

The Application Designer is principally a tool for PeopleSoft developers, not DBAs. A DBA should use it as far as it is reasonably possible. However, certain Oracle features cannot reasonably be introduced and managed via the PeopleSoft DDL models, as I explain in this section.

Partitioned Tables

The Application Designer cannot generate partitioned tables because of the variable length of the PARTITION clause, depending upon the number of partitions. It would be possible to add a number of additional variables to the DDL model and type the PARTITION clauses in as a number of overrides, but this would be very hard to manage.

3. I reproduced this on Oracle 8.1.7.4, 9.2.0.5, and 10.1.0.2.

CASE STUDY: PARTITIONED AND GLOBAL TEMPORARY TABLES

Manually managing objects that the Application Designer cannot is an issue in Global Payroll (GP) systems. To enable efficient parallel batch processing, there are approximately 30 tables that all need to be similarly range partitioned, the largest of which are also hash subpartitioned. In addition, the range partitions are placed in different tablespaces; the number of partitions can change, as well as the range partition boundary values; and patches can add additional columns. There are more than 50 other tables that have been made Global Temporary Tables. It is simply not possible to implement all this in the Application Designer in a manageable fashion.

The solution was to write a PL/SQL script that used the information in the PeopleTools tables and some of the GP application tables to build SQL scripts that would build the tables and indexes, including the extra keywords and partitioning clauses.[4] The result was a utility script that could be easily maintained. When changes are made in either the GP application or its configuration that require the tables to be altered, the script is used to generate the DDL instead of the Application Designer.

The overhead of developing the utility was set against the time spent manually editing and debugging the Application Designer–generated DDL scripts. The cost of making configuration changes that would improve performance was greatly reduced because the manual editing of DDL scripts was removed.

All that is necessary to remember is which tables should be built with the utility and not the Application Designer.

Constraints

Not only is it inappropriate to add constraints to a PeopleSoft database (as discussed in Chapter 5), but they also cannot be added to tables at create time because the syntax is inside the column list, which is generated by the internal PeopleTools variable [TBCOLLIST], which cannot be altered (see Listing 6-14).

Listing 6-14. *PeopleSoft doesn't create a table with a constraint*

```
CREATE  TABLE PS_PERSONAL_DATA
(EMPLID VARCHAR2(11) NOT NULL,
...
   SEX VARCHAR2(1) NOT NULL CONSTRAINT sex_check CHECK SEX IN('M','F','U');
...
```

Function-Based Indexes

We saw in Chapter 5 that searches on character fields in the record locator dialog are, by default, case-insensitive. When PeopleTools builds the query for the search dialog, it adds the UPPER() functions to the query that it constructs, as shown in Listing 6-15.

4. See the paper "Configuring and Operating Streamed Processing in PeopleSoft Global Payroll" on www.go-faster.co.uk.

Listing 6-15. *Case-insensitive search dialog query*

```
SELECT DISTINCT EMPLID, EMPL_RCD, NAME, LAST_NAME_SRCH, NAME_AC, PER_STATUS
FROM   PS_EMPLMT_SRCH_COR
WHERE  ROWSECCLASS=:1
AND    UPPER(NAME) LIKE UPPER('Smith') || '%' ESCAPE '\'
ORDER BY NAME, EMPLID, EMPL_RCD
```

However, placing a function on a column prevents Oracle from scanning an index by that column. If you don't want to suppress the functionality, an alternative is to build a function-based index on the columns that are frequently the subject of case-insensitive searches. If the function in the index matches the function on the column, then the index can be used. Experience has shown that this generally works well in PeopleSoft. There are only a few columns, usually name and address fields, where such an index is necessary.

The CREATE INDEX syntax is simple. You merely specify an expression instead of a column:

```
CREATE INDEX sysadm.PSZCUSTOMER ON sysadm.PS_CUSTOMER (UPPER(NAME1));
```

Function-based indexes require that three Oracle database initialization parameters are set.

- QUERY_REWRITE_INTEGRITY must be set to TRUSTED.
- QUERY_REWRITE_ENABLE must be set to TRUE.
- COMPATIBLE must be at least 8.1.0.0.0.

There is an administrative drawback to this approach. The PeopleSoft Application Designer cannot generate the DDL for function-based indexes. The list of indexed columns comes from an internal variable that cannot be altered other than by specifying columns in the index. Therefore, they must be maintained manually outside of PeopleSoft. They will also appear in the DDDAUDIT exception report because they are not defined in PeopleSoft.

■ **Tip** I have named the function-based index in line with a PeopleSoft user index with an ID of Z. It is very unlikely that a developer will try to add enough indexes to need Z, and if they do, they probably shouldn't. A PeopleSoft DDL script to rebuild the object or its indexes will try to drop any index not specified in PeopleSoft, so I sometimes suggest that the `CREATE INDEX` command be added to a regular database job, such as collecting optimizer statistics. If the index is missing, it gets put back.

Index Organized Tables

It is possible to make the Application Designer create an Index Organized Table. However, it is another complicated solution, with great potential for error. For example, if PSXLATITEM were to be created as an Index Organized Table, the syntax in Listing 6-16 would be required.

Listing 6-16. *Commands to create an Index Organized Table*

```
CREATE TABLE PSXLATITEM (FIELDNAME VARCHAR2(18) NOT NULL,
   FIELDVALUE VARCHAR2(4) NOT NULL,
   EFFDT DATE,
   EFF_STATUS VARCHAR2(1) NOT NULL,
```

```
   XLATLONGNAME VARCHAR2(30) NOT NULL,
   XLATSHORTNAME VARCHAR2(10) NOT NULL,
   LASTUPDDTTM DATE,
   LASTUPDOPRID VARCHAR2(30) NOT NULL,
   SYNCID INTEGER NOT NULL,
   CONSTRAINT PS_PSXLATITEM PRIMARY KEY (FIELDNAME,
   FIELDVALUE,
   EFFDT)
)
ORGANIZATION INDEX
TABLESPACE PTTLRG STORAGE (INITIAL 40000
 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
```

The following changes would be needed to the DDL model to get the Application Designer to generate this:

- The keywords ORGANIZATION INDEX must be added.
- PCTUSED is not a valid attribute of an Index Organized Table and must be hidden inside comments.
- A named PRIMARY KEY constraint must be added to the end of the column list. The primary key is defined in PeopleTools, but the IDXNAME and IDXCOLLIST internal parameters are not available to the Create Table DDL model, so the whole of the constraint, including the leading comma, would have to be manually coded as an override. If the primary key of the table changed, the constraint would have to be changed manually.
- Building the primary key index should be manually suppressed.

The alter table script would also always fail because the constraint name in the copy table would be the same as the original.

Other DDL

Other DDL statements are generated by the Application Designer that do not, either wholly or partly, use the DDL models. The following sections cover these statements.

Alter Table in Place

The Application Designer can generate the ALTER statements to add columns to a table without re-creating it, although it drops and re-creates the index. Only the `CREATE INDEX` command is generated from a DDL model; the rest is hard-coded in the Application Designer.

A table cannot always be altered in place. In such a case, it will be altered by re-creation, as shown in Listing 6-17.

Listing 6-17. *Script to alter a table by re-creation*

```
-- Alters for record PS_GFC_DATA_KEYS
--           DATA_KEY5 - add
-- Add Columns
```

```
ALTER TABLE PS_GFC_DATA_KEYS ADD DATA_KEY5 VARCHAR2(18)
/
-- Set Default Values
UPDATE PS_GFC_DATA_KEYS SET DATA_KEY5 = ' '
/
-- Modify NULLability

ALTER TABLE PS_GFC_DATA_KEYS MODIFY DATA_KEY5 NOT NULL
/
-- Done
DROP INDEX PS_GFC_DATA_KEYS
/
CREATE UNIQUE  INDEX PS_GFC_DATA_KEYS ON PS_GFC_DATA_KEYS (DATA_KEY1,
   DATA_KEY2) TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
```

■ **Caution** If columns are added in a production-like environment to a table and subsequently populated, the rows can consume all the free space in the block, causing the rows to migrate to a second block. This can cause performance problems.

Alter Table by Re-creation

Most table alterations are done by building, populating, and renaming a copy of the original table. The DDL models are used to create the temporary table and the indexes at the end of the script.

Alter table scripts cannot be run directly by the Application Designer; rather, they must be run manually in SQL*Plus. Listing 6-18 shows another script to alter a table by re-creation.

Listing 6-18. *Script to alter a table by re-creation*

```
-- Create temporary table
CREATE TABLE PSYGFC_DATA_KEYS (DATA_KEY1 VARCHAR2(18) NOT NULL,
   DATA_KEY2 VARCHAR2(18) NOT NULL,
   DATA_KEY3 VARCHAR2(18) NOT NULL,
   DATA_KEY4 VARCHAR2(18) NOT NULL,
   DATA_KEY5 VARCHAR2(18) NOT NULL) TABLESPACE USERS STORAGE (INITIAL
 40000 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
 PCTUSED 80
/
-- Copy from source to temp table
INSERT INTO PSYGFC_DATA_KEYS (
        DATA_KEY1,
    DATA_KEY2,
    DATA_KEY3,
    DATA_KEY4,
    DATA_KEY5)
```

```
  SELECT
        DATA_KEY1,
    DATA_KEY2,
    DATA_KEY3,
    DATA_KEY4,
    ' '
  FROM PS_GFC_DATA_KEYS
/
-- CAUTION: Drop Original Table
DROP TABLE PS_GFC_DATA_KEYS
/
-- Rename Table
RENAME PSYGFC_DATA_KEYS TO PS_GFC_DATA_KEYS
/
-- Done
CREATE UNIQUE  INDEX PS_GFC_DATA_KEYS ON PS_GFC_DATA_KEYS (DATA_KEY1,
   DATA_KEY2) TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
```

■ **Caution** This script is not without risks. If the creation or population of the temporary table fails for any reason, the script will continue and drop the original table, and the data will be lost. To stop the SQL*Plus session if an error occurs, add this command:

```
WHENEVER SQLERROR EXIT FAILURE
```

It is not until Oracle 10 that you can "undrop" a table using the flashback facility.

Views

The `CREATE VIEW` commands are entirely hard-coded. It is not possible, for instance, to change the command to `CREATE OR REPLACE`. Listing 6-19 shows how to drop and re-create a view.

Listing 6-19. *Dropping and re-creating a view*

```
DROP VIEW PS_NAME_TYPE_VW
/
CREATE VIEW PS_NAME_TYPE_VW (EMPLID, ORDER_BY_SEQ, NAME_TYPE) AS
 SELECT DISTINCT A.EMPLID ,B.ORDER_BY_SEQ ,A.NAME_TYPE FROM PS_NAMES A
 , PS_NAME_TYPE_TBL B WHERE A.NAME_TYPE = B.NAME_TYPE
/
COMMIT
/
```

PeopleSoft Temporary Tables

These are regular permanent tables that are used for temporary working storage by Application Engine processes. Do not confuse them with Oracle Global Temporary Tables!

In order to permit multiple instances of the same Application Engine process to run concurrently without contention on working storage tables, PeopleSoft creates temporary tables. Additional copies of each table are created in the database, and each instance of the Application Engine process is allocated to a different copy of the table.

A table named according to the usual conventions is always built along with a number of copies of that table, with the instance number of the table added as a suffix. The number of additional copies of the table is determined by the sum of the following:

- A global number of temporary table instances (PSOPTIONS.TEMTBLINSTANCES). This can be set via the PIA on PeopleTools Options.
- The number of instances for each Application Engine process that references a particular table (PSTEMPTBLCNTVW.TEMPTBLINSTANCES).

If the table is not referenced by any Application Engine program, no additional copies will be built. Each copy of the table and its indexes are built with the DDL model in the usual way (as shown in Listing 6-20).

Listing 6-20. *DDL script to create all copies of a PeopleSoft temporary table*

```
DROP TABLE PS_GPJP_YSS4_TAO
/
CREATE TABLE PS_GPJP_YSS4_TAO (PROCESS_INSTANCE DECIMAL(10) NOT NULL,
   EMPLID VARCHAR2(11) NOT NULL,
   GPJP_YEA_NONLIF DECIMAL(18, 6) NOT NULL,
   GPJP_YEA_LTNONL DECIMAL(18, 6) NOT NULL) TABLESPACE GPAPP STORAGE
 (INITIAL 40000 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0)
 PCTFREE 10 PCTUSED 80
/
CREATE UNIQUE  INDEX PS_GPJP_YSS4_TAO ON PS_GPJP_YSS4_TAO
 (PROCESS_INSTANCE,
   EMPLID) TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
DROP TABLE PS_GPJP_YSS4_TAO1
/
CREATE TABLE PS_GPJP_YSS4_TAO1 (PROCESS_INSTANCE DECIMAL(10) NOT NULL,
   EMPLID VARCHAR2(11) NOT NULL,
   GPJP_YEA_NONLIF DECIMAL(18, 6) NOT NULL,
   GPJP_YEA_LTNONL DECIMAL(18, 6) NOT NULL) TABLESPACE GPAPP STORAGE
 (INITIAL 40000 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0)
 PCTFREE 10 PCTUSED 80
/
CREATE UNIQUE  INDEX PS_GPJP_YSS4_TAO1 ON PS_GPJP_YSS4_TAO1
 (PROCESS_INSTANCE,
   EMPLID) TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
```

```
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
...
DROP TABLE PS_GPJP_YSS4_TA04
/
CREATE TABLE PS_GPJP_YSS4_TA04 (PROCESS_INSTANCE DECIMAL(10) NOT NULL,
   EMPLID VARCHAR2(11) NOT NULL,
   GPJP_YEA_NONLIF DECIMAL(18, 6) NOT NULL,
   GPJP_YEA_LTNONL DECIMAL(18, 6) NOT NULL) TABLESPACE GPAPP STORAGE
 (INITIAL 40000 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0)
 PCTFREE 10 PCTUSED 80
/
CREATE UNIQUE  INDEX PS_GPJP_YSS4_TA04 ON PS_GPJP_YSS4_TA04
 (PROCESS_INSTANCE,
   EMPLID) TABLESPACE PSINDEX STORAGE (INITIAL 40000 NEXT 100000
 MAXEXTENTS UNLIMITED PCTINCREASE 0) PCTFREE 10
/
```

The temporary tables can also be built by Data Mover using the `CREATE_TEMP_TABLE` command.

Triggers

From PeopleTools 8.4, the Application Designer will build a trigger (like the one in Listing 6-21) in either one or both of two situations. There is no DDL model for this trigger; it is generated directly by the Application Designer.

- **If a system ID field is defined on a record definition**: When a row is inserted, it will be set to the maximum value that exists on the table. Alternatively, if the key does not already exist, it will take a new value from the PSSYSID table.
- **If a timestamp field is specified on a record definition**: The trigger will set the value of that column to the current system date.

Listing 6-21. *PeopleTools trigger for mobile agent synchronization*

```
CREATE OR REPLACE TRIGGER PSUCURRENCY_CD_TBL BEFORE INSERT OR UPDATE
 OF  CURRENCY_CD,EFFDT,EFF_STATUS,DESCR,DESCRSHORT,COUNTRY,CUR_SYMBOL
,DECIMAL_POSITIONS,SCALE_POSITIONS ON PS_CURRENCY_CD_TBL FOR EACH ROW
 DECLARE PSSYSID  NUMBER (31,0);
BEGIN
IF INSERTING THEN
SELECT MAX(SYNCID) INTO PSSYSID FROM PS_CURRENCY_CD_TBL WHERE
 :NEW.CURRENCY_CD = PS_CURRENCY_CD_TBL.CURRENCY_CD;
IF PSSYSID <= 0 OR PSSYSID IS NULL THEN
UPDATE PSSYSTEMID SET PTNEXTSYSTEMID = PTNEXTSYSTEMID + 1 WHERE
 RECNAME = 'CURRENCY_CD_TBL';
SELECT PTNEXTSYSTEMID INTO PSSYSID FROM PSSYSTEMID WHERE
 RECNAME='CURRENCY_CD_TBL';
:NEW.SYNCID:= PSSYSID;
```

```
ELSE
:NEW.SYNCID:= PSSYSID;
END IF;
END IF;
:NEW.LASTUPDDTTM := SYSDATE;
END;
/
COMMIT
/
```

Synchronizing PeopleSoft with the Oracle Catalogue

Urban legends are created and grow because they seem plausible, but they lack proof. With Oracle, they start because they may have been true in the past, are written in books, are perpetuated by senior DBAs, and most of all because they seem plausible.

Without doubt, the greatest of the Oracle urban legends is that if tables are kept in single extents, you get better performance from DML operations (see Figure 6-11). It's plausible because if the tables are kept together on one extent, then they are kept together on the disk and the disk head doesn't spend time moving around. When you export data (Oracle export compresses extents by default) and then import it again, it produces a performance improvement.

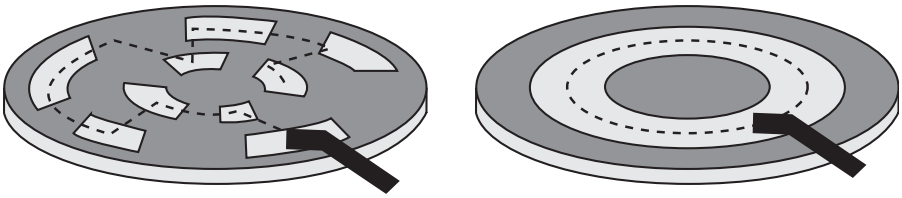


Figure 6-11. *Oracle Urban Legend #1: Fragmentation degrades DDL performance*[5]

However, reality is a little different. Oracle read operations are not infinitely large—they are limited in size by the size of an Oracle block and the maximum number of blocks that are read in a single operation (`db_block_size * db_file_multiblock_read_count`). Operating systems and volume management software also impose limits on the maximum size of a physical read operation. Export and import improves performance because the data is packed into the blocks, space in the index is reclaimed, migrated rows are eliminated, and high-water marks are reset. If the export had been done without compressing extents, it would have had the same effect. And finally, remember that most databases are multiuser systems, and the disk will have to serve requests from different users.

5. Connor McDonald, "Urban Legends" presentation, UKOUG Conference 2002 (www.oracledba.co.uk).

The legend is false. There is no provable link between DML performance and the number of extents that exist in a table.

What is true is that the number of extents can affect DDL operations when the table is in a dictionary managed tablespace. If you drop a table with 1,000 extents, then there are 1,000 entries to remove from the used extent map (uet$) and add to the free extent map (fet$), and you take out an exclusive lock on the single Space Transaction enqueue. It is the management of the dictionary that takes time. One of the reasons that Oracle introduced locally managed tablespaces in 8*i* was to address this problem.

This misunderstanding has caused DBAs (myself included) to waste many happy hours endlessly rebuilding objects with different sizing options in the STORAGE clause. In a world of 24/7 systems, it has become more difficult to schedule downtime to export and reimport an entire database in order to eliminate fragmentation that isn't really a problem in the first place.

Since at least 1990, when Oracle version 6.0 was current and long before locally managed tablespaces were invented, Oracle has recommended using uniform extent sizes in tablespaces to eliminate fragmentation,[6] and having objects of similar size and similar I/O profiles in the same tablespace. This means that the free spaces in the tablespace are reused because the holes will be the same size as the extents.

This does not mean you should have unlimited numbers of extents. Extent sizes should be chosen to limit the number of extents to a reasonable level (there is some consensus for a maximum of approximately 500). If a table has this many extents, then it should be in a different tablespace with a larger extent size.

In PeopleSoft systems, this led to the desire to resize objects and then feed the new extent sizes back into the PeopleTools tables so that PeopleSoft would rebuild the objects with the same sizes. I wrote and published a set of scripts to do exactly this in PeopleTools 7.*x*. In PeopleTools 8, PeopleSoft released three SQR batch programs (`settable.sqr`, `setindex.sqr`, and `setspace.sqr`) to feed all the storage parameters values specified in the default DDL model from USER_TABLES, USER_INDEXES, and DBA_TABLESPACES back into the PeopleSoft data dictionary.

However, feeding back the parameters is not a completely pointless exercise. Row migration occurs when you insert a row into the database and then update it so that it extends beyond the free space in the data block. The response is to adjust the PCT_FREE and PCT_USED parameters to reserve more free space in the blocks for update. It is appropriate to feed these parameters back into the PeopleTools tables.

If you have introduced either Partitioned or Global Temporary Tables, the storage options on USER_TABLES and USER_INDEXES will be NULL. `settable.sqr` and `setindex.sqr` will set the overrides in PeopleSoft to 0. If you forget that you should not use the Application Designer to build these objects, PeopleSoft will try to build them as regular objects, and the CREATE statement will fail because 0 is an invalid value for the INITIAL and NEXT storage parameters. However, if this happens in a PeopleSoft alter table script, and if SQL*Plus continues to execute the script after the failure, you may lose the data in a partitioned table.

If you don't need to defragment tables and indexes, and you have arranged for a uniform extent size in a tablespace, then you don't need to feed storage options back into PeopleTools.

---

6. Bhaskar Himatsingka and Juan Loaiza, Oracle Paper #711: "How to Stop Defragmenting and Start Living: The Definitive Word on Fragmentation," www.oracle.com/technology/deploy/availability/pdf/defrag.pdf, 1997.

At the beginning of this chapter, I suggested that if you are using locally managed tablespaces you might want to comment out the STORAGE clauses from the Create Table and Create Index DDL models. It is an equally valid idea if you are using dictionary managed tablespaces, and you let the tablespace default storage options determine the sizing for objects. I discuss the PeopleSoft tablespace model in the next chapter.

Feeding Back Tablespaces into PeopleTools

The DBA might choose to move tables and indexes to a different tablespace. The official PeopleSoft method is to make the changes and build the DDL script in the Application Designer. However, many DBAs are still going to prefer to move these objects manually. PeopleSoft needs to catch up with reality. `setindex.sqr` will feed the index's tablespace back as a DDL override, however, the table's tablespace is held in PSRECTBLSPC.DDLSPACENAME and is not updated by `settable.sqr`. The PL/SQL in Listing 6-22 updates the tablespace name in PSRECTBLSPC with the one correct tablespace name in USER_TABLES if they are different.

Listing 6-22. `recspcdiff.sql`

```
REM recspcdiff.sql
REM (c) Go-Faster Consultancy Ltd. 2004
set echo on feedback on head on trimspool on lines 80
set serveroutput ON buffer 1000000000
spool recspcdiff

DECLARE
   CURSOR c_spcdiff IS
   SELECT r.recname, s.ddlspacename
   ,      t.table_name, t.tablespace_name
   FROM   psrecdefn r
   ,      psrectblspc s
   ,      user_tables t
   WHERE  t.table_name = DECODE(r.sqltablename,' ','PS_'||r.recname,r.sqltablename)
   AND    r.rectype IN(0,7)
   AND    s.recname = r.recname
   AND    t.tablespace_name != s.ddlspacename
   ;
   p_spcdiff c_spcdiff%ROWTYPE;
   l_updcount INTEGER := 0;
BEGIN
   OPEN c_spcdiff;
   LOOP
      FETCH c_spcdiff INTO p_spcdiff;
      EXIT WHEN c_spcdiff%NOTFOUND;

      UPDATE psrectblspc
      SET    ddlspacename = p_spcdiff.ddlspacename
      WHERE  recname = p_spcdiff.recname
      ;
```

```
      l_updcount := l_updcount + 1;
      sys.dbms_output.put_line(
--       ''||l_updcount||':'||
         p_spcdiff.recname||':'||
         p_spcdiff.ddlspacename||'->'||
         p_spcdiff.tablespace_name);
   END LOOP;
   CLOSE c_spcdiff;
   sys.dbms_output.put_line(''||l_updcount||' records updated.');
END;
/
spool off
```

Summary

This chapter revealed the mechanism that PeopleSoft uses to generate the DDL to create tables and indexes based upon information stored in its own data dictionary. In addition, this chapter covered how to gather statistics for the cost-based optimizer.

The objective should always be to work with the PeopleSoft tools and, where possible, make them work for you, rather than continuously fighting against them. If you know how this mechanism works, then you can intercept it in order to enhance the DDL to include additional database features, such as physical storage options and optimizer statistics.

However, the DDL models have their limitations, and while it is always preferable to manage object parameters from within PeopleSoft, there are times when there really is no alternative but to manage them manually outside. In these cases, such as partitioning in Global Payroll, the benefit of using a feature must outweigh the overhead of managing it manually.

CHAPTER 7

Tablespaces

This chapter looks at the various tablespaces that PeopleSoft creates in an Oracle database and how they are used. Some of the choices that PeopleSoft has made are a little surprising, such as the decision to have many tablespaces for tables but only one for indexes. So, I will also discuss some of the options you have to make life a little easier when managing tablespaces.

In the PeopleSoft installation procedure, all of the default tablespaces in the database are created immediately after the database is created. Therefore, this is also a suitable place to consider how to create the database in the first place.

Database Creation

The procedure for creating a PeopleSoft database is described in the PeopleSoft installation guides. There are separate versions for each database platform, which are reissued for each PeopleTools release, and they can be downloaded from PeopleSoft's Customer Connection website. In this section, I focus on how, during the PeopleSoft installation procedure, the Oracle database itself is created.

Supplied Database Creation Scripts

PeopleSoft has always provided, and continues to provide, SQL scripts with which to create an Oracle database. From PeopleTools 8.4, the description of the process for manual creation of a PeopleSoft database using scripts has been relegated to an appendix of the installation guide by the introduction of PeopleSoft's Database Configuration Wizard, which is discussed later in this chapter. The scripts that PeopleSoft supplies are also clearly intended for use with Oracle8*i*. Most of the tablespaces, including the system and rollback tablespaces, are dictionary managed.

The script `utlspace.sql` runs the Oracle catalogue scripts, creates a system rollback segment, and creates a number of "utility tablespaces."

Temporary Tablespace

The temporary tablespace for the users on a PeopleSoft database has always been PSTEMP. The tablespace can be created manually using the `utlspace.sql` script (see Listing 7-1) supplied by PeopleSoft. Only from PeopleTools 8.4 does it use a TEMPFILE for the temporary tablespace.

Listing 7-1. `utlspace.sql`*: Creating the PSTEMP tablespace*

```
CREATE TEMPORARY TABLESPACE PSTEMP
TEMPFILE                '<drive>:\oradata\<SID>\pstemp01.dbf'     SIZE 300M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K
;
```

Until approximately PeopleTools 8.15, PSTEMP was created as a permanent tablespace. Thereafter, it was created as a tablespace of type temporary, but it still had a DATAFILE. However, only in one release of PeopleTools 7.5*x* did a patch script temporarily create permanent working storage tables in PSTEMP. Therefore, I have no hesitation in recommending that, even in previous versions of PeopleSoft, the PSTEMP tablespace should be created as shown in Listing 7-1, as a temporary tablespace with a TEMPFILE.

The Connect ID, the user whose schema contains the PeopleSoft database, is created by the script `psadmin.sql` (see Listing 7-2). It assigns the PSTEMP tablespace as the temporary tablespace for that user.

Listing 7-2. *An extract from* `psadmin.sql`

```
ACCEPT ADMIN CHAR PROMPT 'Enter name of PeopleSoft Owner ID: '
ACCEPT PASSWORD CHAR PROMPT 'Enter PeopleSoft Owner ID password:'
PROMPT
PROMPT Enter a desired default tablespace for this user.
PROMPT
PROMPT Please Note:  The tablespace must already exist
PROMPT               If you are unsure, enter PSDEFAULT or SYSTEM
PROMPT
ACCEPT TSPACE CHAR PROMPT 'Enter desired default tablespace:'


REMARK -- Create the PeopleSoft Administrator schema.

create user &ADMIN identified by &PASSWORD default tablespace &TSPACE
temporary tablespace pstemp;
grant PSADMIN TO &ADMIN;

REMARK -- PeopleSoft Administrator needs unlimited tablespace in order to
REMARK -- create the PeopleSoft application tablespaces and tables in Data
REMARK -- Mover.  This system privilege can only be granted to schemas, not
REMARK -- Oracle roles.

grant unlimited tablespace to &ADMIN;

REMARK -- Run the commands below to create database synonyms.
REMARK -- Modify the connect string appropriately for your organization.
```

Default Tablespace

The tablespace PSDEFAULT is created for use as the default tablespace, as shown in Listing 7-3. From PeopleTools 8.4 it is also locally managed.

Listing 7-3. `utlspace.sql`*: Creating the PSDEFAULT tablespace*

```
CREATE TABLESPACE       PSDEFAULT
DATAFILE                '<drive>:\oradata\<SID>\psdefault.dbf'    SIZE 100M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
;
```

The comment at the top of the `psadmin.sql` script (in Listing 7-2) supplied by PeopleSoft suggests making SYSTEM the temporary tablespace for the Connect ID. **This is not a good idea**. In any database, the SYSTEM tablespace should be reserved exclusively for use by Oracle-supplied administrative scripts. If you are not sure, do not proceed further.

In earlier versions of this script, the alternative suggestion was to make PSTEMP the default tablespace. This was not a good idea either, because PSTEMP should be made into a tablespace of type temporary, and no permanent objects will be able to be created without explicitly specifying a tablespace.[1]

Tablespace PSDEFAULT was introduced at the same time as PSTEMP was created as type temporary. The comment in `psadmin.sql` changed at the same time to recommend PSDEFAULT as a possible default tablespace. Finally, this is a sensible suggestion[2] and is equally appropriate for earlier versions of PeopleTools.

Rollback Tablespace

Oracle9*i* introduces the concept of Automatic Undo, also referred to as System Managed Undo. At the time of this writing, I am wary of using this feature. It is new in Oracle9*i*, and there are still some reported problems with relinquishing space.

The PeopleSoft template for the Oracle Database Configuration Assistant, which is discussed later in this chapter, creates a database that uses Automatic Undo. From PeopleTools 8.4, the installation guides recommend that this should not be used any longer.

PeopleSoft provides the script shown in Listing 7-4 to create a rollback tablespace, PSRBS, manually. On Oracle8*i*, rollback segments can be created only in a dictionary managed tablespace. It also creates a system rollback segment in the system tablespace.

From Oracle 9, I recommend creating the rollback tablespace as a locally managed tablespace with a reasonably large uniform extent size.

Listing 7-4. `utlspace.sql`*: Creating the rollback tablespace*

```
create rollback segment r00 tablespace system
storage (initial 16k next 16k minextents 2 maxextents 20);

REM * Use ALTER ROLLBACK SEGMENT ONLINE to put r00 online without shutting
REM * down and restarting the database.
REM *
alter rollback segment r00 online;
```

1. This would not be a problem, though, because Application Designer DDL scripts always specify the tablespace, and it is a good practice to enforce.
2. In fact, I think that the script should not ask for a default tablespace. The default tablespace should be specified as PSDEFAULT just as the temporary tablespace is specified as PSTEMP!

```
CREATE TABLESPACE       PSRBS
DATAFILE                '<drive>:\oradata\<SID>\psrbs01.dbf'      SIZE 300M
DEFAULT
STORAGE (               INITIAL     4M
                        NEXT        4M
                        PCTINCREASE   0  )
;
```

PeopleSoft provides a script, `rollback.sql`, to create rollback segments in this tablespace, as shown in Listing 7-5.

Listing 7-5. *An extract from* `rollback.sql`

```
create rollback segment r01 tablespace PSRBS
  storage (minextents 4 maxextents 20 optimal 16M);
create rollback segment r02 tablespace PSRBS
  storage (minextents 4 maxextents 20 optimal 16M);
create rollback segment r03 tablespace PSRBS
  storage (minextents 4 maxextents 20 optimal 16M);
create rollback segment r04 tablespace PSRBS
  storage (minextents 4 maxextents 20 optimal 16M);
create rollback segment rbsbig tablespace PSRBS
  storage (initial 8M next 8M minextents 4 maxextents 32 optimal 32M);

alter rollback segment rbsbig online;
```

I believe that the large rollback segment, RBSBIG, is no longer useful in a modern PeopleSoft system. At no point in the delivered applications are there any `SET TRANSACTION` commands to force particular processing to use a particular rollback segment.

PeopleSoft systems are multilingual; pages and messages can be presented in the chosen language of an operator. The systems are delivered with a base language of English. There is a small performance penalty for operators whose chosen language is not the base language, therefore the base language of a system should be swapped to the language of the majority of the users. Up to PeopleTools 7.*x*, the process of swapping the base language of a PeopleSoft system did require a large rollback segment. In version 8, the process is performed by a Data Mover command and does not have a large rollback overhead. You might prefer not to create RBSBIG in the first place.

Note that the delivered script, `rollback.sql`, sets only RBSBIG online. Remember to add the other rollback segments to the `rollback_segments` initialization parameter.

Oracle9i Database Configuration Assistant

From Oracle9*i*, there is a new Database Configuration Assistant (DBCA) that will generate the database automatically, but it can also generate a set of scripts from which you can create the database in the usual manual manner. The PeopleTools installation guides for later versions of PeopleTools 8.1 that are certified with Oracle9*i* describe creating an Oracle9*i* database with the Oracle DBCA as the preferred method, but they also explain how to use scripts to create an Oracle8*i* database.

Tip PeopleSoft provides a template for use with the Oracle DBCA. It is not supplied with the PeopleSoft distribution CDs and must be downloaded from the Customer Connection website.

I always use scripts to create any Oracle database. I may appear to be a bit of a Luddite[3] when it comes to using these GUI tools. I am not philosophically opposed to simplifying and automating the process. However, unless you can review the scripts before they are executed, you cannot be certain of exactly what will be done. I adjust as much as possible of the database configuration within the DBCA and then have it generate the scripts.

You don't tend to create a brand-new database from scratch all that often in a PeopleSoft environment. When you do, it is likely to be a new enterprise release, and the procedure will have been changed. You are much more likely to clone an existing database, not least because the process of importing all the database objects via Data Mover and applying all the extra scripts specified in the PeopleSoft installation process is long and complex. All the more reason to make sure you know how the database is built, and that it is built correctly in the first place.

Default Oracle 9i DB Template for NTW2K and Unix Installations

Figure 7-1 shows the Oracle DBCA, with the PeopleSoft DBCA template loaded and highlighted.

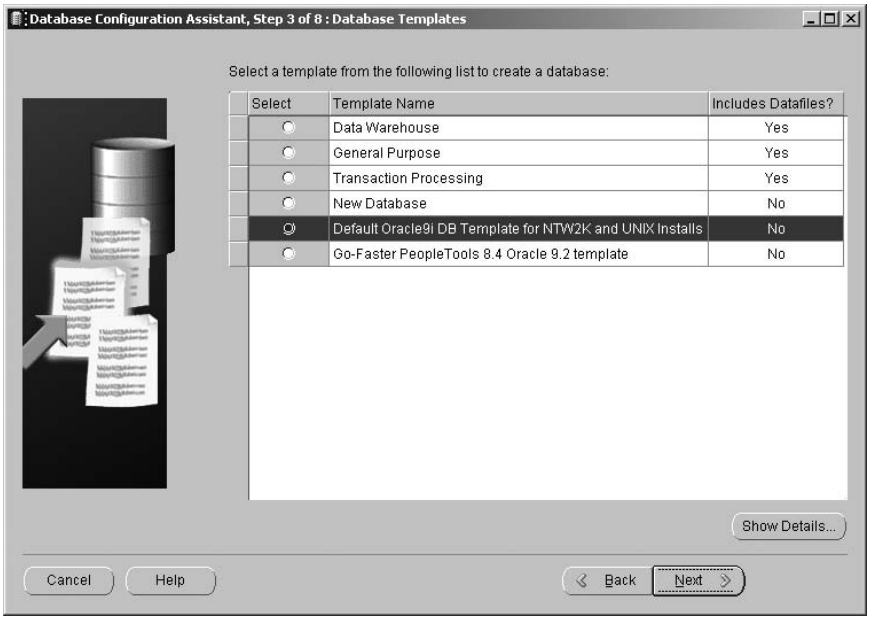


Figure 7-1. *Oracle9i Database Configuration Assistant*

3. **Lŭ'ddīte n. & adj.** (member) of bands of English artisans (1811–16) who raised riots for destruction of machinery; (person) similarly engaged in seeking to obstruct progress. [Perhaps from Ned Lud, insane person who destroyed two stocking frames about 1779]. (Source: *Concise Oxford Dictionary of Current English, 7th Edition*, 1985.)

The PeopleSoft template for the Oracle DBCA creates

- A Unicode database with the character set UTF-8
- A dictionary managed system tablespace, so that other tablespaces can still be dictionary managed
- The Automatic Undo tablespace UNDOTBS
- The temporary tablespace (with a TEMPFILE) TEMP
- A number of locally managed utility tablespaces (USERS, INDX, TOOLS, and CWMLITE) that will not be used by the PeopleSoft application

The PeopleSoft documented installation process continues to create the other application tablespaces with supplied scripts. The result is that you will have two temporary tablespaces, PSTEMP and TEMP.

You can also see in Figure 7-1 that I have created my own version of the template, Go-Faster PeopleTools 8.4, which includes my own setup preferences:

- The entire database, including the SYSTEM and ROLLBACK tablespaces, are locally managed, with automatic extent allocation.
- The control files are placed in different locations.
- A single-byte character set is used instead of Unicode.
- Automatic Undo is disabled, and tablespace UNDOTBS is not created.
- The unused utility tablespaces are not created.
- A number of other database initialization parameters are set.

I can understand creating the product-specific tablespaces by script, but the tablespaces for PeopleTools tables are common to all products and could have been created by the DBCA.

PeopleSoft Database Configuration Wizard

In PeopleTools 8.4, PeopleSoft has introduced its own Database Configuration Wizard, shown in Figure 7-2, to provide a consistent method of creating a PeopleSoft database on all database platforms. The documented process no longer suggests using the Oracle DBCA.

The PeopleSoft wizard is a Java process, and it can be run on any of the supported operating systems. The graphical interface shown in Figure 7-2 only runs on Windows. The wizard provides a text-only version on Unix systems.

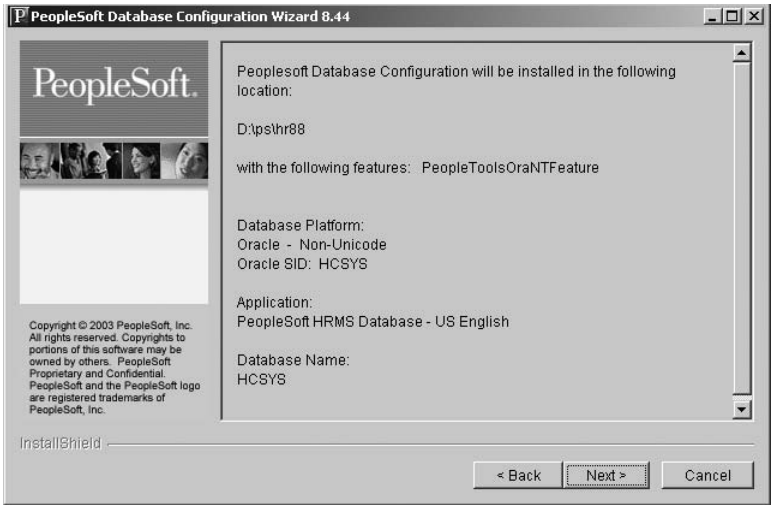


Figure 7-2. *PeopleSoft Database Configuration Wizard on Windows*

This contrasts with the Oracle installation and configuration tools, which are also written in Java but, as opposed to the PeopleSoft version, they require an X terminal when run on Unix platforms, and so present a consistent interface irrespective of the platform upon which they run.

The PeopleSoft wizard appears to be a wrapper for the same actions described in the manual database creation process. It requires to be told where the Oracle database software, and specifically where the SQL*Plus executable, is located. It will create a new Oracle instance and database (on Windows this includes the Oracle services), and populate it with the PeopleSoft database objects. The Data Mover import facility appears to have been incorporated for this purpose.

My chief objection to using this wizard is that there is no way of reviewing or controlling what it is going to do, before it does it. I, for one, will be sticking to the manual script-based database creation process described in the next section. The wizard is probably better suited to other database platforms, such as Microsoft's SQL Server, where the database server is established and then a database is created within that server.

Managing Tablespaces for PeopleSoft

PeopleSoft's approach to tablespaces has not significantly changed since at least PeopleTools 5.0, which was released in 1996, although each successive release has added to it.

The PeopleSoft application product lines or modules are given two character mnemonics, which are used as the first two characters for most of the tablespaces that contain tables. Table 7-1 does not present an exhaustive list—just a few examples.

Table 7-1. *Product Line Mnemonics*

Mnemonic	Product Line
BN	Base Benefits
EO	Enterprise Components
EP	Enterprise Performance Mgmt
FS	Financials
GP	Global Payroll
HR	Human Resources Management
IN	Inventory
PA	Pension Administration
PC	Projects
PI	Payroll Interface
PO	Purchasing
PS	Portal Solutions
PT	People Tools
PY	Payroll for North America
ST	Stock Administration
TL	Time and Labor

PeopleSoft usually creates a number of tablespaces for each product line. The rest of each tablespace name is made up by appending APP, LARGE, or WORK (or something similar; PeopleSoft is not always consistent) to the product line mnemonic. By default, all tablespaces will be dictionary managed.

- Tablespaces whose names end with APP or APPn (where *n* is a single digit) are where most of the application data tables are held. Nearly all product modules have at least one APP tablespace. Some have more (e.g., HRAPP, HRAPP1 through HRAPP6).
- Tablespaces whose names end with LARGE, LRG, or LARGn (where *n* is a single digit) generally hold the larger tables for a project module.
- Tablespaces whose names end in WORK, WRK, or WK are generally used to store tables used for working storage.
- Other tables whose names begin with PT or PS hold PeopleTools tables.
- The tablespaces SYSTEM, TEMP, TOOLS, USERS, INDX, CWMLITE, PSTEMP, and PSRBS should not be considered as PeopleSoft tablespaces. The DBA can safely adjust these as necessary.
- PSINDEX is the tablespace where, by default, ALL indexes on all the PeopleTools and application data tables are created.

PeopleSoft seems to have two completely different attitudes toward the use of tablespaces. On the one hand, all the tables are split into different tablespaces. Nominally, they are grouped by functional area and by size. In a vanilla HCM database, there are 92 such tablespaces. It is

almost at the point at which there may be so many tablespaces that the administration is becoming complicated. However, if the need arises, it is an easy matter to place tablespaces for different modules on different disk subsystems.

On the other hand, all the indexes for the tables in those 92 tablespaces are thrown together in a single tablespace, namely PSINDEX, as described next.

A Single Index Tablespace: PSINDEX

I can think of absolutely no reason why PeopleSoft should have chosen to place all indexes in a single tablespace. It is as if two distinct and different groups of developers had responsibility for tables and indexes.

You can typically expect the total volume of index segments to be around 105% of the total volume of table segments in a PeopleSoft database. So, the index tablespace can be larger than all the table tablespaces put together. As such, a monolithic index tablespace can produce administrative problems and risks. For example, if you have a file corruption to a database in this tablespace, the scope of the recovery operation will be much wider and more complex.

Also, if you are still backing up the database by putting each tablespace into backup mode, the PSINDEX tablespace will be in backup mode for a long time. The additional redo created during this period could be a problem, although this should not be an issue if you are backing up the database with Oracle's RMAN utility.

Creating Multiple Index Tablespaces

There are a number of options for breaking up the monolithic PSINDEX tablespace, which I describe in this section. Breaking up this tablespace will not, in itself, improve performance, but it may help relieve some of the administrative difficulties and risk associated with backup, recovery, and so on, as described in the previous section.

Manual Adjustment

The DBA could simply move indexes to new tablespaces by rebuilding them manually, and rely upon `setindex.sqr` (discussed in Chapter 6) to update the new tablespace name in the PeopleTools tables. The downside of this approach is that every time a new index is created, or an existing index is updated by a PeopleSoft patch, the index is likely to be recreated in PSINDEX again. It will result in a continual process of making changes to PeopleSoft objects.

AN ALTERNATIVE TABLESPACE MODEL

If you are willing to take on the additional work, then rather than separate tables by module and size, you could just organize them by size into locally managed tablespaces for tiny/empty, small, medium, large, huge, and work/temporary tables. You could organize a second set of tablespaces similarly for indexes. Reasonable uniform extent sizes can be chosen for each tablespace. The result is only 12 tablespaces, and not many more DATAFILEs, instead of around 100. This also helps during checkpointing.

One Index Tablespace per Table Tablespace

An easier strategy to break up the index tablespaces is to have one tablespace for indexes per table tablespace, and to place the index accordingly. This can be easily enforced by changing the DDL model for index creation (see Chapter 6).

Listing 7-6 shows the Create Index DDL model. The addition variable, **INDEXSPC**, which defaults to PSINDEX from the DDL model defaults, has been commented out and replaced with the internal variable [TBSPCNAME],[4] which is the tablespace name to which the table is allocated.

■ **Note** The original index tablespace variable **INDEXSPC** cannot be removed because it will be populated with the `setindex.sqr` process. If the variable is defined, by specifying a DDL model default, and does not appear in the DDL model, then the Application Designer will produce errors when the DDL is either viewed or generated.

Listing 7-6. *Alternative Index DDL model*

```
CREATE [UNIQUE] **BITMAP** INDEX [IDXNAME]
ON [TBNAME] ([IDXCOLLIST])
TABLESPACE [TBSPCNAME]_IDX /* **INDEXSPC** */
STORAGE (
    INITIAL **INIT** NEXT **NEXT**
    MAXEXTENTS **MAXEXT** PCTINCREASE **PCT**)
PCTFREE **PCTFREE**;
```

You will see that I have added a fixed suffix of _IDX, so the tablespaces I create for indexes must be named accordingly. For example, the table PS_JOB is stored in the tablespace HRLARGE. The index is now created in the tablespace HRLARGE_IDX, as shown in Listing 7-7.

Listing 7-7. *Creating the PS1JOB index*

```
CREATE   INDEX PS1JOB ON PS_JOB (JOBCODE, EMPLID, EMPL_RCD, EFFDT, EFFSEQ)
TABLESPACE HRLARGE_IDX /* PSINDEX */
STORAGE (INITIAL 123904 NEXT 100000 MAXEXTENTS UNLIMITED PCTINCREASE 0)
PCTFREE 10;
```

Each index will have to be rebuilt to move it into its new tablespace. The build script can be generated by the Application Designer by building the Create Index DDL for an Application Designer project containing every record in the database.

This has the limitation of permitting no flexibility. The location of the index will always follow the table. If the DBA attempts to move an index manually and runs `setindex.sqr`, the new tablespace value will go into **PSINDEX**, which is commented out in the DDL model. The

---

4. Internal variables from the Create Table DDL model appear to be available to the Create Index DDL model, but not vice versa.

PeopleSoft script will build the index only in the tablespace that corresponds to the tablespace for the table. However, this is also a benefit. When a new index is created, it will not, by default, go into PSINDEX, but into the index tablespace that corresponds to the table's tablespace.

This arrangement would also permit tablespaces to be moved or copied to other databases using the transportable tablespaces mechanism. The monolithic index tablespace would prevent transport of individual table tablespaces. This could be effective during testing or when moving data to a reporting database.

Tablespace Creation and Default Storage Options

In Chapter 6, I suggested that using a uniform extent size, even with dictionary managed tablespaces, was a widely recommended strategy even before the introduction of locally managed tablespaces. If a `CREATE TABLE` command does not specify any storage options, then the tablespace default storage options for the tablespace in which it is created will be used. However, this is difficult to implement in PeopleSoft, as DDL commands are always generated with explicit STORAGE parameters.

It is possible to prevent the Application Designer from generating DDL scripts with storage options by commenting them out in the DDL models, but when you first create a PeopleSoft database and import the objects with the PeopleSoft-supplied Data Mover export file, storage options will be specified for all tables and indexes by the DDL model because they are embedded in the export file. Although you can override storage options during the import, you cannot override the DDL model.

From this, we can come to two conclusions:

- Tablespace default storage options are always ignored unless you customize the DDL models.
- Uniform extent sizing is difficult to achieve in PeopleSoft with dictionary managed tables.

Implementing Local Tablespace Management

Toward the end of the last chapter, I talked about the myth of "one segment in one extent." Oracle has recommend using a uniform extent size since Oracle 6, and in 8*i* Oracle provided a method of enforcing it.

When a locally managed tablespace is created, a uniform extent size can be specified. Space management is handled with a bitmap in the file header, with each bit referring to one of the extents. If you specify a STORAGE clause when you create an object in a locally managed tablespace, the extents will have the uniform extent size of the tablespace, but the number of extents allocated will have the same capacity as if the table had been created in a dictionary managed tablespace.

EFFECT OF STORAGE OPTIONS ON OBJECTS IN LOCALLY MANAGED TABLESPACES

In my test HRAPP is a locally managed tablespace with automatic extent sizing. The block size is 8KB.

```
SELECT tablespace_name, block_size, initial_extent, extent_management
FROM dba_tablespaces WHERE tablespace_name = 'HRAPP';

TABLESPACE_NAME                BLOCK_SIZE INITIAL_EXTENT EXTENT_MAN
------------------------------ ---------- -------------- ----------
HRAPP                                8192          65536 LOCAL
```

Now I try to create a table with an initial and next extent size, and two extents:

```
CREATE TABLE dmk (a NUMBER) TABLESPACE hrapp
STORAGE (INITIAL 100K NEXT 50K MINEXTENTS 2);
```

On a dictionary managed tablespace, the extents would total 150KB, so in a locally managed tablespace I get three extents of 64KB, totaling 192KB:

```
SELECT  extent_id, bytes, blocks, tablespace_name
FROM    user_extents
WHERE   segment_name = 'DMK' AND segment_type = 'TABLE';

 EXTENT_ID      BYTES     BLOCKS TABLESPACE_NAME
---------- ---------- ---------- ------------------------------
         0      65536          8 HRAPP
         1      65536          8 HRAPP
         2      65536          8 HRAPP
           ---------- ----------
sum            196608         24
```

In the locally managed tablespace, Oracle allocates extents such that there is at least 150KB allocated in the table. By default, PeopleSoft doesn't specify MINEXTENT.

When a free extent is required, Oracle will take the first free bit in the bitmap, and this will be the free extent nearest the beginning of the file. All the objects in a locally managed tablespace will tend to be at the start of the file. In a dictionary managed tablespace, there are likely to be fragments of free space that may be too small for the extent size specified, and the extents can be spread throughout.

I cannot think of a scenario in which there would be any advantage to using dictionary instead of locally managed tablespaces, but the question is how to implement them in PeopleSoft.

The first challenge is to pick a reasonable uniform extent size. The tables are at least broken into different tablespaces, and within the product line, they are separated into small and large. However, there are lots of tables in all tablespaces that remain empty (often because county-specific functionality isn't used). In the demo HCM database, there are 12,000 empty tables out of 14,700, while only 600 empties are in the tablespaces for larger objects. If I make the

uniform extent size large, I could waste a lot of space (although in these days of absurdly large disks, that may have some benefits). You could perhaps consider using uniform extent sizes of 256KB to 1MB on the %LARGE tablespaces and the level of space wastage would probably be acceptable.

The next problem is what to do with the indexes. There are nearly 12,800 indexes on tables with no rows, all in the same tablespace. If I drop them, then I have to suppress them in the Application Designer; otherwise, they will appear in DDDAUDIT and could be rebuilt when a table is altered. Alternatively, I could move them to a different tablespace (as discussed previously). All options involve a level of customization.

I've begun to think that a reasonable compromise that does not involve *any* PeopleSoft customization is to use the default PeopleSoft tablespace structure, with the single index tablespace, and simply create all the tablespaces as locally managed with automatically allocated extent sizes. If you decide to go down this route, bear in mind the following:

- Tables that do contain data and their indexes will be likely to have multiple extents, but this is neither a performance nor an administrative problem.
- Empty tables will consume at least 64KB (the smallest extent size) instead of 40KB in dictionary managed tablespace (as specified in the DDL model defaults). The additional overhead will usually be acceptable.
- This does not resolve the issue of the monolithic tablespace.

In an AUTOALLOCATE tablespace, each bit in the file header will refer to a 64KB chunk of tablespace. Tables will usually have extents of 64KB, 1MB, 8MB, and even 64MB. The larger extents will be represented by several bits in the map. Although the extent sizes are no longer uniform, each size is a multiple of the smaller ones. This may cause some free spaces in the tablespace to open up, but the spaces will be more likely to be reused. Tables that become huge—perhaps larger than 1GB—could be moved to separate new tablespaces with a large uniform extent size.

DDL Overhead

The grain of truth in the "one extent per segment" myth is that there is a DDL overhead to operations that require additional extents to be added to an object.

In many of the Application Engine batch processes, particularly in the PeopleSoft Financials product, it is common to write data to a working storage table. By version 8, PeopleSoft has developed the technique of having multiple copies of each temporary table and allocating different tables to different processes. The tables are usually truncated by the process, populated again, and then optimizer statistics are estimated. Truncation will deallocate all extents beyond the minimum number of extents. When the table is repopulated, those extents may be reallocated.

Default PeopleSoft DLL does not specify MINEXTENTS, and so it defaults to 1. If the temporary table exists in a dictionary managed tablespace, then the repeated allocation and deallocation of all the extents will impact performance.

This effect is still present in locally managed tablespaces, but it is so greatly reduced that it is almost negligible. It is still appropriate to set MINEXTENTS to a suitable value and use locally managed tablespaces with uniform extents of an adequate size.

Summary

This chapter was devoted to tablespaces in PeopleSoft. Because tablespaces are created when the database is created, we first examined database creation procedures. PeopleSoft started to use the Oracle9*i* Database Configuration Assistant (DBCA) in PeopleTools 8.1, but you may want to review what it generates. PeopleSoft then abandoned the DBCA in favor of its own wizard, but this does not allow the DBA to review the process. I recommend creating the database by script.

Having uniform extent sizes in each tablespace is a good idea. Using locally managed tablespaces will enforce this. Automatic extent sizing will allow you to introduce this feature without any customization and without wasting large amounts of space. This will not improve DML performance, but it has administrative advantages.

From Oracle9*i*, the system tablespace can be made locally managed, and then all other tablespaces must be locally managed too. I can't think of a good reason not to do this. You may also want to break the indexes up into different tablespaces, but this will require customization.

The Application Designer is a developer's tool. As a DBA, use it to the extent that it helps you do your job, but do not be afraid to bypass or ignore it if it gets in the way of you doing your job.

CHAPTER 8

Locking, Transactions, and Concurrency

This chapter discusses some of the methods that PeopleSoft uses to maintain data integrity in a multiuser environment while at the same time avoiding techniques that are specific to a particular database platform.

This chapter begins with a discussion of locking in PeopleSoft. It is useful to understand how and where PeopleSoft locks tables, because this can have implications for the performance and scalability of your applications. I then examine how PeopleSoft uses tables to generate sequence numbers instead of Oracle sequences, in order to maintain platform independence. However, this platform independence has implications for concurrency and scalability. Finally, I cover the row-level locking that occurs during PIA transactions, and how by holding locks for as short a time as possible, PeopleSoft minimizes the impact of the table-based sequences.

Locking

PeopleSoft, like most database applications, relies mostly upon implicit row-level locking to protect data being updated by one transaction from being corrupted by another. However, there are times when it is necessary to deliberately serialize some processing so that only one thing can happen at a time. In such cases, it would be normal to exclusively lock a table, and if necessary wait to acquire the lock.

The Oracle database itself also has mechanisms that must be serialized. For example, whenever space management is performed in a dictionary managed tablespace, the space transaction (ST) enqueue latch must be obtained to protect the updates to the free (SYS.FET$) and used (SYS.UET$) space maps. Only one process can acquire the latch at a time, and this can be the cause of contention.

However, it is unusual to see a PeopleSoft application lock a table with an explicit `LOCK TABLE` command, because that is Oracle-specific code, and corresponding platform-specific code would be required for all the other databases. Instead, PeopleSoft will update a row on a table, possibly without changing the data value.

For example, an Application Engine process may use a PeopleSoft temporary table, in which a single record in PeopleSoft corresponds to a number of copies of the table (see Chapter 4, Table 4-3, where PSRECDEFN, Record Type 7 is described). Each copy is referred to in PeopleTools as an *instance*. When the Application Engine process starts, it must determine which instance of the temporary table is available, and then it allocates that instance to itself by inserting a row into the table PS_AETEMPTBLMGR.

It must make sure that no other Application Engine process is trying to allocate an instance of that record at the same time, so it locks the table PS_AELOCKMGR by updating the only row in it, thus deliberately serializing this processing, as shown in Listing 8-1. A row-level lock on a table with only a single row has the same functional behavior as a table-level lock.

Listing 8-1. *Processing serialized by a lock on table PS_AELOCKMGR*

```
UPDATE PS_AELOCKMGR SET AE_LOCK = :1
;
SELECT MIN(B.CURTEMPINSTANCE)
FROM   PS_AEINSTANCENBR B
,      PSTEMPTBLCNTVW C
WHERE  C.RECNAME = :1
AND    B.CURTEMPINSTANCE > :2
AND    (B.CURTEMPINSTANCE - C.TEMPTBLINSTANCES) <= :3
AND    B.CURTEMPINSTANCE NOT IN (
         SELECT D.CURTEMPINSTANCE
         FROM   PS_AETEMPTBLMGR D
         WHERE  D.RECNAME = C.RECNAME)
;
INSERT INTO PS_AETEMPTBLMGR (PROCESS_INSTANCE, RECNAME, CURTEMPINSTANCE,
  OPRID, RUN_CNTL_ID, AE_APPLID, RUN_DTTM, AE_DISABLE_RESTART, AE_DEDICATED)
VALUES(:1, :2, :3, :4, :5, :6, SYSDATE, :7, :8)
;
COMMIT
;
```

PIA Transactions

This section examines what happens when you change an existing value in a PIA component and save the new value to the database. I want to draw attention to how PeopleSoft manages consistency of data without holding database locks for a long time while the operator enters data.

Figure 8-1 shows the Update Personal Data component from the HCM product line (accessed by selecting Personal Information ➤ Biographical ➤ Update Personal Information). The name is an effective-dated attribute—for example, a name may change on marriage, effective from the date of marriage. In this case, PeopleSoft, when in normal add/update mode, adds a new row to a table with that effective date. However, in the following example, I am using "correction" mode to change an existing name record, not adding a name change.

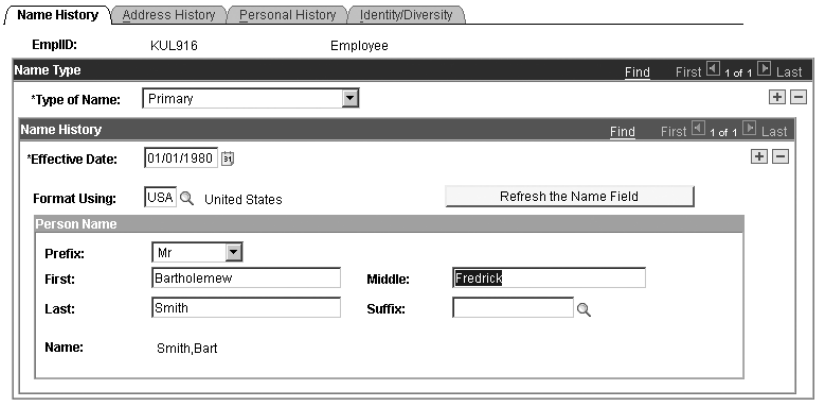


Figure 8-1. *Update Personal Information component*

The SQL generated by the PIA has been captured by the PeopleTools trace. I have only shown the SQL that references the PS_NAMES table where employee names are stored.

When the Update Personal Information component is opened, the PS_NAMES table is queried from the database, as shown in Listing 8-2, into the component buffer for the requested employee ID (EMPLID). Note that all of the name history for the employee is loaded. There may be only a few rows per employee on PS_NAMES, but this is generic PIA behavior, and other tables such as the job and absence histories will grow more quickly.

Listing 8-2. *PS_NAMES is loaded into the component buffer.*

```
PSAPPSRV 1-1988 15.30.00 Cur#1 RC=0 COM
Stmt=SELECT EMPLID, NAME_TYPE, EFFDT, TO_CHAR(EFFDT,'YYYY-MM-DD'),
COUNTRY_NM_FORMAT, NAME, NAME_INITIALS, NAME_PREFIX, NAME_SUFFIX,
NAME_ROYAL_PREFIX, NAME_ROYAL_SUFFIX, NAME_TITLE, LAST_NAME_SRCH,
FIRST_NAME_SRCH, LAST_NAME, FIRST_NAME, MIDDLE_NAME, SECOND_LAST_NAME,
SECOND_LAST_SRCH, NAME_AC, PREF_FIRST_NAME, PARTNER_LAST_NAME,
PARTNER_ROY_PREFIX, LAST_NAME_PREF_NLD
FROM PS_NAMES WHERE EMPLID=:1
ORDER BY EMPLID, NAME_TYPE, EFFDT DESC
PSAPPSRV 1-1989 15.30.00 Cur#1 RC=0 Bind-1 type=2 length=6 value=KUL916
PSAPPSRV 1-1990 15.30.00 Cur#1 RC=0 Fetch
PSAPPSRV 1-1991 15.30.00 Cur#1 RC=1 Fetch
```

The query in Listing 8-2 could have returned more than one row into the Name History scroll,[1] although in this case we know it did not because only the first fetch has a zero return code (also, we can see in Figure 8-1 from the Name History scroll that only one row was returned this time).

1. The page in Figure 8-1 shows two *scrolls*, Name Type and Name History. They are used to control parent-to-child relationship between sets of data. A scroll approximates to a PeopleSoft record that is loaded by the component. That record may be a table, a view, or working storage. Scrolls usually have the record control in the right-hand side of the title bar. Personal Name is not a scroll.

   In this case, Name Type corresponds to the view PS_NAME_TYPE_VIEW, which is keyed on EMPLID and NAME_TYPE, and Name History, which is keyed on EMPLID, NAME_TYPE, and EFFDT.

Fifteen seconds later, after I updated the name fields in the component, I clicked the Save button. Listing 8-3 shows that the PIA queries the data again to make sure that it hasn't been changed by another process in the intervening period. This time, the PS_NAMES table is queried by all three primary key fields, so only a single row is returned. The key values come from the row that I updated.

Listing 8-3. *PS_NAMES is requeried, locked, and updated.*

```
PSAPPSRV 1-2230 15.30.15 Cur#1 RC=0 COM Stmt=SELECT EMPLID, NAME_TYPE, EFFDT,
TO_CHAR(EFFDT,'YYYY-MM-DD'), COUNTRY_NM_FORMAT, NAME, NAME_INITIALS,
NAME_PREFIX, NAME_SUFFIX, NAME_ROYAL_PREFIX, NAME_ROYAL_SUFFIX, NAME_TITLE,
LAST_NAME_SRCH, FIRST_NAME_SRCH, LAST_NAME, FIRST_NAME, MIDDLE_NAME,
SECOND_LAST_NAME, SECOND_LAST_SRCH, NAME_AC, PREF_FIRST_NAME, PARTNER_LAST_NAME,
PARTNER_ROY_PREFIX, LAST_NAME_PREF_NLD
FROM PS_NAMES
WHERE EMPLID=:1 AND NAME_TYPE=:2 AND EFFDT=TO_DATE(:3,'YYYY-MM-DD')
FOR UPDATE OF NAME, LAST_NAME_SRCH, FIRST_NAME_SRCH, FIRST_NAME, MIDDLE_NAME
PSAPPSRV 1-2231 15.30.15 Cur#1 RC=0 Bind-1 type=2 length=6 value=KUL916
PSAPPSRV 1-2232 15.30.15 Cur#1 RC=0 Bind-2 type=2 length=3 value=PRI
PSAPPSRV 1-2233 15.30.15 Cur#1 RC=0 Bind-3 type=26 length=10 value=1980-01-01
PSAPPSRV 1-2234 15.30.15 Cur#1 RC=0 Fetch
PSAPPSRV 1-2235 15.30.15 Cur#2 RC=0 Connect=Primary/HR88/SYSADM/
PSAPPSRV 1-2236 15.30.15 Cur#2 RC=0 Dur=0.010 COM Stmt=UPDATE PS_NAMES SET NAME=:1,
LAST_NAME_SRCH=:2,FIRST_NAME_SRCH=:3,FIRST_NAME=:4,MIDDLE_NAME=:5
WHERE EMPLID=:6 AND NAME_TYPE=:7 AND EFFDT=TO_DATE(:8,'YYYY-MM-DD')
PSAPPSRV 1-2237 15.30.15 Cur#2 RC=0 Bind-1 type=2 length=26
value=Smith,Bartholemew Fredrick
PSAPPSRV 1-2238 15.30.15 Cur#2 RC=0 Bind-2 type=2 length=5 value=SMITH
PSAPPSRV 1-2239 15.30.15 Cur#2 RC=0 Bind-3 type=2 length=11 value=BARTHOLEMEW
PSAPPSRV 1-2240 15.30.15 Cur#2 RC=0 Bind-4 type=2 length=11 value=Bartholemew
PSAPPSRV 1-2241 15.30.15 Cur#2 RC=0 Bind-5 type=2 length=8 value=Fredrick
PSAPPSRV 1-2242 15.30.15 Cur#2 RC=0 Bind-6 type=2 length=6 value=KUL916
PSAPPSRV 1-2243 15.30.15 Cur#2 RC=0 Bind-7 type=2 length=3 value=PRI
PSAPPSRV 1-2244 15.30.15 Cur#2 RC=0 Bind-8 type=26 length=10 value=1980-01-01
PSAPPSRV 1-2245 15.30.15 Cur#2 RC=0 Disconnect
...
PSAPPSRV 1-2362 15.30.15 0.030 Cur#1 RC=0 Dur=0.030 Commit
```

Notice also that the row is selected FOR UPDATE OF the columns that have changed, thus taking out a database lock on this row. The same columns that are updated in the UPDATE statement appear in the FOR UPDATE OF clause.

This has certain implications:

- The row is not locked while the user is working in the component. The lock is only requested within the save-time processing.
- Once I have acquired the lock on this row, nobody else can update this row until my transaction has been committed.

- If somebody else has a lock on this row, my transaction will wait to be able to complete. This will appear to extend the time taken to save the component, during which the "processing" message will be displayed.
- The transaction is short-lived, because it does not wait for the user at any point. It starts when the row is requeried for update, thus locking the row, and it commits the update at the end. In this case, all the save-time PeopleCode and SQL processing have the same clock time in the trace in Listing 8-3, and so took less than a second to complete.

If the second query detects that the data has changed since it was first loaded into the component buffer, the PIA raises the error "(18,1) Page data is inconsistent with database". The help text for the message (see Listing 8-4) points out that the problem can also be generated by SQL updates issued from within the save-time PeopleCode.

Listing 8-4. *Additional explanation for the "Page data is inconsistent with database" error message*

```
When trying to save your page data, the system found that the information
currently in the database did not match what was expected.

This problem can happen if another user has changed the same information while
you were making your changes. Note the changes you have made, then cancel the
page. Reload the page and view any changes made by the other user. Ensure your
changes are compatible and retry, if appropriate.

If the problem persists, it may be a result of an application or other
programming error and should be reported to technical support staff.

Possible application errors that can cause this message include:
- changing page data from SavePostChange PeopleCode, without making a
corresponding change to the database.
- changing the database via SqlExec at various points, for data that is also in
the component buffers.
```

Different SQL statements are generated to save component data, depending upon which columns are updated by either the operator or any PeopleCode that executes. The code in Listing 8-5 was generated by the same component when I updated only the employee's middle name and not the first name.

Listing 8-5. *PS_NAMES is requeried, locked, and updated again.*

```
PSAPPSRV   1-7144   17.13.14    Cur#1 RC=0 COM Stmt=SELECT EMPLID, NAME_TYPE, EFFDT,
TO_CHAR(EFFDT,'YYYY-MM-DD'), COUNTRY_NM_FORMAT, NAME, NAME_INITIALS,
NAME_PREFIX, NAME_SUFFIX, NAME_ROYAL_PREFIX, NAME_ROYAL_SUFFIX, NAME_TITLE,
LAST_NAME_SRCH, FIRST_NAME_SRCH, LAST_NAME, FIRST_NAME, MIDDLE_NAME,
SECOND_LAST_NAME, SECOND_LAST_SRCH, NAME_AC, PREF_FIRST_NAME, PARTNER_LAST_NAME,
PARTNER_ROY_PREFIX, LAST_NAME_PREF_NLD
FROM PS_NAMES WHERE EMPLID=:1 AND NAME_TYPE=:2 AND EFFDT=TO_DATE(:3,'YYYY-MM-DD')
FOR UPDATE OF NAME, LAST_NAME_SRCH, FIRST_NAME_SRCH, MIDDLE_NAME
...
```

```
Dur=0.000 COM Stmt=UPDATE PS_NAMES
SET NAME=:1,LAST_NAME_SRCH=:2,FIRST_NAME_SRCH=:3,MIDDLE_NAME=:4
WHERE EMPLID=:5 AND NAME_TYPE=:6 AND EFFDT=TO_DATE(:7,'YYYY-MM-DD')
...
```

NAME, LAST_NAME_SRCH, and FIRST_NAME_SRCH are updated by save-time PeopleCode, and so also appear in the FOR UPDATE OF clause of the query and the SET clause of the UPDATE statement. FIRST_NAME was not updated and so does not appear in either statement.

This shows that PeopleSoft updates only the columns that have changed rather than the whole row. Although this does reduce redo logging, the main implication is that there are a greater number of different SQL statements, each of which has to be parsed. A larger library cache in the SGA will help, but there will always be a lot of SQL parsing in a PeopleSoft database (see the sidebar "Effects of Length-Checking Constraints on Parse Time" in Chapter 4 for more information).

Sequence Numbers and Concurrency

PeopleSoft does not use Oracle sequences. This is mainly because it would require widespread use of platform-specific code in the PIA. While this is not completely unprecedented, PeopleSoft generally avoids it.

The platform-agnostic alternative that PeopleSoft has chosen is to store a sequence number in a table and increment it as a part of the transaction. For example, in HCM when a new employee is hired, the employee ID is shown as NEW (see Figure 8-2) until the transaction is saved.

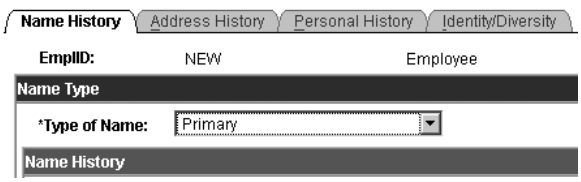


Figure 8-2. *Hiring a new employee*

During the save-time processing, the PeopleCode function `assign_employee_id()` (see Listing 8-6) is used to increment the value of EMPLID_LAST_EMPL on the table PS_INSTALLATION (which contains only a single row). The new value is then selected and used as the new employee's number.

Listing 8-6. *Incrementing the EMPLID sequence number*

```
[FUNCLIB_HR.EMPLID.FieldFormula]
...
SQLExec("Update PS_INSTALLATION Set EMPLID_LAST_EMPL = EMPLID_LAST_EMPL +  1");
SQLExec("Select EMPLID_LAST_EMPL From PS_INSTALLATION", &EMPLID);
```

From the point in the save-time processing at which the new employee ID was obtained up to the point at which the transaction is committed, the single row on the PS_INSTALLATION table is locked and nobody else can allocate a sequence number. So this method effectively serializes allocation of sequence numbers.

The amount of time for which this lock is held is a function of the amount of save-time processing in the application. In the case of the employee hire processing, it can be several seconds, so only one employee hire transaction can be saved at any one time.

PeopleSoft applications derive sequence numbers from various tables, but in HCM the situation is compounded because there are at least 17 sequence numbers derived from the single row on the table PS_INSTALLATION (see Table 8-1). Incrementing any one of these columns will lock the row and prevent another process from allocating a sequence number for any column, even a different one.

Table 8-1. *Sequence Numbers on PS_INSTALLATION*

Column Name	Description
ACCT_CD_LAST	Account Code
CAR_LAST	Car Number
CLAIM_NBR_LAST	H&S Claim Number
COBRA_EMPLID_LAST	COBRA Employee
DEMAND_ID_LAST	Demand ID
EMPLID_LAST_EMPL	Employee ID
FSA_CLAIM_NBR_LAST	FSA Claim Number
GRIEVANCE_NBR_LAST	Grievance Number
ILL_NBR_LAST_GER	Illness Number (Germany)
ILL_NBR_LAST_NLD	Illness Number (Netherlands)
INCIDENT_NBR_LAST	H&S Incident Number
LAST_TL_CONTRCTR	T&L Contractor ID
NON_EMPLOYEE_LAST	H&S Non-Employee Number
POSN_NBR_LAST	Position Number
RETRODED_SEQ_LAST	Payroll Retro Deduction Sequence
RETROPAY_SEQ_LAST	Payroll Retro Payment Sequence
TEMP_ASSGN_ID_LAST	Temporary Job Assignment ID

Although most systems are unlikely to use all 17 sequences, they will usually use several. It is easy to see how a moderately active system could start to experience contention on sequence number allocation.

If you must generate sequence numbers from a table, then you should at least have different sequences on different rows, so that they can be updated independently. For example, when PeopleSoft applications are developed, version numbers are allocated to objects to control the caching. There is a sequence of version numbers for each object type. Until PeopleTools 7, these numbers came from different columns in the single row on PSLOCK. In PeopleTools 8, PSLOCK has just two columns and one row for each of the 46 PeopleTools object types, as shown in Listing 8-7. Each sequence number can be allocated independently.

Listing 8-7. *PeopleTools version number sequences*

```
OBJECTTY    VERSION
-------- ----------
ACM              2
AEM              2
AES              2
BCM              2
BPM              2
...
```

Cached[2] Oracle sequences would resolve the concurrency issues of allocating sequence numbers, but would also permit gaps in the sequence of numbers. If, for some reason, a transaction that selects a number from a sequence rolls back, the sequence is not rolled back, and so the number is never used in the application.

However, that may not be acceptable in some applications. Unexplained gaps in a sequence of invoice numbers would give rise to concern among accountants and auditors.

Summary

This chapter covered locking, transactions, and concurrency in PeopleSoft. As discussed in the first part of this chapter, when a PeopleSoft application does require a database lock, it will usually do so by creating a row-level lock by updating a table that has only a single row.

The next part of the chapter delved into PIA transactions. Database transactions generated by PIA activity should be short-lived because they never wait for user response. In that respect, they can be thought of as very small batch processes. Thus, any locks that they hold are held only for a short time. It is unusual to find PIA save-time processing locked on other PIA save-time processing.

Last, I discussed sequence numbers and concurrency. Care must be taken when making customizations—especially in SaveEdit, FieldChange, FieldEdit, and SavePreChg PeopleCode, which fire at save time—that these changes do not introduce locking or significantly extend the length of the transaction because of poorly performing SQL. A single, isolated transaction may be affected by "only" a fraction of a second, which may escape unnoticed in unit testing. However, in a high concurrency situation, the extended holding of a row lock can easily translate into many seconds or even minutes of performance degradation to the business function (I've seen it), ultimately to the point at which the system becomes effectively unusable. PeopleSoft derives sequence numbers from tables. This serializes processing that allocates new sequence numbers and can lead to performance problems. In online processing, the effect is mitigated because the locks are held for only a short time. However, batch processes can hold locks for a longer time, and this can impact other batch processes and online activity.

There is not much DBAs can do about the issues discussed in the chapter—this is the way PeopleSoft applications work—but DBAs should at least be aware of them.

2. Oracle sequences are held on the table SYS.SEQ$. If the sequences were not cached, you would have the same concurrency issue as the table-based method.

CHAPTER 9

Performance Metrics

Performance tuning is a search for lost time.

Performance improvements are achieved by reducing response time.[1] Therefore, I am always looking for the processes or parts of processes that take the most time. These places are where it is most efficient to direct the tuning effort, and where the minimum amount of tuning effort can have the maximum amount of performance effect.

Oracle provides the SQL trace facility to allow you to determine how long SQL statements take to execute and tell you how they are executing. The trace can be enhanced to report what Oracle is waiting for when it isn't executing SQL (event 10046 level 8). However, what if the problem is not the database? If it isn't the database, then what is it? The SQL trace file will report so-called idle wait time. That could be client process busy time, but it could also be user coffee time. You need also to be able obtain measurements for other parts of the PeopleSoft technology.

The only place that it is reasonable to measure the performance of the PeopleSoft Internet Architecture (PIA) is standing behind the user with a stopwatch in hand. The response time that the user experiences is the sum of the response times for the database, application server, Java servlet, web server, network, and browser.

This chapter details the various sources of performance metrics and monitoring facilities within PeopleSoft, specifically online monitoring and metrics, batch metrics, and trace files. Some of these are physical log files, whereas others are stored within the database. While none of them exactly measures the user experience, they are certainly closer to the user than the database. From these, it is possible to obtain better information about how well a particular piece of the technology chain or a particular process is performing.

For data that isn't already in the database, I often load it with SQL*Loader. Then I can process it with SQL and relate different time-based sets of data. I can then query it from Microsoft Excel via an ODBC link and present the data graphically. The resulting graphs can be easily incorporated into documents or presentations.

The techniques described in this chapter require a certain amount of time and effort to implement and operate, and they cannot easily be completely automated. They are not an industrial-strength solution, and I do not suggest that they are a suitable long-term solution to performance monitoring of a PeopleSoft system. For that, you need to look at the commercial packages that are available, some of which have PeopleSoft-specific modules.

---

1. See Anjo Kolk, Shari Yamaguchi, and Jim Viscusi, "Yet Another Performance Profiling Method," www.oraperf.com/whitepapers.html, 1999.

However, the techniques I outline here do have the merit of not requiring any additional software. They can quickly be brought to bear upon a current performance problem, which is perhaps not the best time to procure, install, and "bed in" another sophisticated software package.

PeopleSoft has started to address the problem of performance measurement[2] with the introduction of PeopleSoft Ping in PeopleTools 8.4 and with the Performance Monitor utility in PeopleTools 8.44. These tools work by instrumenting PIA service requests from the browser to the database and back again. I'll look at them more closely in the next chapter.

Online Monitoring and Metrics

When an operator logs into PeopleSoft via the PIA, that operator is at one end of a chain of technology that stretches across the enterprise to the core of the IT infrastructure. By measuring the performance of the chain at various points along that chain, as shown in Figure 9-1, it is possible to isolate the performance of each section and, by extension, performance problems in each section.

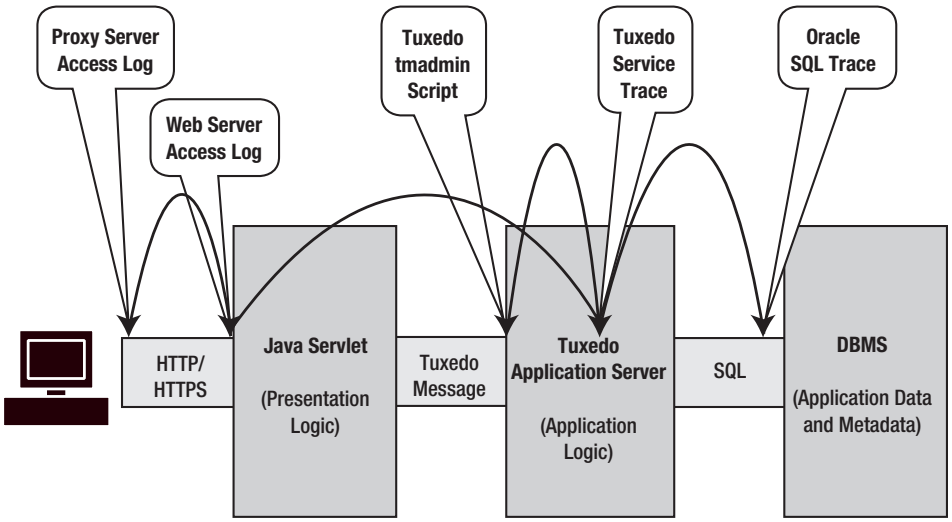


Figure 9-1. *PeopleSoft Internet Architecture with sources of metrics*

For instance, by deducting the sum of the Tuxedo service times from the sum of the service times for servlet requests for a given period, it is possible to calculate how much time is spent in the Java servlet and queuing.[3] A `tmadmin` script will determine how much queuing occurred in the application server.

2. Don't confuse the term *performance measurement* with the PeopleSoft Enterprise Performance Measurement (EPM) product. I am talking only about measuring the performance of the technology.
3. It is very difficult to do this for individual servlet requests because the requests can only be matched on the basis of time, and in some cases, one servlet request can correspond to more than one Tuxedo service request.

In this section, I discuss each of the sources of online performance metrics in the web and application server tiers, including how they can be configured and analyzed.

Application Server

Most of the performance metrics for online activity are generated from the Tuxedo Application Server, although later you will see that the web server access log is also useful. The following sections cover the various sources of performance information in the application server.

EnableDBMonitoring

It is sometimes useful to know which user is responsible for issuing a particular SQL statement to his database. Oracle provided the DBMS_APPLICATION_INFO PL/SQL package (see `$ORACLE_HOME/rdbms/dbmsapin.sql`) to provide a mechanism to register information about the client, module, and the current action with the database.

From PeopleTools 7.53, PeopleSoft introduced the parameter `EnableDBMonitoring` to both the application server configuration file, `psappsrv.cfg` (see Listing 9-1), and the Process Scheduler configuration file, `psprcs.cfg`. In PeopleTools 8.4, this parameter has been removed from `psprcs.cfg` because it is enabled by default.

Listing 9-1. *Extract from* `psappsrv.cfg`

```
[Database Options]
;============================================================================
; Database-specific configuration options
;============================================================================

SybasePacketSize=
; Please see Chapter "Tuning and Administration", in
; Oracle Installation and Administration Guide for details
UseLocalOracleDB=0
;ORACLE_SID=
; Dynamic change allowed for EnableDBMonitoring
EnableDBMonitoring=1
```

Enabling this parameter causes the application server to write a string to the session information using the Oracle-supplied PL/SQL package DBMS_APPLICATION_INFO.SET_CLIENT_INFO. The value can be read back with DBMS_APPLICATION_INFO.READ_CLIENT_INFO, or it can be queried from `v$session.client_info` (see Listing 9-2).

Listing 9-2. *Database sessions of PeopleSoft processes*

```
SELECT client_info, program
FROM   v$session
WHERE  client_info IS NOT NULL
```

```
CLIENT_INFO                                          PROGRAM
---------------------------------------------------- ---------------
PSAPPS,David,GO-FASTER-3,HR88,PSSAMSRV.exe,          PSSAMSRV.exe
PSAPPS,David,GO-FASTER-3,HR88,PSMONITORSRV.exe,      PSMONITORSRV.exe
PS,,go-faster-3,HR88,PSAPPSRV.exe,                   PSAPPSRV.exe
PSAPPS,David,GO-FASTER-3,APPSRV7,PSAESRV.exe,        PSAESRV.exe
PSAPPS,David,GO-FASTER-3,APPSRV7,PSPRCSRV.exe,       PSPRCSRV.exe
PSAPPS,David,GO-FASTER-3,APPSRV7,PSDSTSRV.exe,       PSDSTSRV.exe
```

The string reports the PeopleSoft Operator ID, the operating system user who started the process, the IP address of the browser that made the request (or the physical machine name if it can be resolved), the database name, and the program name. When the application server processes a service request, it first rewrites this string with the operator who made the current request. Thus the DBA can see the Operator ID who requested the currently active SQL. In Listing 9-2, all the application server and process scheduler processes[4] were started by PSAPPS, but PSAPPSRV is processing, or just processed, a request from the operator PS.

This is not really a performance metric, but it is useful to identify which PeopleSoft Operator ID is responsible for what SQL, so this can be incorporated into other scripts. Of course, if operators shared Operator IDs, this information loses some of its value. Other PeopleSoft processes populate this value too, but in the application server, it has to be explicitly enabled.

It would be useful to know which component or which PeopleCode module is being executed, but PeopleSoft does not use the SET_MODULE or SET_ACTION functions in DBMS_APPLICATION_INFO. So you only know which user submitted a request through which server process, but not what that user is doing.

tmadmin

This is BEA's administrative utility for Tuxedo. It is a command-line interface that can be used to administer or monitor a BEA Tuxedo system, and it can also be used within scripts.

Commands can be piped in as keyboard input, and the output captured as standard output, as shown in Listing 9-3.

Listing 9-3. `tmadmin.bat`

```
REM tmadmin.bat
REM (c) Go-Faster Consultancy Ltd. 2004
set TUXDIR=d:\ps\bea\tuxedo8.1
set TUXCONFIG=d:\ps\hr88\appserv\hr88\PSTUXCFG
%TUXDIR%\bin\tmadmin -r <tmadmin.in >tmadmin.out
```

■ **Tip** Monitoring scripts should always use the -r switch to run the `tmadmin` utility in read-only mode. It prevents them from issuing any administrative commands. Only one instance of `tmadmin` can connect to a Tuxedo domain in administrative mode at the same time.

4. The example is from PeopleTools 8.44.06. I have no idea why the Process Scheduler processes erroneously report the database name as APPSRV7. This value does not appear in any of the configuration files.

A full description of the `tmadmin` commands can be found on the BEA documentation CD or on the BEA website (http://e-docs.bea.com). The most useful commands for monitoring are presented in Listing 9-4.

In a Unix script, the commands can be piped directly into the `tmadmin` command, but on Windows you need to put them in a separate file (as shown in Listing 9-4) and pipe the file into `tmadmin` (as shown in Listing 9-3).

Listing 9-4. `tmadmin.in`

```
pclt
pq
psr
q
```

where:

- `pclt` or `printclient`: Lists clients of the bulletin board.
- `pq` or `printqueue`: Lists queues within the application server and the number of requests queued.
- `psr` or `printserver`: Lists application server processes and how many service requests they have processed.
- `q` or `quit`: Scripts that call `tmadmin` must manually quit from the utility or they hang indefinitely.

Listing 9-5 shows the output from the script in Listing 9-3.

Listing 9-5. `tmadmin.out`

```
     LMID         User Name       Client Name    Time    Status  Bgn/Cmmt/Abrt
--------------- --------------- --------------- -------- ------- -------------
GO-FASTER-3     NT              WSH             3:03:14 IDLE    0/0/0
GO-FASTER-3     NT              JSH             3:03:14 IDLE    0/0/0
GO-FASTER-3     NT              tmadmin         0:00:00 IDLE    0/0/0

Prog Name      Queue Name  # Serve Wk Queued  # Queued  Ave. Len    Machine
---------      ------------------ ---------  --------  --------    -------
PSAPPSRV.exe   APPQ              1         -         0         - GO-FASTER+
JREPSVR.exe    00094.00250       1         -         0         - GO-FASTER+
PSMONITORSRV.e MONITOR           1         -         0         - GO-FASTER+
BBL.exe        61778             1         -         0         - GO-FASTER+
WSL.exe        00001.00020       1         -         0         - GO-FASTER+
JSL.exe        00095.00200       1         -         0         - GO-FASTER+
PSWATCHSRV.exe WATCH             1         -         0         - GO-FASTER+
PSSAMSRV.exe   SAMQ              1         -         0         - GO-FASTER+
```

```
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
---------      ----------  --------      -- ------ --------- ---------------
BBL.exe        61778       GO-FAST+       0    187      9350 (  IDLE )
PSMONITORSRV.e MONITOR     MONITOR        1      0         0 (  IDLE )
PSAPPSRV.exe   APPQ        APPSRV         1    192      9600 (  IDLE )
PSWATCHSRV.exe WATCH       WATCH          1      0         0 (  IDLE )
JREPSVR.exe    00094.00250 JREPGRP      250    141      7050 (  IDLE )
JSL.exe        00095.00200 JSLGRP       200      0         0 (  IDLE )
WSL.exe        00001.00020 BASE          20      0         0 (  IDLE )
PSSAMSRV.exe   SAMQ        APPSRV       100      0         0 (  IDLE )
```

Tuxedo Service Trace (-r Option)

If the application server process is started with the `-r` option (see Listing 9-6), then Tuxedo will log every service request that passes through the server, noting the start and end times.

Listing 9-6. *Extract from* `psappsrv.ubb`*: Application server process command line*

```
CLOPT="-r -e D:\ps\hr88\appserv\HR88\LOGS\APPQ.stderr  -s@..\psappsrv.lst
-s@..\psqcksrv.lst -sICQuery -sSqlQuery:SqlRequest -- -C psappsrv.cfg -D HR88 -S PSAPPSRV"
```

The output is written to the standard error file, which by default is in the current working directory for the server process, `$PS_HOME/appserv/<domain name>`, and is called `stderr`. Thus all application servers will write to the same file. The output file name can be specified explicitly with the `-e` option, so servers on different queues can write to different files. In Listing 9-6, I have chosen to call the file `APPQ.stderr`, and I have also specified an output directory.

The overhead of the Tuxedo service trace is low. It is reasonable to enable it on production systems for long periods. Unfortunately, this trace has become less useful in PeopleTools 8, because nearly all the work of the PIA is spent executing `ICPanel` services. In contrast, PeopleTools 7 provided different services for different actions in the Windows client.

This trace cannot be enabled or disabled dynamically. The application server must be shut down, reconfigured, and restarted. If they are not managed, the log files will simply continue to grow. There is no log file rotation in Tuxedo; trace files must be administered manually.

For each service, there are two pairs of numbers (see Listing 9-7). The date and time at the start and end of the service is logged. The dates are in whole seconds, and the origin is 00:00hrs GMT on January 1, 1970. The units of the time columns are operating system dependent. On Windows they are milliseconds, whereas on most Unix platforms they are centiseconds.

Listing 9-7. *Extract from* `APPQ.stderr`

```
SERVICE         PID         SDATE         STIME         EDATE         ETIME
 -------     ---         -----         -----         -----         -----
@JavaMgrGetObj 1320     1082982338     6892861       1082982338    6892981
@ICPanel       1320     1082982346     6900772       1082982346    6900993
@JavaMgrGetObj 1320     1082982346     6901063       1082982346    6901073
@ICPanel       1320     1082982354     6909044       1082982354    6909295
@JavaMgrGetObj 1320     1082982354     6909305       1082982354    6909315
@ICPanel       1320     1082982370     6924937       1082982370    6925237
@ICPanel       1320     1082982373     6927811       1082982373    6927971
@ICPanel       1320     1082982375     6930215       1082982376    6930695
```

```
@JavaMgrGetObj 1320    1082982376    6930705      1082982376    6930725
@ICPanel       1320    1082982387    6941761      1082982387    6941962
@ICPanel       1320    1082982389    6943914      1082982389    6944125
@ICPanel       1320    1082982391    6946408      1082982392    6946578
@ICPanel       1320    1082982393    6947890      1082982393    6948080
```

The times are 32-bit integers that increment and recycle when they reach the maximum value (in approximately 50 days on Windows and 500 days on Unix). It is not possible to directly determine the time at which the service began from this value. The times can be used only to determine the duration of the service request. I have noticed on some platforms that the times will lose a few time units per day against the date column, although I have no explanation for this.

The log file is written by the Tuxedo libraries that are compiled into each application server program. It records the time at which the service message reaches the server and the time it left the server. Therefore, the difference is the time spent inside the server process, and it does not include any queuing time. The entry is written only when the service completes. If the service times out or the server crashes for some reason, the service will not be logged.

Tuxedo provides a simple command-line utility, `txrpt`, to process the `stderr` file, as shown in Listing 9-8. This utility is fully described in the Tuxedo documentation.

Listing 9-8. *Running* `txrpt`

```
txrpt <APPQ.stderr >txrpt.out
```

The `txrpt` utility produces an hour-by-hour-by-service summary of the performance of the application server, showing the number of services processed and the average service execution time, as shown in Listing 9-9.

Listing 9-9. `txrpt.out`

```
      SERVICE SUMMARY REPORT


SVCNAME         13p-14p     TOTALS
                Num/Avg     Num/Avg
--------------- --------    -------
ICPanel         18/7.16     18/7.16
PortalRegistry  13/0.28     13/0.28
ICScript        10/2.07     10/2.07
JavaMgrGetObj   4/0.04       4/0.04
GetCertificate  3/1.04       3/1.04
HomepageT       1/1.39       1/1.39
PpmMonSvc       1/0.10       1/0.10
GetWebProfile   1/6.03       1/6.03
--------------- -------     -------
TOTALS          51/3.22     51/3.22
```

Although `txrpt` is a good "quick and dirty" way to find out how the application server is performing, it reports only average execution times, which can hide a great deal of information. If the standard error file were to be loaded into a table (see Listing 9-10) in the database, it would be possible to ask more sophisticated questions and correlate them with other metrics to produce graphs.

Listing 9-10. *Script to create a table to hold Tuxedo service trace information*

```
CREATE TABLE txrpt
(service    VARCHAR2(20)                NOT NULL
,pid        NUMBER(6)                   NOT NULL
,stimestamp DATE                        NOT NULL
,stime       NUMBER(10,3)                NOT NULL
,etime       NUMBER(10,3)                NOT NULL
,queue       VARCHAR2(5)  DEFAULT 'XXXXX' NOT NULL
,concurrent NUMBER(2)    DEFAULT 0       NOT NULL
,scenario   NUMBER(2)    DEFAULT 0       NOT NULL
);

CREATE unique index txrpt
ON txrpt(service,pid,stimestamp,stime,etime);
```

Extra analysis columns, `concurrent` and `queue`, have been added to the table. They can be updated later.

It is then easy to load the `stderr` file into the table using Oracle's SQL*Loader utility, using the control file described in Listing 9-11. You could also use external tables to load this data.

Listing 9-11. *SQL*Loader control file*

```
LOAD DATA
INFILE 'APPQ.stderr'
REPLACE
INTO TABLE txrpt
WHEN (1) = '@'
FIELDS TERMINATED BY WHITESPACE
TRAILING NULLCOLS
(service    "SUBSTR(:service,2)"       -- remove leading @
,pid
,stimestamp ":stimestamp/86400+1/24+TO_DATE('01011970','DDMMYYYY')"
                                       -- convert to Oracle data (GMT+1)
,stime       ":stime/1000"              -- convert to seconds (NT)
,queue       "'APPQ'"                   -- do not remove this line
,etime       ":etime/1000"              -- convert to seconds (NT)
)
```

In this example control file:

- The `stimestamp` column in the table is converted to an Oracle date using a function string.
- The function includes +1/24 to convert the time from GMT to the local time zone. On Unix systems, the time zone variable can be set to GMT instead.

  ```
  set TZ=GMT0; export TZ
  ```

- The `stime` and `etime` columns are converted on load from operating system times to seconds. On Windows, the raw data is 1/1,000 of a second; on most Unix platforms, the raw data is 1/100 of a second.

- The queue name is defaulted to APPQ because `APPQ.stderr` is being loaded, and this was produced by server processes on the APPQ queue. Do not change the position of the `queue` directive in the loader file because it is used to skip the `etimestamp` column in the `stderr` file.
- The data cannot be loaded in direct path mode in conjunction with the SQL operator strings.

Application Server Log

The LogFence parameter in the application server configuration file, `psappsrv.cfg` (see Listing 9-12), controls the amount of information written to the application server log file, `APPSRV_mmdd.log` (where `mmdd` is the month and day). If the parameter is set to 4, the additional trace information includes some performance metrics.

Listing 9-12. *Extract from* `psappsrv.cfg`

```
[Domain Settings]
;------------------------------------------------------------------------
; Logging detail level
;
; Level     Type of information
; -----     -------------------
;  -100     - Suppress logging
;  -1       - Protocol, memory errors
;   0       - Status information
;   1       - General errors
;   2       - Warnings
;   3       - Tracing Level 1 (default)
;   4       - Tracing Level 2
;   5       - Tracing Level 3
; Dynamic change allowed for LogFence
LogFence=4
```

Listing 9-13 is a sample of the additional trace information generated with LogFence set to a value of 4. It shows that PeopleSoft operator PS logged in and opened component JOB_DATA in the Administer Workforce menu, and the `ICPanel` service took 0.421 seconds to execute. Note also that the LogFence value of the message appears in brackets on every line in the trace.

Listing 9-13. *Application server log* `APPSRV_0822.log`

```
PSAPPSRV.1980 [08/23/04 12:41:25 PS@GO-FASTER-3 ICPanel](4) Starting Service
PSAPPSRV.1980 [08/23/04 12:41:25 PS@GO-FASTER-3 (IE 6.0; WINNT) ICPanel](4)
Executing component JOB_DATA/GBL in menu ADMINISTER_WORKFORCE_(GBL)
PSAPPSRV.1980 [08/23/04 12:41:25 PS@GO-FASTER-3 (IE 6.0; WINNT) ICPanel](4) (NET.109):
Client RamList service request object created
PSAPPSRV.1980 [08/23/04 12:41:25 PS@GO-FASTER-3 (IE 6.0; WINNT) ICPanel](4)
(NET.111):
Client RamList service request object destroyed
PSAPPSRV.1980 [08/23/04 12:41:25 PS@GO-FASTER-3](4) Service ICPanel completed:
```

```
elapsed time=0.4210
PSAPPSRV.1980 [08/23/04 12:41:26 PS@GO-FASTER-3 MgrGetObject](4) Starting Service
PSAPPSRV.1980 [08/23/04 12:41:26 PS@GO-FASTER-3 MgrGetObject](4) MgrGetService :
13 objects CRM(CRM(PS_BEL_EX/1/_/ENG))
PSAPPSRV.1980 [08/23/04 12:41:26 PS@GO-FASTER-3](4) Service JavaMgrGetObj completed:
elapsed time=0.0300
```

Listing 9-14 is the Tuxedo service trace output that corresponds to the entries in Listing 9-13. This reports that the `ICPanel` service took 0.441 seconds.

Listing 9-14. *Tuxedo service trace* `APPQ.stderr`

```
@ICPanel       3976       1093261285   2059100      1093261285   2059541
@JavaMgrGetObj 3976       1093261286   2059591      1093261286   2059621
```

Note that there is a discrepancy between the two trace files. The Tuxedo server process was busy for 0.441 seconds, but the PeopleSoft code took 0.421 seconds to execute. This is not surprising, as these traces are generated by different instrumentation at different points in the technology chain, and so mean slightly different things. This also implies that the Tuxedo code took 0.02 seconds to handle the messages.

While the application server log tells you which operator accessed which part of the application, the Tuxedo service trace gives a more accurate timing for the duration of the service. Setting LogFence to 4 also generates a lot of additional information, but there is only a small overhead on service time for so doing.

BEA WebLogic Server Access Log

BEA's WebLogic server supports the extended log file format defined by the World Wide Web Consortium (W3C), which permits additional information to be recorded in the web server's access log file. The current working draft for this standard is at www.w3.org/TR/WD-logfile.html (this address is subject to change, so see also www.w3.org/pub/WWW/TR).

However, access logging must first be enabled and the extended format selected in the WebLogic configuration file. A custom enhanced access log format can be specified by directives in the access log itself, as I discuss in the sections that follow.

WebLogic 5.1 Configuration

PeopleTools 8.1*x* uses WebLogic 5.1. The configuration file, `weblogic.properties`, shown in Listing 9-15, is a plain file. The name of the access log is specified as `access.log`.

Listing 9-15. *Extract from* `weblogic.properties`

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# HTTPD ADMINISTRATIVE PROPERTIES
# ------------------------------------------------
# Enables logging of HTTPD info in common log format and
# sets the log file name (default is "access.log" in "myserver")
weblogic.httpd.enableLogFile=true
weblogic.httpd.logFileName=access.log
```

```
#dmk 19.2.2002 default values
#weblogic.httpd.logFileFlushSecs=60
#weblogic.httpd.logFileBufferKBytes=8
#weblogic.httpd.logFileFormat=common or extended

#dmk 19.2.2002 selected extended log format, force frequent flushing
weblogic.httpd.logFileFlushSecs=1
weblogic.httpd.logFileFormat=extended

#rotate log files
weblogic.httpd.logRotationType=date

#dmk 19.2.2002 default rotation time is 1 day
#weblogic.httpd.logRotationPeriodMins=1440

#dmk 19.2.2002 the files will rotate every midnight
weblogic.httpd.logRotationBeginTime=11-24-2000-00:00:00
```

In Listing 9-15, log rotation has been configured. WebLogic will rename the current `access.log` file and start writing a new file each day at midnight.

I have also forced WebLogic to flush the log buffer to disk every second. This can be useful if you want to watch the log file on a Unix telnet session with `tail -f`, but this may degrade performance. Further details are available in the BEA WebLogic online manuals.

WebLogic 6.1 and 8.1 Configuration

WebLogic 6.1 was introduced from PeopleTools 8.4, and WebLogic 8.1 from PeopleTools 8.44. The configuration is now formatted in XML in a file called `config.xml`, as shown in Listing 9-16.

Listing 9-16. *Extract from* `config.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Domain ConfigurationVersion="8.1.1.0" Name="peoplesoft">
...
        ListenAddress="" ListenPort="7201"
        MSIFileReplicationEnabled="true"
        ManagedServerIndependenceEnabled="true" Name="PIA"
        ServerVersion="8.1.1.0" StagingDirectoryName="./stage"
        StagingMode="nostage" UploadDirectoryName="./upload">
        <WebServer LogFileFlushSecs="1" LogFileFormat="extended"
            LogFileLimitEnabled="true"
            LogFileName="./logs/PIA_access.log"
            LogRotationTimeBegin="01-01-2004-0:00:00"
            LogRotationType="date" LoggingEnabled="true" Name="PIA"/>
```

You can edit this file, but it is safer and easier to configure the server with the WebLogic console, as shown in Figure 9-2. The access log file is now called `PIA_access.log` by default.

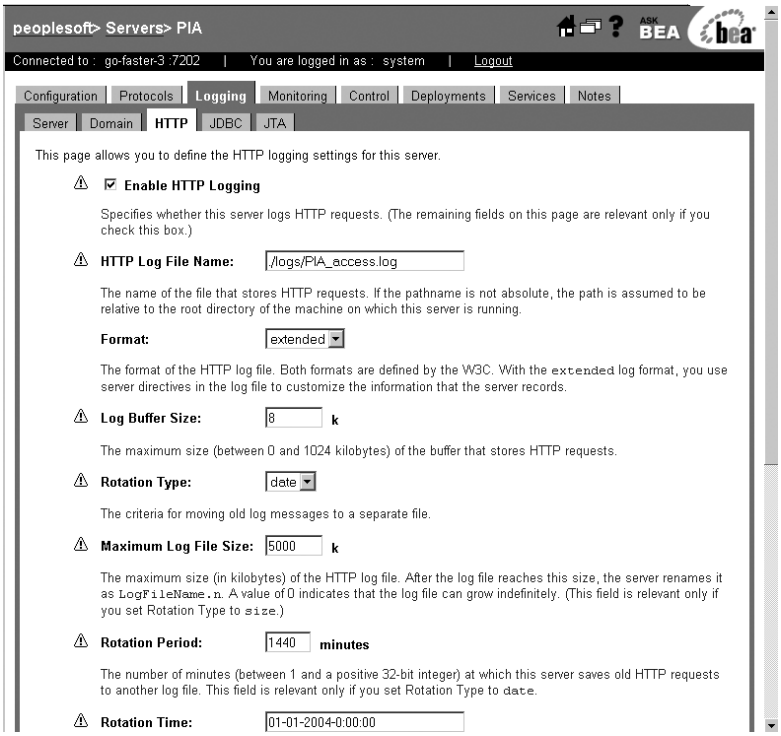


Figure 9-2. *Part of the WebLogic 8.1 console*

Defining the Access Log Format

In all versions of WebLogic, the format of the log file is defined by directives in the log file itself. The directives can be appended to the end of the log file. I normally create a seed log file and copy that to the access log file. When a log file rotates, the directives are rewritten at the top of the new file by WebLogic (see Listing 9-17).[5]

Listing 9-17. *Seed* `access.log`

```
#Version: 1.0
#Fields: date time time-taken bytes c-ip c-dns cs-method sc-status cs(User-
Agent) cs-uri-stem cs-uri-query
```

Table 9-1 describes the directives that I have used in Listing 9-17. It is not an exhaustive list.

5. I have encountered problems with some ports of WebLogic 5.1, in which the log file format is lost when the file rotates, and the new file is written with the default extended format. You can test for this by setting the rotation period to a small value.

Table 9-1. *W3C Extended Log File Formats*

Custom Log Format	Description
date	Date at which the transaction completed. The format is fixed as YYYY-MM-DD, where YYYY, MM, and DD stand for the numeric year, month, and day, respectively.
time	Time at which the transaction completed. The format is fixed as HH:MM:SS, where HH is the hour in 24-hour format, MM is minutes, and SS is seconds. All times are specified in GMT.
time-taken	Time taken for the transaction to complete in seconds. This is accurate to the limit of the operating system. On Windows NT, this is in 1/1,000 of a second.
bytes	Number of bytes transferred.
c-ip	Client IP address and port.
c-dns	DNS name of the client.
sc-status	Server to client status code.
cs-status	Client to server status code.
cs-method	Client to server method.
cs(User-Agent)	The user-agent string in the HTTP header in the message from the client.
cs-uri-stem	The stem portion of the URI requested by the client, omitting the query.
cs-uri-query	The query portion of the URI requested by the client.

The variables are fully described in the W3C standard to which the WebLogic documentation refers.

Listing 9-18 shows a sample of the resultant log for a PeopleTools 8.4 system.

Listing 9-18. *Extract of* `PIA_access.log`

```
#Version: 1.0
#Fields: date time time-taken bytes c-ip c-dns cs-method sc-status cs(User-
Agent) cs-uri-stem cs-uri-query
2004-04-20 17:25:38 0.551 7782 10.0.0.3 elgin GET 200 "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"
/psc/ps/EMPLOYEE/HRMS/s/WEBLIB_PT_NAV.ISCRIPT1.FieldFormula.IScript_PT_NAV_PAGEL
ET
Folder=HC_WORKFORCE_INFO&CREF=&FolderRef=HC_WORKFORCE_INFO&c=PfDKxdOQe8E/R8xQKXo
Cng==&Bodyid=true
2004-04-20 17:25:42 0.761 674 10.0.0.3 elgin GET 200 "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"
/psp/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL -
2004-04-20 17:25:42 0.52 2713 10.0.0.3 elgin GET 200 "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"
/psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL
PortalActualURL=http%3a%2f%2fgo-faster-
3%3a7201%2fpsc%2fps%2fEMPLOYEE%2fHRMS%2fc%2fADMINISTER_WORKFORCE_(GBL).JOB_DATA.
GBL&PortalContentURL=http%3a%2f%2fgo-faster-
```

```
3%3a7201%2fpsc%2fps%2fEMPLOYEE%2fHRMS%2fc%2fADMINISTER_WORKFORCE_(GBL).JOB_DATA.
GBL&PortalContentProvider=HRMS&PortalCRefLabel=Job%20Data&PortalRegistryName=EMP
LOYEE&PortalServletURI=http%3a%2f%2fgo-faster-
3%3a7201%2fpsp%2fps%2f&PortalURI=http%3a%2f%2fgo-faster-
3%3a7201%2fpsc%2fps%2f&PortalHostNode=HRMS&NoCrumbs=yes
```

The fields in the log file are tab-separated. The W3C standard states the following:

Each logfile entry consists of a sequence of fields separated by whitespace and terminated by a CR or CRLF sequence. The meanings of the fields are defined by a preceding `#Fields` *directive. If a field is omitted for a particular entry a single dash "-" is substituted.*[6]

This log can be loaded into a table in the database that is created as shown in Listing 9-19.

Listing 9-19. `wl_pre.sql`

```
CREATE TABLE weblogic
(timestamp      DATE             NOT NULL
,duration       NUMBER(9,3)      NOT NULL
,bytes_sent     NUMBER(7)        NOT NULL
,return_status  NUMBER(3)
,remote_host1   VARCHAR2(3)      NOT NULL
,remote_host2   VARCHAR2(3)      NOT NULL
,remote_host3   VARCHAR2(3)      NOT NULL
,remote_host4   VARCHAR2(3)      NOT NULL
,remote_dns     VARCHAR2(100)
,user_agent     VARCHAR2(1000)
,request_method VARCHAR2(8)
,request_status NUMBER(4)
,url            VARCHAR2(4000)   NOT NULL
,query_string1  VARCHAR2(4000)
,query_string2  VARCHAR2(4000)
,query_string3  VARCHAR2(4000)
,query_string4  VARCHAR2(4000)
,query_string5  VARCHAR2(4000)
,query_string6  VARCHAR2(4000)
,query_string7  VARCHAR2(4000)
,query_string8  VARCHAR2(4000)
,scenario       NUMBER DEFAULT 0
,domain         VARCHAR2(10)
);
```

6. See http://www.w3.org/TR/WD-logfile.html.

The WebLogic access log can then be imported into the table with the SQL*Loader control file shown in Listing 9-20.

Listing 9-20. *SQL*Loader control file* `wl.ldr`

```
LOAD DATA
INFILE 'PIA_access.log'
REPLACE
INTO TABLE weblogic
WHEN (1) != '#'
FIELDS TERMINATED BY WHITESPACE
TRAILING NULLCOLS
(timestamp      position(1:19) "TO_DATE(:timestamp,'YYYY-MM-DD HH24:MI:ss')"
,duration
,bytes_sent
,remote_host1   TERMINATED BY '.'
,remote_host2   TERMINATED BY '.'
,remote_host3   TERMINATED BY '.'
,remote_host4
,request_method
,return_status  NULLIF (return_status="-")
,user_agent
,url
,query_string1  TERMINATED BY '&' "SUBSTR(:query_string1,2)"
,query_string2  TERMINATED BY '&'
,query_string3  TERMINATED BY '&'
,query_string4  TERMINATED BY '&'
,query_string5  TERMINATED BY '&'
,query_string6  TERMINATED BY '&'
,query_string7  TERMINATED BY '&'
,query_string8  TERMINATED BY '&'
)
```

The query string is broken up into up to eight strings delimited by the ampersand character (&). The different parts of the query string contain information about which component and action is in use (although the query strings are very different in PeopleTools 8.1 and 8.4). Hence, it is possible to analyze PIA performance for different panels in the system.

For example, in Figure 9-3, I have looked at the cumulative execution time for different panels in the system and produced a graph of the top ten components. This required that I group the data by the first three parts of the query string.

HANDLING HEXADECIMAL CODES IN PEOPLETOOLS 8.4 ACCESS LOGS

The access log in PeopleTools 8.4 converts special characters back to their ASCII values in hexadecimal, so you get "%3a" instead of ":", for example. I find the data is easier to read if it is converted back to the characters. I do this with a trigger and a packaged function (as shown in Listing 9-21) that executes as the rows are inserted into the table with SQL*Loader.

Listing 9-21. dehex.sql

```
CREATE OR REPLACE PACKAGE dehex AS
FUNCTION dehex(p_string VARCHAR2) RETURN VARCHAR2;
PRAGMA restrict_references(dehex,wnds,wnps);
END dehex;
/

CREATE OR REPLACE PACKAGE BODY dehex AS
FUNCTION dehex(p_string VARCHAR2) RETURN VARCHAR2 IS
   l_string VARCHAR2(4000);
BEGIN
   l_string := p_string;
   WHILE INSTR(l_string,'%')>0 LOOP
      l_string :=
      SUBSTR(l_string,
         1,
         INSTR(l_string,'%')-1
      )
      ||CHR(
         TO_NUMBER(
            SUBSTR(l_string
               ,   INSTR(l_string,'%')+1
               ,   2
               )
            ,   'XXXXXXXX'
         )
      )
      ||SUBSTR(l_string
      ,   INSTR(l_string,'%')+3
      );
   END LOOP;
   RETURN l_string;
   END dehex;
END dehex;
/
```

```
CREATE OR REPLACE TRIGGER weblogic_query_string_dehex
BEFORE INSERT OR UPDATE ON weblogic
FOR EACH ROW
BEGIN
   :new.query_string1 := dehex.dehex(:new.query_String1);
   :new.query_string2 := dehex.dehex(:new.query_String2);
   :new.query_string3 := dehex.dehex(:new.query_String3);
   :new.query_string4 := dehex.dehex(:new.query_String4);
   :new.query_string5 := dehex.dehex(:new.query_String5);
   :new.query_string6 := dehex.dehex(:new.query_String6);
   :new.query_string7 := dehex.dehex(:new.query_String7);
   :new.query_string8 := dehex.dehex(:new.query_String8);
END;
/
;
```

The result is that the data loaded into the table is much easier to read, as shown in Listing 9-22.

Listing 9-22. *Access log after conversion by the* dehex *procedure*

```
17:26:07 20/04/2004   .39   7930   200 10  0   0   8
go-faster-3
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
GET
/psc/ps/EMPLOYEE/HRMS/s/WEBLIB_PT_NAV.ISCRIPT1.FieldFormula.IScript_PT_NAV_INFRA
ME
=PfDKxdOQe8E/R8xQKXoCng==
PortalActualURL=https://go-faster-3:7202/psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKF
ORCE_(GBL).JOB_DATA.GBL
PortalContentURL=https://go-faster-3:7202/psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORK
FORCE_(GBL).JOB_DATA.GBL
PortalContentProvider=HRMS
PortalRegistryName=EMPLOYEE
PortalServletURI=https://go-faster-3:7202/psp/ps/
```

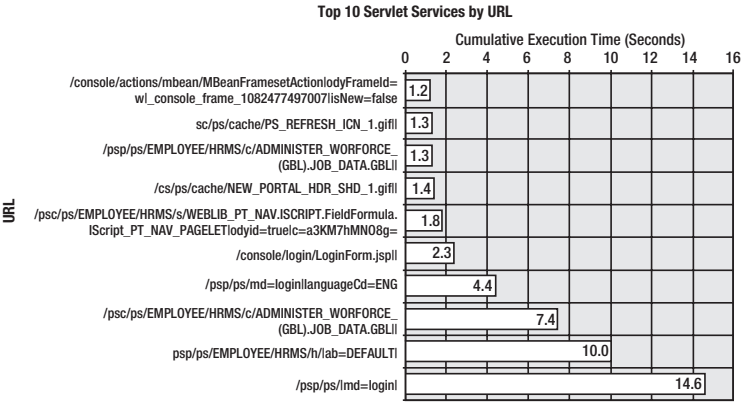


Figure 9-3. *Top ten operations by cumulative execution time*

Apache Web Server Access Log

PeopleSoft certified Apache 1.3 for PeopleTools 8.1*x* only. The Apache web server module `mod_log_config` performs access logging. Access logging is specified in the Apache configuration file `httpd.conf`.

The `LogFormat` command defines formats and associates them with aliases. The `CustomLog` command enables logging to a particular file with a particular format. For example, Listing 9-23 shows part of an Apache configuration file. A new format is created with the alias `monitoring`. The access log is then enabled with this format.

■ **Note** Pipe symbols (|) have been used to separate the values, as most other common separators may appear within the data.

Listing 9-23. *Extract of the Apache configuration file* `httpd.conf`

```
#
# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
#
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

#dmk - enhanced format for feed to monitoring tools
#dmk - this defines an alias called monitoring
#see http://httpd.apache.org/docs/mod/mod_log_config.html
LogFormat "%{%Y.%m.%d %H:%M:%S}t|%T|%B|%u|%h|%{User-Agent}i|%>s|%m|%U|%q" monitoring
```

```
#
# The location and format of the access logfile (Common Logfile Format).
# If you do not define any access logfiles within a <VirtualHost>
# container, they will be logged here. Contrariwise, if you *do*
# define per-<VirtualHost> access logfiles, transactions will be
# logged therein and *not* in this file.
#
#CustomLog logs/access.log common
CustomLog logs/access.log monitoring
```

The log file formats are fully documented in the `mod_log_config` section of the Apache documentation http://<web server>/manual/mod/mod_log_config.html, and on the Web at http://httpd.apache.org/docs/mod/mod_log_config.html. The example in Listing 9-23 uses the formats set out in Table 9-2.

Table 9-2. *Apache Access Log Formats*

Custom Log Format	Description
`%t` For instance, `%{%Y.%m.%d %H:%M:%S}t`	Time, in the form given by the format, that should be in `strftime(3)` format (potentially localized). `%Y` = Year as a four-digit number `%m` = Month of the year as a two-digit number `%d` = Day of the month as a two-digit number `%H` = Hour of the day as a two-digit number `%M` = Minutes as a two-digit number `%S` = Seconds as a two-digit number
`%T`	Time taken to serve the request, in seconds.
`%B`	Bytes sent, excluding HTTP headers.
`%u`	Remote user (from auth; may be bogus if return status [`%s`] is 401).
`%h`	Remote host.
`%{User-Agent}I`	Web browser identifier string.
`%>s`	Status. For requests that got internally redirected, this is the status of the *original* request; `%...>s` for the last.
`%m`	Request method.
`%U`	URL path requested.
`%q`	Query string (prepended with ? if a query string exists; otherwise, an empty string).

Listing 9-24 shows a sample of the output. Notice how the format of the URL request in PeopleTools 8.4 (see Listing 9-21) differs from PeopleTools 8.1.

Listing 9-24. *Extract of Apache* `access.log`

```
2002.02.26 09:57:06|0|67|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|200|GET|/peoplesoft8/cache/PT_TAB3MIXB8B090_ENG_1.GIF|
2002.02.26 09:57:06|0|187|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|200|GET|/peoplesoft8/cache/PS_FRA_COL_ENG_1.gif|
```

```
2002.02.26 09:57:06|0|214|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|200|GET|/peoplesoft8/cache/PT_PREVTAB_D_ENG_1.gif|
2002.02.26 09:57:06|0|275|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|200|GET|/peoplesoft8/cache/PT_NEXTTAB_ENG_1.gif|
2002.02.26 09:57:17|0|31847|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 4.0; Q312461)|200|POST|/servlets/iclientservlet|?ICType=Panel&Menu=ADMINISTER_
WORKFOR
CE_(GBL)&Market=GBL&PanelGroupName=JOB_DATA
2002.02.26 09:57:18|0|161|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|200|GET|/peoplesoft8/cache/PS_FRA_EX_ENG_1.gif|
2002.02.26 09:57:26|0|30959|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 4.0;
Q312461)|200|POST|/servlets/iclientservlet|?ICType=Panel&Menu=ADMINISTER_WORKFOR
CE_(GBL)&Market=GBL&PanelGroupName=JOB_DATA
2002.02.26 10:17:29|2|6007|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 4.0;
Q312461)|200|GET|/servlets/iclientservlet|?cmd=expire&languageCd=ENG
2002.02.26 10:17:30|0|0|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 4.0; Q312461)|304|GET|/peoplesoft8/signin.css|
2002.02.26 10:17:30|0|6007|-|127.0.0.1|Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 4.0; Q312461)|200|GET|/servlets/iclientservlet|?cmd=expire&languageCd=ENG
```

The logging can be made conditional on a variable. For example, in Listing 9-25, the `image` variable is set if the request is for a GIF or a JPG file. These objects are not logged by this CustomLog directive.

Listing 9-25. *Conditional access logging in Apache*

```
SetEnvIfNoCase Request_URI \.gif$ image
SetEnvIfNoCase Request_URI \.jpg$ image
CustomLog logs/access.log monitoring env=!image
```

The Apache log can be imported into a database table, in the same manner as shown with WebLogic, using SQL*Loader with the control file in Listing 9-26.

Listing 9-26. *SQL*Loader control file* `apache.ldr`

```
LOAD DATA
INFILE 'access.log'
REPLACE
INTO TABLE apache
FIELDS TERMINATED BY '|'
TRAILING NULLCOLS
(timestamp      "TO_DATE(:timestamp,'YYYY.MM.DD HH24:MI:ss')"
,duration
,bytes_sent
,FILLER status
,remote_host1   TERMINATED BY '.'
,remote_host2   TERMINATED BY '.'
```

```
,remote_host3   TERMINATED BY '.'
,remote_host4
,user_agent
,return_status  NULLIF (return_status="-")
,request_method
,url
,query_string1  TERMINATED BY '&' "SUBSTR(:query_string1,2)"
,query_string2  TERMINATED BY '&'
,query_string3  TERMINATED BY '&'
,query_string4  TERMINATED BY '&'
,query_string5  TERMINATED BY '&'
,query_string6  TERMINATED BY '&'
,query_string7  TERMINATED BY '&'
,query_string8  TERMINATED BY '&'
)
```

■ **Note** The duration in the Apache access log is expressed only in whole seconds. The time appears to be rounded—rather than truncated—to the nearest second. This makes it more difficult to examine exact timings.

Query Metrics

On more than one occasion I have encountered PeopleSoft systems that have been brought to their knees by unrestricted use of the ad hoc Query tool. It is relatively easy to write a query that functions correctly, but it is often more challenging to write one that also executes efficiently. Queries are also used as the data source by Crystal and nVision reports, and for database agent queries.

The question that needs to be answered is "Which queries are consuming the most time?" This section provides information on a number of techniques to obtain query metrics that can help you answer this question. PeopleSoft has added additional instrumentation to PeopleTools 8.4 to aggregate query execution time and to optionally log each query execution to the database. This new functionality is discussed in Chapter 10. Otherwise, this question has always been difficult to answer.

SQL Tracing Queries

If the Windows client PS/Query, Crystal, or nVision is run in two-tier mode, and it is possible to enable Oracle session trace with an AFTER LOGON database trigger, then the trace files could be processed with `tkprof`. However, you usually end up with a huge number of trace files to go through manually. I discuss methods of enabling SQL trace with PeopleTools in Chapter 11.

From PeopleTools 7.53, ad hoc queries from PeopleTools programs connecting to the application server in three-tier mode, including the PIA, are routed via a separate dedicated application server process, PSQRYSRV. Oracle session trace can be enabled on the database sessions for these servers. Now there are a limited number of trace files to be processed and examined.

However, SQL trace will only give you the SQL statement. If you have enabled `EnableDBMonitoring` in the application server, you can also find the call to `dbms_application_info` before the query in the trace, and so identify the user who submitted it. To identify the query defined in PeopleSoft, you need to query the PeopleTools tables (this method is described in Chapter 11). So it is still a lot of manual work.

Web Server Access Log

The web server access log provides some information about query performance, as detailed in the following sections.

PeopleTools 8.1

In PeopleTools 8.1, the enhanced web server access log will show the query name in the URL query string. In Listing 9-27, the query DMK took 1.312 seconds to execute.

Listing 9-27. *Web server access log entries for a query*

```
#Fields: date time time-taken bytes c-ip c-dns cs-method sc-status cs-uri-stem
cs-uri-query cs(User-Agent)
2004-05-21 13:50:48 1.312 30792  10.0.0.8 go-faster-3 GET 200
/servlets/iclientservlet/peoplesoft8/ ICType=Query&ICAction=ICQryNameURL=DMK
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
2004-05-21 13:50:51 0.271 31046 10.0.0.8 go-faster-3 POST 200
/servlets/iclientservlet/peoplesoft8/ ICType=Query "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
```

If the query returns more than 100 rows, only the first 100 rows will be displayed. The user must click a link to the next page to see more. The servlet will make another `ICQuery` request to the application server, but it only checks the PeopleTools system version number. All the data for the query is retrieved by the servlet in the first request and is kept in the Java memory pool, and the servlet does all the work to display the next page. Navigating between pages in a query generates entries in the access log for `ICType=Query`, but there will be no `ICAction` parameters.

Therefore, it is possible to aggregate the total amount of time for each query from the access log. Provided that there has been no queuing in the application server on query requests, this provides a good estimate for query execution time and so makes it possible to determine which queries are consuming the most time and require to be addressed.

Having loaded the access logs into a database table in the manner described earlier in this chapter, it is relatively easy to construct some SQL (as in Listing 9-28) to identify the long-running queries.

Listing 9-28. `topqry81.sql`

```
COLUMN durrank    FORMAT 90
COLUMN query_name FORMAT a30
SPOOL topqry81
SELECT *
```

```
FROM   (
       SELECT RANK() OVER (ORDER BY duration DESC) AS durrank
       ,      SUBSTR(query_string2,INSTR(query_string2,'=',1,2)+1) query_name
       ,      duration
       ,      executions
       FROM   (
              SELECT   query_string2
              ,        SUM(duration) duration
              ,        COUNT(*) executions
              FROM     weblogic
              WHERE    query_string1 = 'ICType=Query'
              AND      query_string2 IS NOT NULL
              GROUP BY query_string2
              )
       )
WHERE  durrank <= 20
ORDER BY durrank, executions DESC
/
SPOOL OFF
```

Listing 9-28 produces a simple report (see Listing 9-29) showing which PeopleSoft queries have accumulated the most execution time.

Listing 9-29. `topqry81.lst`*: PeopleSoft queries by cumulative execution time*

```
DURRANK QUERY_NAME                      DURATION EXECUTIONS
------- ------------------------------ ---------- ----------
      1 RCT_BUSCON                             26          3
      2 SAYE_APP_CLOSURES                      11          2
      3 ALLACCIDENTS                           10          2
      4 SBC_ISSUES                              6          1
      5 BB_JOB_FAMILY_DESCRITIONS               5          1
      5 PM_LVRS_MTH                             5          1
```

PeopleTools 8.4

In PeopleTools 8.4, the ad hoc query designer has been incorporated into the PIA, although the Windows-based version is still available. Queries can be developed and then run directly from the Query Manager component. A new browser window is spawned, as shown in Figure 9-4. Note that the format of the URL in the access log is slightly different from PeopleTools 8.1.

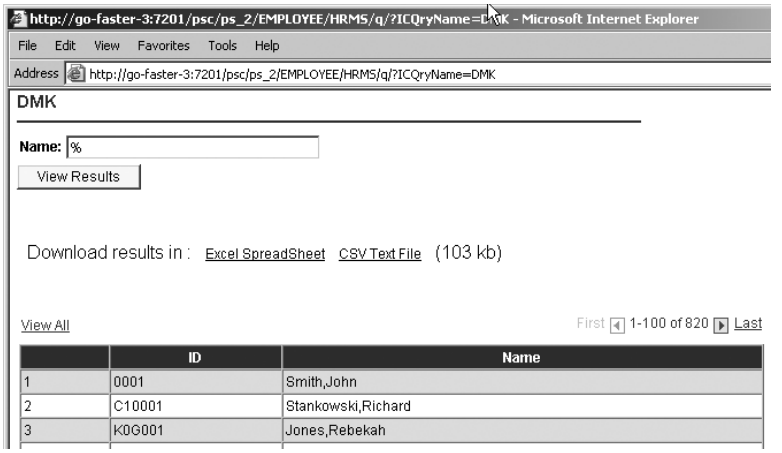


Figure 9-4. *Query results*

When the query is first run, and the data is actually selected from the database by the `ICQuery` service, there are two servlet requests. The query name is embedded in the URL query string, but it is prefixed with PUBLIC or PRIVATE depending upon the query type.

In Listing 9-30, the access log shows that a privately called DMK is executed, but the name of the Operator ID that owns it is not shown. Figure 9-4 shows that the query has a run-time parameter. The first entry in Listing 9-30 is for the parameter dialog; the second is to execute the query. If there is a parameter dialog before the query is executed, this activity also generates similar access log entries. Therefore, if different users execute their own private queries that happen to have the same name, you will not be able to distinguish between them, except by the IP address of the client.

Listing 9-30. *The owner of a private query does not appear in the access log*

```
2004-05-21 16:41:49 0.01 361 10.0.0.8 go-faster-3 GET 302 "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)" /psc/ps_newwin/EMPLOYEE/HRMS/q/
ICAction=ICQryNameURL=PRIVATE.DMK
2004-05-21 16:41:51 1.773 37274 10.0.0.8 go-faster-3 GET 200 "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)" /psc/ps_7/EMPLOYEE/HRMS/q/
ICAction=ICQryNameURL=PRIVATE.DMK
```

When the user navigates around the Query results, then there are still additional entries in the access log. For example, in Figure 9-4, only the first 100 rows of the result set are shown, and the user could navigate to the next 100 rows. The access log entries still have the name of the query in the URL query string, but this time without the query type prefix (see Listing 9-31).

Users will often take a public query and clone it as a private query, but keep the same name. You cannot tell from the access log which one the users running.

Listing 9-31. *Access log entries for navigating results, but not executing the query*

```
2004-05-21 16:41:55 0.481 53383 10.0.0.8 go-faster-3 POST 200 "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)" /psc/ps_7/EMPLOYEE/HRMS/q/ ICQryName=DMK
2004-05-21 16:41:57 0.16 41523 10.0.0.8 go-faster-3 POST 200 "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)" /psc/ps_7/EMPLOYEE/HRMS/q/ ICQryName=DMK
```

However, if a user is in the Query Manager, developing an ad hoc query, and they run the query by navigating to the Preview tab, then all that is reported in the access log is the component name (see Listing 9-32), and the specific query cannot be determined. So, the access log does not provide any specific information about the performance of queries run in the Query Manager.

Listing 9-32. *A query run in the Query Manager component—but which one?*

```
2004-05-21 16:37:18 0.35 86748 10.0.0.8 go-faster-3 POST 200 "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)"
/psc/ps/EMPLOYEE/HRMS/c/QUERY_MANAGER.QUERY_MANAGER.GBL -
```

In neither version of PeopleTools 8 does the web server access log provide a complete picture of query activity, although you can compute, for each query, the total amount of time spent entering parameters, executing the query, and navigating the results online. In version 8.4, PeopleSoft has instrumented the Query Manager to collect query execution statistics and put them in the database (see Chapter 10).

Batch Metrics

There are various sources of time-based statistics available from PeopleSoft batch and report processing that can be collected and analyzed. I describe these sources in this section of the chapter.

Process Scheduler

The Process Scheduler agent is responsible for initiating most batch and report processes in PeopleTools.

Request Table

Every time a process is scheduled to run via the Process Scheduler, a request record is created on various tables, but most notably on PSPRCSRQST. The process logs the start time and end time of the process on the request record in this table. From this, you know who ran what process and when, and how long it took; therefore, this table is a rich source of performance data within PeopleSoft. The Process Monitor page, shown in Figure 9-5, is essentially a query of this table.

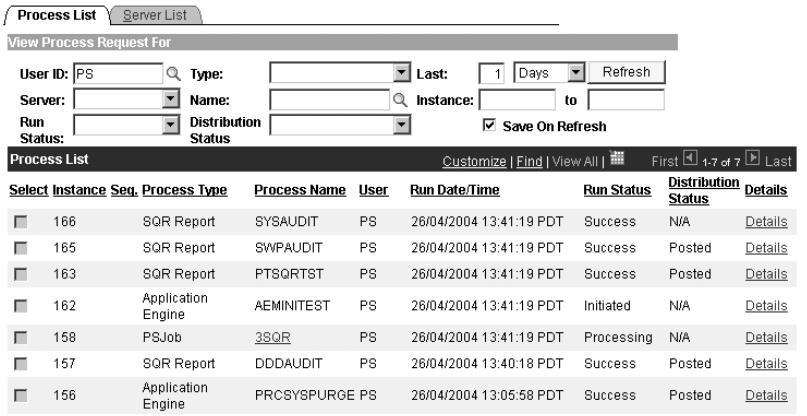


Figure 9-5. *Process Monitor summary page*

Within the Process Monitor, you can drill into a single process, as shown in Figure 9-6.

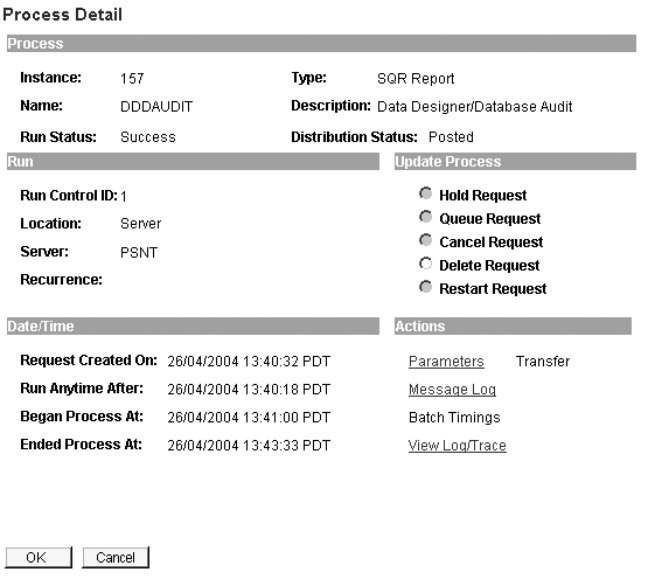


Figure 9-6. *Process Monitor detail page*

From PeopleTools 8.4, jobs are displayed differently in the Process Monitor. There are still separate rows on PSPRCSRQST for the job and the processes within the job, but the job can no longer be expanded within the Process Monitor. The job name is now a link that expands into another page, as shown in Figure 9-7.

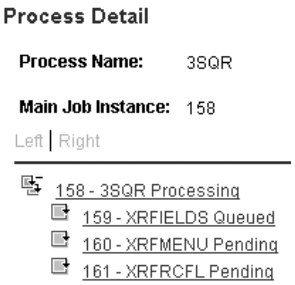


Figure 9-7. *Job detail*

All the information in the preceding three figures comes from the PSPRCSRQST table. Table 9-3 describes some of the more useful columns in that table.

Table 9-3. *PSPRCSRQST Columns*

Column Name	Description
PRCSINSTANCE	Process instance number. This uniquely identifies each process run through any process scheduler. This number is allocated sequentially.
PRCSTYPE	Type of process.
PRCSNAME	Name of the process.
SERVERNAMERUN	Name of the process scheduler upon which the process is run.
OPRID	ID of the operator who requested the process. This can be looked up against PSOPRDEFN.
RUNSTATUS	Status of the process request. When a process request is created, it is first given a status of 5 (Queued), and then when the Process Scheduler picks up the request and initiates it, the Process Scheduler updates the status to 6 (Initiated). When the process is started, the status is updated to 7 (Processing). When the process terminates, the status will change again to 9 (Success) or 2 (Error).
BEGINDTTM	Date and time at which the process initiated.
ENDDTTM	Date and time at which the process terminated.

More statuses have been added with successive releases. PeopleSoft has created translate values for the column RUNSTATUS that can be seen in the Application Designer or can be queried directly from PSXLATITEM, as shown in Listing 9-33.

Listing 9-33. *Translate values for RUNSTATUS*

```
>SELECT fieldvalue, eff_status, xlatlongname
  2  FROM psxlatitem i
  3  WHERE fieldname = 'RUNSTATUS'
  4  AND effdt = (
  5      SELECT max(effdt)
  6      FROM   psxlatitem i1
  7      WHERE  i1.fieldname = i.fieldname
  8      AND    i1.fieldvalue = i.fieldvalue
  9      AND    i1.effdt <= SYSDATE)
 10  ORDER BY TO_NUMBER(fieldvalue)
 11  /

FIELDVALUE EFF_STATUS XLATLONGNAME
---------- ---------- --------------------
1          A          Cancel
2          A          Delete
3          A          Error
4          A          Hold
5          A          Queued
6          A          Initiated
7          A          Processing
8          A          Cancelled
9          A          Success
10         A          Not Successful
11         I          Posted
12         I          Unable to post
13         I          Resend
14         I          Posting
15         I          Content Generated
16         A          Pending
17         A          Success With Warning
18         A          Blocked
19         A          Restart
```

In SQL, if one date is subtracted from another, the result is the difference in days. The difference between ENDDTTM and BEGINDTTM is the execution time of the process in days. This can easily be converted to hours, minutes, or seconds in a simple SQL query, such as the example in Listing 9-34.

■ **Caution** There is a flaw with this approach. If a process, usually a restartable Application Engine, fails and is restarted by the operator, BEGINDTTM will be the time at which the first attempt started, but ENDDTTM will be updated every time the process terminates. In this case, the difference between the two times is not a genuine execution time, and any calculations would be affected.

Listing 9-34. *Performance metrics from the Process Scheduler request table*

```
SELECT prcsinstance
,      prcstype
,      prcsname
,      TO_CHAR(TRUNC(SYSDATE)+(enddttm-begindttm),'HH24:MI:SS') exec_time
,      enddttm-begindttm exec_days
,      (enddttm-begindttm)*24 exec_hrs
,      (enddttm-begindttm)*1440 exec_mins
,      (enddttm-begindttm)*86400 exec_secs
FROM   psprcsrqst
...
/
PRCSINSTANCE PRCSTYPE                       PRCSNAME     EXEC_TIM  EXEC_DAYS
------------ ------------------------------ ------------ -------- ----------
  EXEC_HRS  EXEC_MINS  EXEC_SECS
---------- ---------- ----------
         156 Application Engine             PRCSYSPURGE  00:00:16 .000185185
.004444444 .266666667         16

         157 SQR Report                     DDDAUDIT     00:02:33 .001770833
     .0425       2.55        153
```

Hence, it is possible to write queries that are more sophisticated. Oracle's analytic functions may also be of assistance. Listing 9-35 is an example query that reports the five processes that have the longest cumulative execution time on successful completions in the last 30 days.

Listing 9-35. *Top five processes in the last 30 days*

```
SELECT x.*
--,    d.descr
FROM   (
       SELECT prcstype, prcsname
       ,      dur
       ,      avg_dur
       ,      freq
       ,      RANK() OVER (ORDER BY dur DESC) prcrnk
       FROM   (
              SELECT prcstype, prcsname
              ,      COUNT(*) freq
              ,      SUM(enddttm-begindttm)*86400 dur
              ,      AVG(enddttm-begindttm)*86400 avg_dur
              FROM   psprcsrqst p
              WHERE  begindttm IS NOT NULL
              AND    enddttm IS NOT NULL /*have completed*/
              AND    runstatus IN(9,11,12,13,14,15,16,17) /*successful proc*/
              AND    begindttm > SYSDATE - 30 /*in the last 30 days*/
              GROUP BY prcstype, prcsname
```

```
              )
       ) x
,      ps_prcsdefn d
WHERE  x.prcrnk <= 5
AND    d.prcstype(+) = x.prcstype
AND    d.prcsname(+) = x.prcsname
ORDER BY x.prcrnk
/
PRCSTYPE           PRCSNAME            DUR    AVG_DUR FREQ PRCRNK
------------------ ------------ ---------- ---------- ---- ------
SQR Report         SYSAUDIT          29333    14666.5    2      1
SQR Report         DDDAUDIT            284        142    2      2
SQR Report         SWPAUDIT            213        213    1      3
Application Engine PRCSYSPURGE          79       15.8    5      4
SQR Report         PTSQRTST             36         12    3      5
```

Purging the Process Scheduler Table

The Process Scheduler request table can grow at a significant rate; in some busy systems, I have seen thousands of requests per day. As the table grows, the performance of the Process Monitor and the Process Scheduler degrades. Therefore, it is important to purge this table of completed requests as aggressively as possible. Purging is vanilla PeopleSoft functionality.

If you are purging for the first time, or if you change the purge parameters so that you delete an unusually large amount of data, it may be advisable to rebuild the tables from which you have deleted data in order to reset their high-water marks. You can use the Application Designer to build an alter table script to do the job. Alternatively, you could use the `ALTER TABLE MOVE` command, and then rebuild the indexes that would have become UNUSABLE. Either method requires system downtime: not only must all Process Schedulers be stopped, but also the users must be prevented from scheduling new requests.

In PeopleTools 8.1, the purging is done with a delivered SQR, `PRCSPURG.sqr`. The number of days before purging is an attribute in the Process Scheduler definition. The SQR deletes requests that are older than this attribute.

In PeopleTools 8.4, there is an Application Engine process called PRCSYSPURGE. The details of the purge process are hidden within a number of PeopleCode functions called from this process.

Archiving Trigger

The data that is, and should be, purged by the Process Scheduler is the same data that I described earlier as a rich source of batch performance metrics, so it needs to be retained in a different table.

I suggest creating a copy of the PSPRCSRQST record in the Application Designer and building this as an archive table, in this case called PS_PRCSRQSTARCH. It is advisable to have at least the same primary key index on both tables. The easiest way is simply to duplicate the record in the Application Designer. Then, create a trigger that copies the record into the archive table when it is deleted from the request table.

Over the years, PeopleSoft has added some columns to this record. Rather than supply a script that creates the trigger, I have provided a script in Listing 9-36 that dynamically builds a script to create the trigger. Thus all the columns in PSPRCSRQST are built into the INSERT statement in the trigger.

Listing 9-36. prcsarch.sql*: Script to generate a script to create an archiving trigger*

```
rem prcsarch.sql
set head off trimout on trimspool on message off feedback off timi off echo off
spool prcsarch0.sql
SELECT 'CREATE OR REPLACE TRIGGER '||user||'.psprcsrqst_archive'
FROM   dual
;
SELECT 'BEFORE DELETE ON '||user||'.psprcsrqst'
FROM   dual
;
SELECT 'FOR EACH ROW'
FROM   dual
;
SELECT 'BEGIN'
FROM   dual
;
SELECT '   INSERT INTO '||user||'.ps_prcsrqstarch'
FROM   dual
;
SELECT '   '||DECODE(column_id,1,'(',',')||column_name
FROM   user_tab_columns
WHERE  table_name = 'PSPRCSRQST'
ORDER BY column_id
;
SELECT  '   ) VALUES'
FROM   dual
;
SELECT '   '||DECODE(column_id,1,'(',',')||':old.'||column_name
FROM   user_tab_columns
WHERE  table_name = 'PSPRCSRQST'
ORDER BY column_id
;
SELECT '   );'
FROM   dual
;
SELECT 'end;'
FROM   dual
;
SELECT '/'
FROM   dual
;
spool off
set head on message on feedback on echo on
spool prcsarch
@prcsarch0
show errors
set echo off
```

```
SELECT COUNT(*) psprcsrqst      FROM psprcsrqst;
DELETE                          FROM psprcsrqst WHERE runstatus=2;
SELECT COUNT(*) psprcsrqst      FROM psprcsrqst;
SELECT COUNT(*) ps_prcsrqstarch FROM ps_prcsrqstarch;
ROLLBACK;
spool off
```

The output of this script is shown in Listing 9-37.

Listing 9-37. `prcsarch0.sql`*: Script to create an archiving trigger*

```
CREATE OR REPLACE TRIGGER SYSADM.psprcsrqst_archive
BEFORE DELETE ON SYSADM.psprcsrqst
FOR EACH ROW
BEGIN
   INSERT INTO SYSADM.ps_prcsrqstarch
   (PRCSINSTANCE
...
   ,TIMEZONE
   ) VALUES
   (:old.PRCSINSTANCE
...
   ,:old.TIMEZONE
   );
end;
/
```

Queries on the performance data should now be run on the combination of both tables in order to get a complete view of performance. The tables can be combined within a query using an inline view, as shown in Listing 9-38.

Listing 9-38. *Query live and reporting table with an inline UNION ALL view*

```
SELECT ...
FROM   (
       SELECT * FROM psprcsrqst
       UNION ALL
       SELECT * FROM ps_prcsrqstarch
       )
/
```

■ **Tip** Use UNION ALL instead of UNION. The UNION operator will sort both result sets before combining them.

Statspack Trigger

I generally prefer to use SQL trace with the wait interface enabled, not least because the trace gives you accurate information about just one process, while systemwide statistics can be polluted by the background noise of other activity. In Chapter 11, I describe how a trigger on the Process Scheduler request table, PSPRCSRQST, can be used to enable SQL trace.

Nonetheless, there are occasions when Oracle's Statspack utility is of use:

- If you suspect that a process is being adversely affected by contention with other activity on the database—in particular, if SQL trace reports a large discrepancy between CPU and elapsed time that is not accounted for by wait events.
- If you do not have access to the `user_dump_dest` directory, because you are not the DBA.[7]
- Occasionally I have been in situations in which SQL trace cannot be used because enabling it has seriously aggravated performance problems.[8]

Statspack works by taking various snapshots from a number of Oracle system performance views (v$ views) and storing them. In any performance tuning exercise, it is important to collect measurements for exactly the thing in which you are interested. That means collecting information for the right process and for the right period of time. It is common to set up a database job to perform a snapshot on a regular basis. If you are examining the effect of batch processes, it can also be useful to collect additional snapshots when a process starts or finishes. This can be done with another trigger on the process request table.

When PeopleSoft batch processes are initiated by the Process Scheduler, they update their status on the process request record to 7, indicating that they are processing. When they terminate, they again update their status. The trigger created in Listing 9-39 takes a Statspack snapshot when a process starts and ends, hence it is possible to generate a Statspack report for almost exactly the period that a particular process was running.

Listing 9-39. *Trigger to generate Statspack snapshots when a process starts or ends*

```
spool snap_trigger
rollback;
rem requires following grants to be made explicitly by sys
rem GRANT EXECUTE ON perfstat.sysadm TO sysadm;

CREATE OR REPLACE TRIGGER sysadm.snap
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (  (new.runstatus = 7 AND old.runstatus != 7)
     OR (new.runstatus != 7 AND old.runstatus = 7)
     )
```

7. By default on Unix systems, the trace files in `user_dump_dest` are only readable by a member of the DBA group. However by setting the following undocumented initialization parameter, they become readable by anyone: `_trace_files_public = TRUE`. This parameter cannot be dynamically changed.
8. If you run Oracle on Microsoft Windows, make sure that the antivirus software excludes all Oracle files, including trace files!

```
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
   l_comment VARCHAR2(160);
BEGIN
   IF :new.runstatus = 7 THEN
      l_comment := 'Start';
   ELSE
      l_comment := 'End';
   END IF;

   l_comment := SUBSTR(:new.prcstype
                ||', '||:new.prcsname
                ||', '||:new.prcsinstance
                ||', '||l_comment
                ||', '||:new.oprid
                       ,1,160);

   perfstat.statspack.snap
      (i_snap_level=>5
      ,i_ucomment=>l_comment
      );
   COMMIT;
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

■ **Note** The autonomous transaction is required because `SYS.STATSPACK.SNAP()` includes a commit.

■ **Caution** Be careful that this trigger does not fire too often. You may need to restrict the trigger, perhaps to specific processes, in the same way as the trace trigger suggested in Chapter 11.

The comment on the snapshot can be seen when reports are generated in the usual way with the `$ORACLE_HOME/rdbms/admin/spreport.sql` script (see Listing 9-40).

Listing 9-40. *List of Statspack snapshots produced by* `spreport.sql`

```
Completed Snapshots

                          Snap                       Snap
Instance     DB Name         Id   Snap Started    Level Comment
------------ ------------ ----- ----------------- ----- ----------------------
hr88         HR88            92 22 Aug 2004 00:34     5 Application Engine, PR
                                                         CSYSPURGE, 338, Start,
                                                          PS
                             93 22 Aug 2004 00:36     5 Application Engine, PR
                                                         CSYSPURGE, 338, End, P
                                                         S
                             94 23 Aug 2004 01:01     5 Application Engine, PR
                                                         CSYSPURGE, 339, Start,
                                                          PS
                             95 23 Aug 2004 01:02     5 Application Engine, PR
                                                         CSYSPURGE, 339, End, P
                                                         S
```

System and Session Statistics Trigger

One of the problems with Statspack is that it reports only systemwide statistics. However, it is also possible to build a pair of triggers that capture database session and system statistics when a process begins and ends (see Listing 9-41). Only a small proportion of statistics is going to be of interest, and not all statistics report at a session level. Nonetheless, in the following example I have simply captured all the statistics.

A Global Temporary Table is used to capture the statistics at the start of the process, and it is only written to the final output directory when the process terminates. This handles the problem of matching start and end statistics, particularly when an Application Engine process has failed and been restarted.

Listing 9-41. *Triggers to collect* `v$sysstat` *and* `v$mystat` *for a process:* `gfc_sysstats.sql`

```
rem gfc_sysstats.sql
rem the triggers require the following grants to be issued by SYS
rem GRANT SELECT ON v_$sysstat  TO sysadm;
rem GRANT SELECT ON v_$mystat   TO sysadm;
rem GRANT SELECT ON v_$database TO sysadm;
ROLLBACK;
clear screen

DROP TABLE gfc_sys_stats_temp
/
CREATE GLOBAL TEMPORARY TABLE gfc_sys_stats_temp
(process_instance NUMBER        NOT NULL
,statistic#       NUMBER        NOT NULL
,db_value         NUMBER        NOT NULL
```

```
,my_value         NUMBER        NOT NULL
,begindttm        DATE          NOT NULL
)
ON COMMIT PRESERVE ROWS
/
CREATE INDEX gfc_sys_stats_temp
ON gfc_sys_stats_temp(process_instance, statistic#)
/

DROP TABLE gfc_sys_stats
/
CREATE TABLE gfc_sys_stats
(process_instance NUMBER        NOT NULL
,statistic#       NUMBER        NOT NULL
,db_value_before  NUMBER        NOT NULL
,my_value_before  NUMBER        NOT NULL
,begindttm        DATE          NOT NULL
,db_value_after   NUMBER        NOT NULL
,my_value_after   NUMBER        NOT NULL
,enddttm          DATE          NOT NULL
)
TABLESPACE users
/
DROP INDEX gfc_sys_stats
/
CREATE UNIQUE INDEX gfc_sys_stats
ON gfc_sys_stats(process_instance, statistic#, begindttm)
/

CREATE OR REPLACE TRIGGER sysadm.psprcsrqst_sys_stats_before
AFTER UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.runstatus = 7 AND old.runstatus != 7)
BEGIN
   INSERT INTO gfc_sys_stats_temp
   (      process_instance, statistic#
   ,      db_value, my_value
   ,      begindttm)
   SELECT :new.prcsinstance, s.statistics#
   ,      S.VALUE, M.VALUE
   ,      NVL(:new.begindttm,SYSDATE)
   FROM   v$sysstat s
   ,      v$mystat m
   WHERE  s.statistics# = m.statistic#
   ;
   EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

```
CREATE OR REPLACE TRIGGER sysadm.psprcsrqst_sys_stats_after
AFTER UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.runstatus != 7 and old.runstatus = 7)
BEGIN
   INSERT INTO gfc_sys_stats
   (      process_instance, statistic#
   ,      db_value_before, my_value_before, begindttm
   ,      db_value_after , my_value_after , enddttm
   )
   SELECT :new.prcsinstance, s.statistics#
   ,      b.db_value, b.my_value, b.begindttm
   ,      S.VALUE, M.VALUE
   ,      NVL(:new.enddttm,SYSDATE)
   FROM   v$sysstat s
   ,      v$mystat  m
   ,      gfc_sys_stats_temp b
   WHERE  s.statistics# = m.statistic#
   AND    b.statistic# = s.statistics#
   AND    b.statistic# = m.statistic#
   AND    b.prcsinstance = :new.prcsinstance
   ;
   --from PT8.4 AE may not shut down
   DELETE FROM gfc_sys_stats_temp
   WHERE  process_instance = :new.prcsinstance
   ;
   EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

■ **Note** The Global Temporary Table GFC_SYS_STATS_TEMP must be created ON COMMIT PRESERVE because the update that causes the trigger to fire is always committed so that the status can be seen in the Process Monitor and by the Process Scheduler.

The rows in the Global Temporary Table must be explicitly deleted by the trigger PSPRCSRQST_SYS_STATS_AFTER because from PeopleTools 8.4, the Application Engine is managed under Tuxedo and the same processes, hence the same session may handle many requests.

■ **Caution** The warning about frequency of execution on the Statspack trigger applies here too. These triggers can generate many metrics very quickly.

Message Log

The Application Engine and COBOL processes also write timestamped records to the message log table, PS_MESSAGE_LOG. These messages can be seen in the Process Monitor. When a process terminates abnormally, the Process Scheduler may write rows on behalf of the processes.

When a COBOL process is initiated by the remote call facility, it does not have a process instance, and there is no row on the process request table, PSPRCSRQST. However, the process will still write rows to the message log table, so this can be used to measure the execution time of remote call processes.

Process Scheduler Configuration

Two settings in the process scheduler configuration file (`$PS_HOME/appserv/prcs/<process scheduler name>/psprcs.cfg`) cause PeopleSoft COBOL and Application Engine processes to generate addition trace information. I describe these settings in the sections that follow.

COBOL Statement Timings

PeopleSoft still uses COBOL for much of its batch processing, in particular many processes in the Financials product and the Global Payroll calculation process. Some SQL is hard-coded within the COBOL, but many statements are read from the stored statements table, PS_SQLSTMT_TBL, and are then executed dynamically.

Setting the TraceSQL flag in the process scheduler configuration file to 128 (see Listing 9-42) will cause the COBOL process to produce a report detailing the aggregated elapsed time for each stored statement.

Listing 9-42. *Trace file settings: Extract from* `psprcs.cfg`

```
;------------------------------------------------------------------------
; SQL Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - SQL statements
; 2         - SQL statement variables
; 4         - SQL connect, disconnect, commit and rollback
; 8         - Row Fetch (indicates that it occurred, not data)
; 16        - All other API calls except ssb
; 32        - Set Select Buffers (identifies the attributes of columns
;             to be selected).
; 64        - Database API specific calls
; 128       - COBOL statement timings
; 256       - Sybase Bind information
; 512       - Sybase Fetch information
; 1024      - SQL Informational Trace
; 4096      - Manager information
; 8192      - Mapcore information
; Dynamic change allowed for TraceSql and TraceSqlMask
TraceSQL=128
```

There are some version differences to be aware of:

- This parameter can be dynamically changed from PeopleTools 8.4, so that trace settings can be changed without restarting the process scheduler or the Application Engine server processes.
- In PeopleTools 8.44, the options for bit values 1024, 4096, and 8192 were added.

Each COBOL process will produce its own trace file. The path and name of the file will be of the following form:

```
<$PS_HOME>\appserv\prcs\<process scheduler name>\log_output\
<short process type>_<prcsname>_<prcsinstance>\
<long process type>_<prcsname>_<prcsinstance>.trc
```

where:

- `$PS_HOME` is the root of the PeopleSoft installation, as chosen during the installation.
- `<procss scheduler name>` is the name of the Process Scheduler, for which there is a directory of the same name.
- `<short process type>` is a two- or three-letter code indicating the type of program: CBL for COBOL, SQR, or Application Engine.
- `<process type>` is a another string indicating the type of program: COBSQL for COBOL, SQR, or Application Engine.
- `<prcsname>` is the name of the process. This is up to eight characters.
- `<prcsinstance>` is the unique number that identifies each process request.

The report, shown in Listing 9-43, displays the amount of time taken by each stored and dynamic SQL statement in the process. Hence, you can determine where the most execution time was expended and direct any tuning effort accordingly.

Listing 9-43. `cobsql_ptptedit_2.trc`

```
PeopleSoft Batch Timings Report: cobsql_ptptedit_2.trc

Encoding Scheme Used: Ansi


                                    PeopleSoft Batch Statistics
                                      (All timings in seconds)

Encoding Scheme Used: Ansi

             R e t r i e v e   C o m p i l e   E x e c u t e   F e t c h   STMT TOTALS
Statement      Count    Time   Count   Time   Count   Time   Count   Time    Time % SQL

APIBNN             0    0.00       0    0.00      64    0.00       0    0.00    0.00  0.00
APISSB             0    0.00       0    0.00     110    0.00       0    0.00    0.00  0.00
COMMIT             0    0.00       0    0.00       8    0.08       0    0.00    0.08  1.23
```

```
CONNECT                0    0.00      0    0.00     12    1.62      0    0.00    1.62 25.05
DISCONNECT             0    0.00      0    0.00     12    0.00      0    0.00    0.00  0.00
PTPEDIT_U_HE000        0    0.00      1    0.00      1    0.01      0    0.00    0.01  0.15
PTPLOGMS_I_LOGMSG      2    0.00      2    0.00      2    0.19      0    0.00    0.19  2.93
PTPLOGMS_S_GETMSG      1    0.00      1    0.00      2    0.18      2    0.00    0.18  2.78
PTPLOGMS_S_MAXSEQ      1    0.00      1    0.00      1    0.01      1    0.00    0.01  0.15
PTPLOGMS_S_OPRDEFN     1    0.25      1    0.00      1    0.01      1    0.00    0.26  4.01
PTPSHARE_S_LODSHR1     1    0.04      1    0.00      3    0.35      3    0.00    0.39  6.02
PTPTEDIT_D_LOGFLD      1    0.00      1    0.00      1    0.03      0    0.00    0.03  0.46
PTPTEDIT_S_RECFLD      1    0.01      1    0.00      1    0.35      9    0.00    0.36  5.56
PTPTEDIT_U_EDITTBL     1    0.00      1    0.00      1    0.12      0    0.00    0.12  1.85
PTPTLREC_S_RECFLD      7    0.00      7    0.01      9    1.39     30    0.00    1.40 21.62
PTPTSFLD_U_FC440       0    0.00      7    0.00      7    0.06      0    0.00    0.06  0.93
PTPTSLOG_I_JA000       0    0.00      7    0.00      7    0.72      0    0.00    0.72 11.14
PTPTSREQ_S_RECDEFN     1    0.00      1    0.00      1    0.13      1    0.00    0.13  2.02
PTPTSTBL_S_RECFLD      1    0.02      1    0.00      1    0.00      9    0.00    0.02  0.31
PTPTSTBL_S_SUBREC      5    0.00      5    0.00      5    0.26      5    0.00    0.26  4.03
PTPTSWHE_S_RECDEFN     3    0.00      3    0.00      3    0.27      3    0.00    0.27  4.17
PTPUSTAT_S_JOBINST     2    0.01      2    0.00      2    0.00      2    0.00    0.01  0.15
PTPUSTAT_U_PRCQUE      1    0.00      1    0.00      1    0.03      0    0.00    0.03  0.46
PTPUSTAT_U_PRCRQSB     1    0.00      1    0.00      1    0.14      0    0.00    0.14  2.18
PTPUSTAT_U_PRCRQSE     1    0.12      1    0.00      1    0.06      0    0.00    0.18  2.78
                   ----- -------  ----- -------  ----- -------  ----- -------
Total:                31    0.45     46    0.01    257    6.02     66    0.00
                          ------         ------         -------         ------
Percent of Total:         6.95%          0.15%          92.90%          0.00%

                                 Time  % Total
Total in SQL:                    6.48    97.14
Total in Application:            0.19     2.86
                                -----
Total Run Time:                  6.67


Total Statements:                  25
Max Cursors Connected:             11
```

In PeopleTools 7.*x*, only the statements stored on PS_SQLSTMT_TBL were explicitly listed in the trace. There are other SQL statements that are dynamically built up into a string by the COBOL process before being submitted to the database. They were all aggregated into single entry in the trace file, described as "dynamic."

From PeopleTools 8, each dynamic statement is given a unique name, of a similar format to the stored statements, and reported separately in the trace file.

Application Engine Batch Timings

The AE_TRACE flag has a similar effect on Application Engine processes as TraceSQL does upon PeopleSoft COBOL processes. It is also set in the Process Scheduler configuration file `psprcs.cfg` (see Listing 9-44).

Listing 9-44. *Extract from* psprcs.cfg

```
;------------------------------------------------------------------------
; AE Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - Trace STEP execution sequence to AET file
; 2         - Trace Application SQL statements to AET file
; 4         - Trace Dedicated Temp Table Allocation to AET file
; 8         - not yet allocated
; 16        - not yet allocated
; 32        - not yet allocated
; 64        - not yet allocated
; 128       - Timings Report to AET file
; 256       - Method/BuiltIn detail instead of summary in AET Timings Report
; 512       - not yet allocated
; 1024      - Timings Report to tables
; 2048      - DB optimizer trace to file
; 4096      - DB optimizer trace to tables
TraceAE=1152
```

Setting TRACE_AE to 128 causes the Application Engine to produce a report of step timings, similar to that produced by the COBOL processes. Listing 9-45 shows a sample of this report. It shows how much time has been spent in each operation on each step, and how many times that operation has been executed. It also contains an analysis of the time spent in PeopleCode.

■ **Note** Remember, the timings in the PeopleSoft reports are elapsed times for the operations as measured in the client process. Therefore, they will also include any database waits.

Listing 9-45. AE_PER099_181.AET*: Application Engine trace file with batch timings*

```
-- PeopleTools 8.44.09 -- Application Engine
-- Copyright (c) 1988-2004 PeopleSoft, Inc.
-- All Rights Reserved
-- Database: HR88 (Oracle)


                    PeopleSoft Application Engine Timings
                           (All timings in seconds)
                             2004-05-04  23.59.06
```

```
                          C o m p i l e    E x e c u t e    F e t c h        Total
SQL Statement         Count   Time     Count   Time     Count   Time     Time
--------------------- ------- -------- ------- -------- ------- -------- --------

Application Engine

COMMIT                    0      0.0      78      4.1       0      0.0      4.1
                                                                          --------
                                                                               4.1

AE Program: PER099

COPY.Step01.S             5      0.0       5      0.7       0      0.0      0.7
COPY.Step02.S             5      0.0       5      2.1       0      0.0      2.1
COPY.Step03.S             5      0.0       5      3.3       0      0.0      3.3
COPY_LNG.Step01.S        54      0.0      54      0.3       0      0.0      0.3
COPY_LNG.Step02.S        54      0.0      54      0.0       0      0.0      0.0
COPY_LNG.Step03.S        54      0.0      54      0.0       0      0.0      0.0
DELETES.Step01.S          1      0.0       1      2.6       0      0.0      2.6
DELETES.Step02.S          1      0.0       1      0.3       0      0.0      0.3
DELETES.Step03.S          1      0.0       1      0.2       0      0.0      0.2
DELETES.Step04.S          1      0.0       1      0.1       0      0.0      0.1
INIT.Step02.S             1      0.0       1      0.0       1      0.0      0.0
INSERT.Step01.S           1      0.0       1      3.8       0      0.0      3.8
RELLANG.Step02.D          1      0.0       1      0.1      19      0.0      0.1
UPDATE.Step01.S           1      0.0       1      1.2       0      0.0      1.2
UPDATE.Step03.S           1      0.0       1      0.9       0      0.0      0.9
UPDATE.Step05.S           1      0.0       1      3.1       0      0.0      3.1
UPDATE.Step09.S           1      0.0       1      1.1       0      0.0      1.1
UPDATE.Step11.S           1      0.0       1      1.1       0      0.0      1.1
UPD_LNG.DELETE.S         18      0.0      18      0.5       0      0.0      0.5
UPD_LNG.DEPT.S           18      0.0      18      0.8       0      0.0      0.8
UPD_LNG.EMPLMT.S         18      0.0      18      0.2       0      0.0      0.2
UPD_LNG.INITROWS.S       18      0.0      18      0.5       0      0.0      0.5
UPD_LNG.JOBCODE.S        18      0.0      18      0.4       0      0.0      0.4
                                                                          --------
                                                                              23.5

-------------------------------------------------------------------------------

                              Call    Non-SQL  SQL      Total
PeopleCode                    Count   Time     Time     Time
----------------------------- ------- -------- -------- --------
```

```
AE Program: PER099

INIT.Step03                          1      0.0      0.0      0.0
                                       -------- -------- --------
                                            0.0      0.0      0.0

--------------------------------------------------------------------------------

                                                      E x e c u t e
PEOPLECODE Builtin/Method                             Count   Time
------------------------------------------------------ ------- --------
Boolean(Type 5) BuiltIns                                    1      0.0

--------------------------------------------------------------------------------

Total run time                :       48.4

Total time in application SQL :       27.7   Percent time in application SQL :       57.1%

Total time in PeopleCode      :        0.0   Percent time in PeopleCode      :        0.0%

Total time in cache           :        4.0   Number of calls to cache        :         11

--------------------------------------------------------------------------------

PeopleTools SQL Trace value: 128 (0x80)
PeopleTools PeopleCode Trace value: 0 (0)
Application Engine Trace value: 1152 (0x480)
Application Engine DbFlags value: 0 (0)
```

However, if AE_TRACE is set to 1024, then this information is stored on three tables in the database: PS_BAT_TIMINGS_DTL, PS_BAT_TIMINGS_FN, and PS_BAT_TIMINGS_LOG. These tables are updated only when a process completes successfully. If you want both the trace file and the batch timings enabled, set AE_TRACE to 1152 and both bits will be set.

These statistics can then be viewed in the Process Monitor (see Figure 9-8) for a given process instance.

However, this data can also be queried directly from the database, as shown in Listing 9-46. This query aggregates the timing data from all executions and reports the top 20 Application Engine steps by their cumulative execution time.

Once the AE_TRACE flag is set, a set of metrics will start to build up that describes the performance of all the Application Engine processing. Hence, it is possible to determine which steps are consuming the most time across the whole system.

Batch Timings - Summary

Process

Instance:	181	**Type:**	Application Engine
Name:	PER099	**Description:**	Fill EMPLOYEES Table

Time (in milliseconds)

Elapsed:	48400
In PeopleCode:	20
In SQL:	27660

Trace Level

Application Engine:	1152
SQL & PeopleCode:	128

Customize | Find | View All | First 1-25 of 25 Last

Program	Detail line identifer	Compile Count	Compile Time	Execute Count	Execute Time	Fetch Count	Fetch Time	PC Count	PC Time
AE Internal	COMMIT	0	0	78	4147	0	0	0	0
PER099	RELLANG.Step02.D	1	0	1	120	19	0	0	0
PER099	INIT.Step03.P	0	0	0	0	0	0	1	20
PER099	COPY.Step01.S	5	0	5	740	0	0	0	0
PER099	COPY.Step02.S	5	10	5	2124	0	0	0	0
PER099	COPY.Step03.S	5	0	5	3325	0	0	0	0
PER099	COPY_LNG.Step01.S	54	10	54	300	0	0	0	0
PER099	COPY_LNG.Step02.S	54	0	54	31	0	0	0	0
PER099	COPY_LNG.Step03.S	54	10	54	10	0	0	0	0
PER099	DELETES.Step01.S	1	0	1	2604	0	0	0	0
PER099	DELETES.Step02.S	1	0	1	301	0	0	0	0
PER099	DELETES.Step03.S	1	0	1	170	0	0	0	0
PER099	DELETES.Step04.S	1	0	1	110	0	0	0	0
PER099	INIT.Step02.S	1	0	1	40	1	0	0	0
PER099	INSERT.Step01.S	1	0	1	3786	0	0	0	0
PER099	UPDATE.Step01.S	1	0	1	1202	0	0	0	0
PER099	UPDATE.Step03.S	1	10	1	851	0	0	0	0
PER099	UPDATE.Step05.S	1	0	1	3084	0	0	0	0
PER099	UPDATE.Step09.S	1	0	1	1081	0	0	0	0

Figure 9-8. *Batch timings in Process Monitor*

Listing 9-46. `topae.sql`: *Top 20 Application Engine steps by cumulative total execution time*

```
SET PAUSE OFF AUTOTRACE OFF ECHO OFF PAGES 40 LINES 80
COLUMN stmtrank          HEADING 'Stmt|Rank'         FORMAT 99
COLUMN detail_id         HEADING 'Statement ID'      FORMAT a21
COLUMN pct_sqltime       HEADING '%|SQL|Time'        FORMAT 90.0
COLUMN pct_total_time    HEADING '%|Total|Time'      FORMAT 90.0
COLUMN cum_pc_sqltime    HEADING 'Cum %|SQL|Time'    FORMAT 90.0
COLUMN cum_pc_total_time HEADING 'Cum %|Total|Time' FORMAT 90.0
COLUMN executions        HEADING 'Execs'             FORMAT 9990
COLUMN compile_time      HEADING 'Compile|Time'      FORMAT 9990.0
COLUMN compile_count     HEADING 'Compile|Count'     FORMAT 9990
COLUMN fetch_time        HEADING 'Fetch|Time'        FORMAT 9990.0
COLUMN fetch_count       HEADING 'Fetch|Count'       FORMAT 9990
COLUMN retrieve_time     HEADING 'Retrieve|Time'     FORMAT 9990.0
COLUMN retrieve_count    HEADING 'Retrieve|Count'    FORMAT 9990
COLUMN execute_time      HEADING 'Exec|Time'         FORMAT 9990.0
COLUMN execute_count     HEADING 'Exec|Count'        FORMAT 9990
COLUMN ae_sqltime        HEADING 'AE|SQL|Time'       FORMAT 9990.0
```

```
COLUMN pc_sqltime        HEADING 'PC|SQL|Time'      FORMAT 9990.0
COLUMN pc_time           HEADING 'PC|Time'          FORMAT 990.0
COLUMN pc_count          HEADING 'PC|Count'         FORMAT 9990
spool topae
SELECT stmtrank
,      detail_id
,      execute_count
,      ae_sqltime
,      pc_sqltime
,      pc_time
,      ratio_sqltime*100 pct_sqltime
,      SUM(ratio_sqltime*100)
           OVER (ORDER BY stmtrank RANGE UNBOUNDED PRECEDING) cum_pc_sqltime
,      ratio_total_time*100 pct_total_time
,      SUM(ratio_total_time*100)
           OVER (ORDER BY stmtrank RANGE UNBOUNDED PRECEDING) cum_pc_total_time
FROM   (
       SELECT rank() OVER (ORDER BY sqltime desc) as stmtrank
       ,      a.*
       ,      RATIO_TO_REPORT(sqltime) OVER () as ratio_sqltime
       ,      RATIO_TO_REPORT(total_time) OVER () as ratio_total_time
       FROM   (
              SELECT bat_program_name||'.'||detail_id detail_id
              ,      COUNT(distinct process_instance) executions
--            ,      SUM(compile_time)/1000 compile_time
--            ,      SUM(compile_count) compile_count
--            ,      SUM(fetch_time)/1000 fetch_time
--            ,      SUM(fetch_count) fetch_count
--            ,      SUM(retrieve_time)/1000 retrieve_time
--            ,      SUM(retrieve_count) retrieve_count
              ,      SUM(execute_time)/1000 execute_time
              ,      SUM(execute_count) execute_count
              ,      SUM(peoplecodesqltime)/1000 pc_sqltime
              ,      SUM(peoplecodetime)/1000 pc_time
              ,      SUM(peoplecodecount) pc_count
              ,      SUM(execute_time +compile_time +fetch_time +retrieve_time)
                        /1000 ae_sqltime
              ,      SUM(execute_time +compile_time +fetch_time +retrieve_time
                        +peoplecodesqltime)/1000 sqltime
              ,      SUM(execute_time +compile_time +fetch_time +retrieve_time
                        +peoplecodesqltime +peoplecodetime)/1000 total_time
              FROM   ps_bat_timings_dtl a
              GROUP BY bat_program_name, detail_id
              ) a
       )
WHERE  stmtrank <= 20
/
spool off
```

The result is a simple report (see Listing 9-47) that tells you which Application Engine steps are consuming the most processing time.

Listing 9-47. `topae.lst`*: Report of top Application Engine statements*

```
                                        AE       PC            % Cum %     % Cum %
 Stmt                          Exec     SQL      SQL      PC   SQL   SQL Total Total
 Rank Statement ID            Count    Time     Time    Time  Time  Time  Time  Time
----- ---------------------- ----- -------- -------- ------ ----- ----- ----- -----
    1 HR_FASTVIEW.SEC_UPD.S     39      6.4      0.0    0.0  16.4  16.4   4.8   4.8
      tep01.D

    2 AE Internal.COMMIT       101      4.8      0.0    0.0  12.3  28.7   3.6   8.5
    3 PER099.INSERT.Step01.      1      3.8      0.0    0.0   9.7  38.5   2.9  11.4
      S

    4 PER099.COPY.Step03.S       5      3.3      0.0    0.0   8.6  47.0   2.5  13.9
    5 PER099.UPDATE.Step05.      1      3.1      0.0    0.0   7.9  54.9   2.3  16.2
      S
...
```

PeopleSoft Trace Files

All PeopleTools client and application server processes that connect to the database can produce PeopleTools trace files. The level of detail can be controlled, but they can show the following:

- The SQL submitted by the process
- Any PeopleCode executed
- The duration of the SQL execute or fetch operations
- The time since the last trace line was emitted

In this section, I discuss how PeopleTools trace can be enabled for various components in the PeopleSoft architecture and show how the trace files can be analyzed.

Application Designer and Client

The Configuration Manager, shown in Figure 9-9, enables the user to set trace flags that will be picked up by the Windows client, the Application Designer tool, and Data Mover.

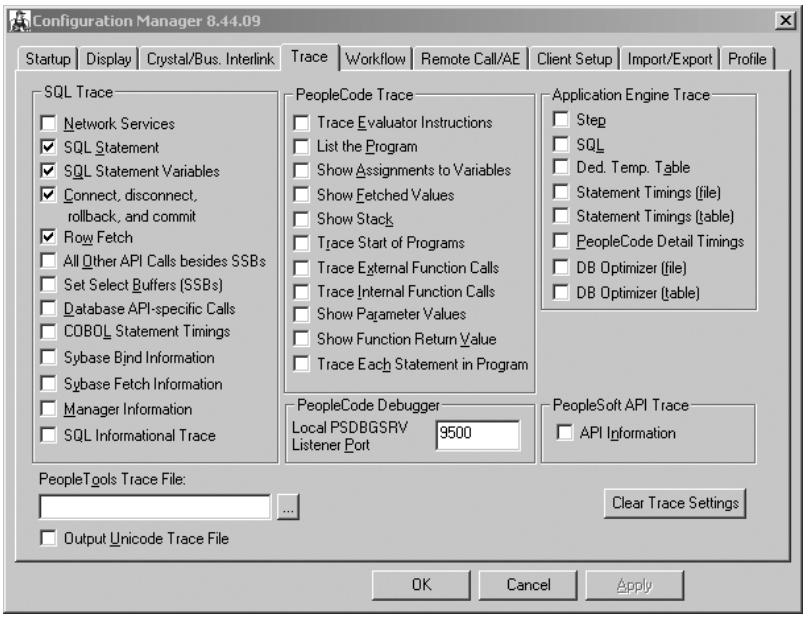


Figure 9-9. *Configuration Manager*

It is effectively a method of setting some registry keys that invoke the trace. Different Enterprise Release versions of PeopleTools write to different registry keys, and so can coexist. The three registry keys shown in Figure 9-10 correspond to the three sets of trace flags in Figure 9-9. Each check box corresponds to a bit in a binary number, and that number is stored in the registry.

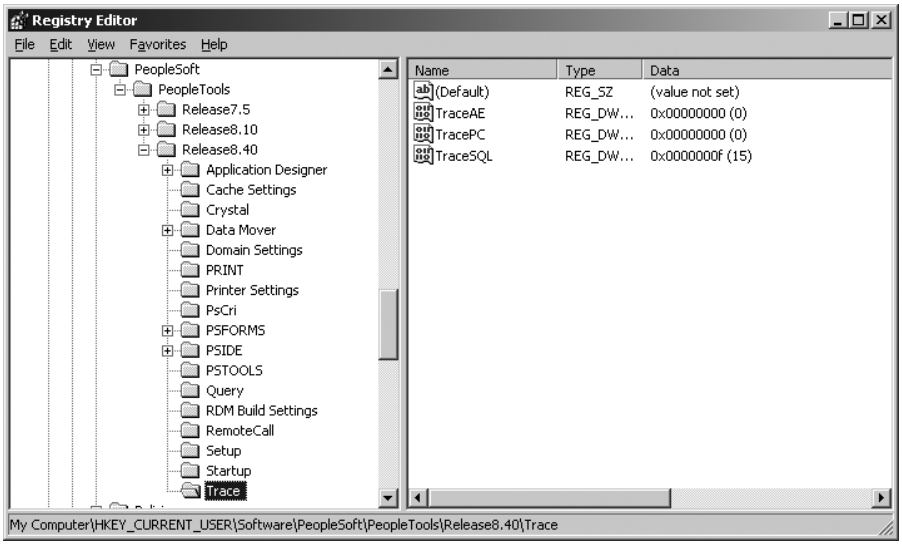


Figure 9-10. *PeopleTools trace registry settings*

The trace file that is subsequently produced is called DBG1.tmp and is written to the directory indicated by the Windows environment variable %TEMP%. Each line in the trace (see Listing 9-48) has an elapsed duration in seconds. This can be used to find the parts of the processing that are taking the most time.

Listing 9-48. *Extract of Windows client trace* DBG1.tmp

```
PeopleTools 8.44.09 Client Trace - 2004-05-05

PID-Line  Time      Elapsed Trace Data...
--------  --------  ------- -------------------->
1-1      00.46.06          Tuxedo session opened {oprid='PS',
appname='TwoTier', addr='//TwoTier:7000', open at0196E008, pid=3104}
 1-2      00.46.08    1.632 Cur#1.3104.HR88 RC=0 Dur=0.321
Connect=Primary/HR88/people/
...
1-6081   00.46.40    0.020 Cur#1.3104.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
DISPLAYONLY FROM PSOPROBJ A, PSOBJGROUP B WHERE B.ENTTYPE = :1 AND B.ENTNAME =
:2 AND A.CLASSID = :3 AND (A.OBJGROUPID = '*ALL DEFINITIONS*'   OR A.OBJGROUPID
= B.OBJGROUPID) UNION SELECT 2 FROM PSOBJGROUP WHERE ENTTYPE = :4 AND ENTNAME =
:5 ORDER BY 1
 1-6082   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Bind-1 type=1
length=1 value=J
 1-6083   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Bind-2 type=2
length=8 value=PATCH844
 1-6084   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Bind-3 type=2
length=7 value=HCPPALL
 1-6085   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Bind-4 type=1
length=1 value=J
 1-6086   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Bind-5 type=2
length=8 value=PATCH844
 1-6087   00.46.40    0.150 Cur#1.3104.HR88 RC=1 Dur=0.000 Fetch
 1-6088   00.46.40    0.000 Cur#1.3104.HR88 RC=0 Dur=0.000 Commit
```

Application Server

The application server can be traced in a similar way. The TraceSQL flag in the application server configuration file psappsrv.cfg (see Listing 9-49) will enable trace for all server processes.

Listing 9-49. *Extract from* psappsrv.cfg

```
;------------------------------------------------------------------------
; SQL Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - SQL statements
; 2         - SQL statement variables
; 4         - SQL connect, disconnect, commit and rollback
```

```
; 8         - Row Fetch (indicates that it occurred, not data)
; 16        - All other API calls except ssb
; 32        - Set Select Buffers (identifies the attributes of columns
;             to be selected).
; 64        - Database API specific calls
; 128       - COBOL statement timings
; 256       - Sybase Bind information
; 512       - Sybase Fetch information
; 1024      - SQL Informational Trace
; 4096      - Manager information
; 8192      - Mapcore information
; Dynamic change allowed for TraceSql and TraceSqlMask
TraceSql=15
TraceSqlMask=12319
```

When the server is started, the trace file will be generated as `<operator ID>_<server name>.tracesql`, where `operator ID` is the operator used to start the application server, specified in the Startup section of the configuration file. As each service request is received by the server process, the trace is switched to a new file called `<operator ID>_<client machine name>.tracesql` (see Listing 9-50). If the machine name cannot be determined, the IP address is used instead. So, a trace file is produced for each operator/machine combination.

Listing 9-50. *Extract of client trace file* `PS_go-faster-3.tracesql`

```
PSAPPSRV.4060   1-1853   19.52.52    0.060 Cur#1.4060.HR88 RC=0 Dur=0.030 COM
Stmt=Select COUNTRY, STATE from PS_PERSON_ADDRESS where ADDRESS_TYPE = 'HOME'
and EMPLID = :1
PSAPPSRV.4060   1-1854   19.52.52    0.000 Cur#1.4060.HR88 RC=0 Dur=0.000 Bind-1
type=2 length=1 value=
PSAPPSRV.4060   1-1855   19.52.52    0.541 Cur#1.4060.HR88 RC=1 Dur=0.000 Fetch
```

There is no heading on the application server log as there is on the client log, but the structure is essentially the same. Additional columns identify the server process name and process ID.

Enabling serverwide trace in this way is a very blunt instrument. You generate a lot of trace that is fairly difficult to process. I would be much more likely to enable Oracle SQL trace for all of the PSAPPSRV processes than PeopleTools SQL trace.

■ **Caution** There is also a significant performance overhead to this trace. Tracing fetches considerably increases the amount and overhead of trace. I have experienced up to 20% increase in response times. It should be used sparingly in a production environment, and only the elements of interest should be traced.

PIA Trace

Occasionally you will want to get metrics for a particular transaction. One method is to configure an application server with just a single PSAPPSRV process. You will also need a PIA domain that references only that application server. Then you know that all the activity will go through

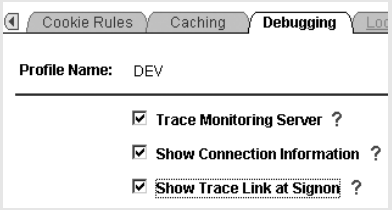
that application server process, and you can then enable SQL trace for that database session. It is not always possible to implement this quickly—or at all—in a production environment.

The alternative is to enable PeopleTools SQL trace for that session. This can be done in two ways. First, the operator can enable trace for their session at login. There is a link to an alternative login page from which trace options can be set, as shown in Figure 9-11.

Figure 9-11. *PIA login challenge*

SUPPRESSING THE TRACE LINK ON THE PIA SIGN-ON PAGE

From PeopleTools 8.44 onward, the link to the trace options on the login screen can be suppressed in the web profile, as shown in the following image (PeopleTools ➤ Web Profile ➤ Web Profile Configuration).



Up to PeopleTools 8.43, this link could be suppressed by setting `enableTrace=false` in the PIA servlet's `configuration.properties` file.

In any version, if the trace link has been suppressed, you can still get to the trace configuration page by manually adding `&trace=y` to the URL of the sign-on page, for example:

```
http://go-faster-3:7201/psp/ps/?cmd=login&trace=y
```

With each release of PeopleTools, more trace options have been added. In the following example (see Figure 9-12), I have chosen to trace all SQL statements, including fetch and commit operations. Bind variable values are also logged. Then all of the SQL issued by this user's session will be logged to a file in the application server log directory.

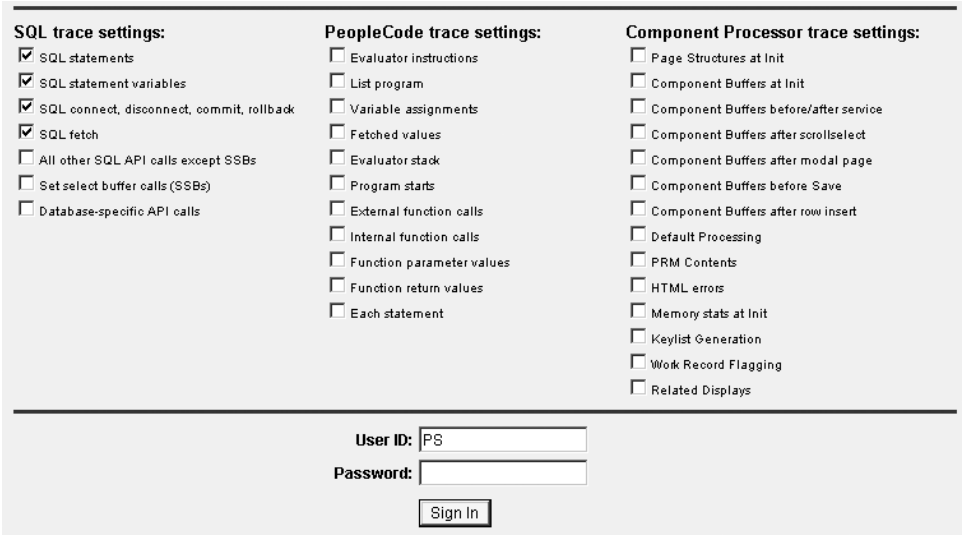


Figure 9-12. *PIA sign-on trace option (PT8.44)*

■ **Tip** If you are not sure which application server you are connected to, press Ctrl+J to get the address and port number of the Jolt Listener. You will need access to the application server's log directory in order to obtain the trace file.

Trace can be also enabled or disabled midsession by opening a new window, navigating to PeopleTools ➤ Utilities ➤ Debug ➤ Trace SQL (see Figure 9-13), and setting the trace flags as required. There is a separate page on the same menu to enable PeopleCode traces.

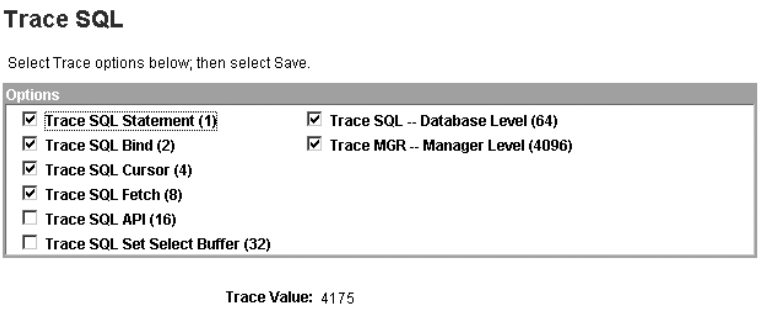


Figure 9-13. *Selecting trace options*

The earlier warnings about the impact on performance still apply. PeopleTools trace can seriously degrade application server performance. The resulting trace file is shown in Listing 9-51.

Listing 9-51. *Extract of the PeopleTools trace file* `PS_go-faster-3.tracesql`

```
PSAPPSRV.4060   1-1853   19.52.52  0.060 Cur#1.4060.HR88 RC=0 Dur=0.030 COM
Stmt=Select COUNTRY, STATE from PS_PERSON_ADDRESS where ADDRESS_TYPE = 'HOME'
and EMPLID = :1
PSAPPSRV.4060   1-1854   19.52.52  0.000 Cur#1.4060.HR88 RC=0 Dur=0.000 Bind-1
type=2 length=1 value=
PSAPPSRV.4060   1-1855   19.52.52  0.541 Cur#1.4060.HR88 RC=1 Dur=0.000 Fetch
```

Two times are recorded in the trace:

- The duration of the logged operation is recorded and prefixed with `Dur=`. This measures only in centiseconds on most platforms, although the number is formatted with three decimal places.
- The time since the last trace line was emitted is recorded after the time at which the line was emitted. It is accurate to milliseconds on Windows and centiseconds on most Unix systems. Idle time since the last service also seems to be accurate to milliseconds, even on Unix platforms.

So the SQL statement in line 1853 took 0.03 seconds to parse and execute, but it was 0.06 seconds since the last trace line was emitted. The difference suggests the time taken by processing in the application server process itself. The fetch operation at line 1855 reports a duration of 0, so it took less than 1millisecond to execute, but it was 0.541 seconds since the last trace line was emitted. Therefore, this was time consumed in the PSAPPSRV process, handling the result of the fetch. The return code of 1 indicates that nothing was returned by the fetch.

Be aware that sometimes the duration is incorrectly reported in the PeopleTools trace as zero, as in the following example, where a rowset is populated by PeopleCode:

```
   &PnlField_Rs = CreateRowset(Record.CO_PNLFIELD_VW);
   &PnlField_Rs.Flush();
   &PnlField_Rs.Fill("WHERE PNLNAME = :1 and FIELDTYPE = 16 and LBLTYPE = 7
AND RECNAME = :2 and FIELDNAME = :3", %Page, &LinkRecName, &LinkFieldName);
```

The resulting SQL query will have the alias FILL. The duration of the statement will be reported as zero. The first SQL in any application service is a query on the PSVERSION table, so a rowset FILL command is unlikely to be the first SQL in a service. Therefore, there is no pause waiting for user action, so the duration can safely be assumed to be equal to the time since the last trace line was emitted.

```
PSAPPSRV.2564    1-4321   01.52.36    0.551 Cur#1.2564.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
FILL.PNLNAME,FILL.PNLFLDID,FILL.FIELDNUM,FILL.PNLFIELDNAME,FILL.FIELDTYPE,FILL.R
ECNAME,FILL.FIELDNAME,FILL.LBLTYPE,FILL.GOTOPORTALNAME,FILL.GOTONODENAME,FILL.GO
TOMENUNAME,FILL.GOTOPNLGRPNAME,FILL.GOTOMKTNAME,FILL.GOTOPNLNAME,FILL.GOTOPNLACT
ION FROM PS_CO_PNLFIELD_VW FILL  WHERE PNLNAME = :1 and FIELDTYPE = 16 and
LBLTYPE = 7 AND RECNAME = :2 and FIELDNAME = :3
```

In this example, it took 0.551 seconds for PeopleSoft to generate and execute the SQL statement, and load the results into memory structures in the application server.

Analyzing PeopleSoft Trace Files

Sometimes, to help find the long-running statements, it can be useful to load the trace file into a table in the database (created by Listing 9-52).

Listing 9-52. *Extract from* `tracesql_pre.sql`

```
CREATE TABLE tracesql
(program         VARCHAR2(12)   DEFAULT 'Client' NOT NULL
,pid             NUMBER         DEFAULT 0        NOT NULL
,line_id         NUMBER                          NOT NULL
,line_num        NUMBER                          NOT NULL
,timestamp       DATE                            NOT NULL
,time_since_last NUMBER                          NOT NULL
,cursor          NUMBER                          NOT NULL
,database        VARCHAR2(10)                    NOT NULL
,return_code     NUMBER                          NOT NULL
,duration        NUMBER                          NOT NULL
,operation       VARCHAR2(4000)                  NOT NULL
,CONSTRAINT nonzero CHECK (duration>0 OR time_since_last>0)
);
```

The constraint can be used to prevent lines that do not account for any time from being loaded. The SQL*Loader control file in Listing 9-53 can be used.

Listing 9-53. *SQL*Loader control file* `tracesql.ldr`

```
LOAD DATA
INFILE 'PS_go-faster-3.tracesql'
REPLACE
INTO TABLE tracesql
FIELDS TERMINATED BY WHITESPACE
TRAILING NULLCOLS
(program                    TERMINATED BY '.'
,pid
,line_id                    TERMINATED BY '-'
,line_num
,timestamp       "TO_DATE(REPLACE(:timestamp,'.',':'),'HH24:MI:SS')"
,time_since_last
,cursor_lead     FILLER     TERMINATED BY '#'
,cursor                     TERMINATED BY '.'
,database
,return_lead     FILLER     TERMINATED BY '='
,return_code
,duration_lead   FILLER     TERMINATED BY '='
,duration
,operation       CHAR(4000) TERMINATED BY '&' "SUBSTR(:operation,1,4000)"
--,tracesql_seq SEQUENCE(MAX,1)
)
```

Traces from Windows client processes do not have a program name and process ID. Therefore, these columns should be removed from the SQL*Loader file when loading client trace files.

Aggregating SQL Statements with Different Literal Values

When working with nVision or Application Engine traces, it can be useful to remove literal values from SQL statements, so that time can be aggregated across SQL statements that are identical except for the literal values (the same principle as `CURSOR_SHARING=FORCE` in Oracle).

The packaged function shown in Listing 9-54 replaces all literal values with a colon (:).

Listing 9-54. *Extract from* `tracesql_pre.sql`

```
CREATE OR REPLACE PACKAGE cleansql AS
FUNCTION cleansql(p_operation VARCHAR2) RETURN VARCHAR2;
PRAGMA restrict_references(cleansql,wnds,wnps);
END cleansql;
/

CREATE OR REPLACE PACKAGE BODY cleansql AS
FUNCTION cleansql(p_operation VARCHAR2) RETURN VARCHAR2 IS
   l_newop     VARCHAR2(4000) := '';           --output string
   l_char      VARCHAR2(1)    := '';           --current char in input string
   l_inquote   BOOLEAN        := FALSE;        --are we in a quoted string
   l_inlitnum  BOOLEAN        := FALSE;        --are we in literal number
   l_lastchar  VARCHAR2(1)    := '';           --last char in output string
   l_len       INTEGER;                        --length of input string
   l_opsym     VARCHAR2(20)   := ' =<>+-*/,'; --string of symbols
   l_nextchar  VARCHAR2(1);                    --next character
   l_numbers   VARCHAR2(20)   := '1234567890'; --string of symbols
   l_pos       INTEGER        := 1;            --current pos in input string
BEGIN
   l_len := LENGTH(p_operation);
   WHILE (l_pos <= l_len) LOOP
      l_lastchar := l_char;
      l_char := SUBSTR(p_operation,l_pos,1);
      l_nextchar := SUBSTR(p_operation,l_pos+1,1);
      l_pos := l_pos+1;
      IF l_char = CHR(39) THEN -- we are on a quote mark
         IF l_inquote THEN -- coming out of quote
            l_inquote := FALSE;
         ELSE --going into quote
            l_inquote := TRUE;
            l_newop := l_newop||':';
         END IF;
         l_char := '';
      END IF;
      IF l_inquote THEN
         l_char := '';
```

```
      ELSIF (l_char = ' '
             AND INSTR(l_opsym,l_lastchar)>0) THEN --after symbol supress space
         l_char := '';
      ELSIF (l_lastchar = ' '
             AND INSTR(l_opsym,l_char)>0) THEN -- supress space before symbol
         l_newop := SUBSTR(l_newop,1,LENGTH(l_newop)-1)||l_char;
         l_char := '';
      END IF;

      IF (l_inlitnum) THEN --in a number
         IF (l_char = '.'
             AND INSTR(l_numbers,l_lastchar)>0
             AND INSTR(l_numbers,l_nextchar)>0) THEN
            l_inlitnum := TRUE; --still a number if a decimal point
         ELSIF (INSTR(l_numbers,l_char)=0) THEN -- number has finished
            l_inlitnum := FALSE;
         ELSE -- number continues
            l_char := '';
         END IF;
      ELSIF (NOT l_inlitnum
             AND INSTR(l_opsym,l_lastchar)>0
             AND INSTR(l_numbers,l_char)>0) THEN --start literal
         l_newop := l_newop||':';
         l_char := '';
         l_inlitnum := TRUE;
      END IF;

      l_newop := l_newop||l_char;

      IF l_newop = 'CEX Stmt=' THEN
         l_newop := '';
      END IF;
   END LOOP;
    RETURN l_newop;
END cleansql;
END cleansql;
/
show errors

CREATE OR REPLACE TRIGGER tracesql
BEFORE INSERT on tracesql
FOR EACH ROW
BEGIN
   :new.operation := cleansql.cleansql(:new.operation);
END;
/
;
```

For example, the SQL queries in Listing 9-55 generated by nVision are the same, but have different literal values.

Listing 9-55. *Raw PeopleTools trace generated by an nVision report*

```
CEX Stmt=SELECT L.TREE_NODE_NUM,A.PROJECT_ID,SUM(A.POSTED_TOTAL_AMT) FROM
PS_LEDGER_BUDG A, PSTREESELECT10 L, PSTREESELECT15 L1 WHERE A.LEDGER='BUDGET'
AND A.FISCAL_YEAR=2005 AND (A.ACCOUNTING_PERIOD BETWEEN 0 AND 1 OR
A.ACCOUNTING_PERIOD=998) AND A.BUSINESS_UNIT='XXXXX' AND  L.SELECTOR_NUM=143 AND
A.ACCOUNT>= L.RANGE_FROM_10 AND A.ACCOUNT <= L.RANGE_TO_10 AND (L.TREE_NODE_NUM
BETWEEN 511278162 AND 1548872078 OR L.TREE_NODE_NUM BETWEEN 1578947265 AND
1989974894) AND  L1.SELECTOR_NUM=140 AND A.PROJECT_ID>= L1.RANGE_FROM_15 AND
A.PROJECT_ID <= L1.RANGE_TO_15 AND L1.TREE_NODE_NUM BETWEEN 38461539 AND
2000000000 AND A.CURRENCY_CD='GBP' AND A.STATISTICS_CODE=' ' GROUP BY
L.TREE_NODE_NUM,A.PROJECT_ID
CEX Stmt=SELECT L.TREE_NODE_NUM,A.PROJECT_ID,SUM(A.POSTED_TOTAL_AMT) FROM
PS_LEDGER_BUDG A, PSTREESELECT10 L, PSTREESELECT15 L1 WHERE A.LEDGER='BUDAPP'
AND A.FISCAL_YEAR=2005 AND (A.ACCOUNTING_PERIOD BETWEEN 0 AND 1 OR
A.ACCOUNTING_PERIOD=998) AND A.BUSINESS_UNIT='XXXXX' AND  L.SELECTOR_NUM=143 AND
A.ACCOUNT>= L.RANGE_FROM_10 AND A.ACCOUNT <= L.RANGE_TO_10 AND (L.TREE_NODE_NUM
BETWEEN 511278162 AND 1548872078 OR L.TREE_NODE_NUM BETWEEN 1578947265 AND
b1989974894) AND  L1.SELECTOR_NUM=140 AND A.PROJECT_ID>= L1.RANGE_FROM_15 AND
A.PROJECT_ID <= L1.RANGE_TO_15 AND L1.TREE_NODE_NUM BETWEEN 38461539 AND
2000000000 AND A.CURRENCY_CD='GBP' AND A.STATISTICS_CODE=' ' GROUP BY
L.TREE_NODE_NUM,A.PROJECT_ID
```

However, the package function removed all the literals and any unnecessary spaces when the trace file was imported by SQL*Loader. So now you can aggregate time with a simple SQL query, as shown in Listing 9-56.

Listing 9-56. *Aggregated execution time for a SQL statement*

```
SELECT   operation, SUM(duration), MAX(duration), AVG(duration), COUNT(*)
FROM     tracesql
GROUP BY operation
HAVING SUM(duration)>300
/
OPERATION
--------------------------------------------------------------------------------
SUM(DURATION) MAX(DURATION) AVG(DURATION)   COUNT(*)
------------- ------------- ------------- ----------
SELECT L.TREE_NODE_NUM,A.PROJECT_ID,SUM(A.POSTED_TOTAL_AMT) FROM PS_LEDGER_BUDG
A,PSTREESELECT10 L,PSTREESELECT15 L1 WHERE A.LEDGER=: AND A.FISCAL_YEAR=: AND (A
.ACCOUNTING_PERIOD BETWEEN : AND : OR A.ACCOUNTING_PERIOD=:) AND A.BUSINESS_UNIT
=: AND L.SELECTOR_NUM=: AND A.ACCOUNT>=L.RANGE_FROM_10 AND A.ACCOUNT<=L.RANGE_TO
_10 AND (L.TREE_NODE_NUM BETWEEN : AND : OR L.TREE_NODE_NUM BETWEEN : AND :) AND
 L1.SELECTOR_NUM=: AND A.PROJECT_ID>=L1.RANGE_FROM_15 AND A.PROJECT_ID<=L1.RANGE
_TO_15 AND L1.TREE_NODE_NUM BETWEEN : AND : AND A.CURRENCY_CD=: AND A.STATISTICS
_CODE=: GROUP BY L.TREE_NODE_NUM,A.PROJECT_ID
     1468.509      965.003    112.962231         13
```

Summary

If performance tuning is a search for lost time, then you need ways to determine where time is being lost. Oracle's SQL trace, augmented by the wait interface, is a very effective tool at the database level, but PeopleSoft delivers a huge stack of technology that sits between the database and the user.

The metrics described in this chapter can identify which parts of the application, in which tier, are consuming the most time. Some problems are not caused by SQL performance, in which case the trace will tell you that the performance bottleneck is not the database, and you will not be much further forward.

Fortunately, many problems do come down to SQL performance, in which case PeopleSoft metrics will indicate which processes, and sometimes which step within a process, that you should examine in more detail. You can then choose a specific process or transaction to trace. Chapter 11 looks at how to use SQL trace in PeopleSoft in more detail.

CHAPTER 10

PeopleTools Performance Utilities

In PeopleTools 8.4, PeopleSoft has started to introduce some utilities to help detect and identify performance problems in the online part of the application. In this chapter, I discuss some aspects of these utilities.

This chapter does not seek to replace PeopleSoft's documentation. Instead, you are advised to read this chapter in conjunction with PeopleBooks.

Query Metrics in PeopleTools 8.4

In PeopleTools 8.4, PeopleSoft introduced two new methods of collecting query execution times, query statistics and query logging.

Query Statistics

Whenever a query is executed by PeopleTools,[1] the new average execution time, fetch time, and number of rows for that query, across all executions, is either inserted into or updated on the table PSQRYSTATS, as shown in Listing 10-1.

Listing 10-1. *How PeopleSoft updates the query statistics*

```
UPDATE PSQRYSTATS
SET    AVGEXECTIME = (AVGEXECTIME * EXECCOUNT + :4)/(EXECCOUNT + 1)
,      AVGFETCHTIME = (AVGFETCHTIME * EXECCOUNT + :5)/(EXECCOUNT + 1)
,      AVGNUMROWS = (AVGNUMROWS * EXECCOUNT + 0)/(EXECCOUNT + 1)
,      LASTEXECDTTM = TO_DATE(SUBSTR(:3, 0, 19),'YYYY-MM-DD-HH24.MI.SS')
,      EXECCOUNT = EXECCOUNT + 1
WHERE  OPRID = :1
AND    QRYNAME = :2
```

1. PeopleSoft Incident: 693629000: "Updates to PSQRYSTATS table causing nVision reports to bring back 0 values." The Query Statistics feature has been removed from 8.44.08 because it interfered with execution of queries, causing them to timeout, hang, or return incorrect results. This feature will be reintroduced in 8.45 with a User Configurable Option.

The limitation of this technique is that you can only see an average for a query since the averages were last deleted from the table. Query logging, described in the next section, provides information about each query execution.

The content of this table can be queried via the PeopleTools ➤ Utilities ➤ Administration ➤ Query Administration component (see Figure 10-1).

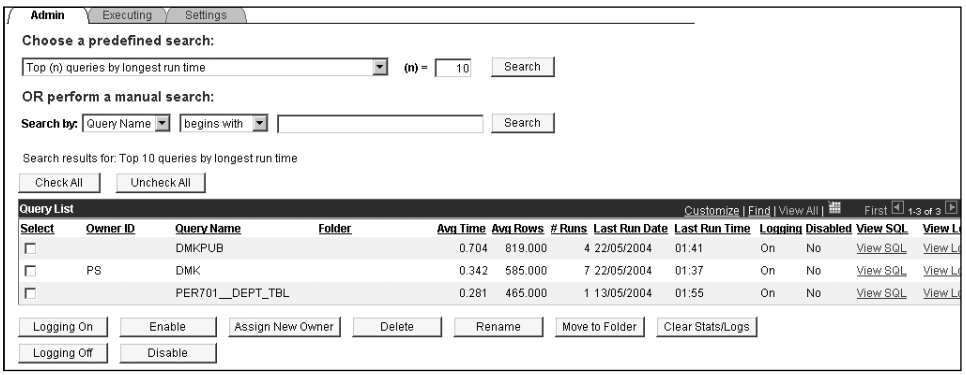


Figure 10-1. *Query Administration component*

Although the component can list the top *n* queries by cumulative execution time, it does not calculate total execution time. Queries are listed by average rather than cumulative execution time. However, it is easy to query this table in SQL, as shown in Listing 10-2.

Listing 10-2. `top10qry.sql`*: Top ten queries by cumulative execution time*

```
SELECT *
FROM   (
       SELECT RANK() OVER (ORDER BY tottime DESC) as qryrank
       ,      oprid, qryname, totexec, tottime
       ,      100*RATIO_TO_REPORT(tottime) OVER () as pcttime
       FROM   (SELECT oprid, qryname
              ,       SUM(execcount) totexec
              ,       SUM(execcount*avgexectime) tottime
              FROM    psqrystats
              GROUP BY oprid, qryname
              ) a
       )
WHERE  qryrank <= 10
/
```

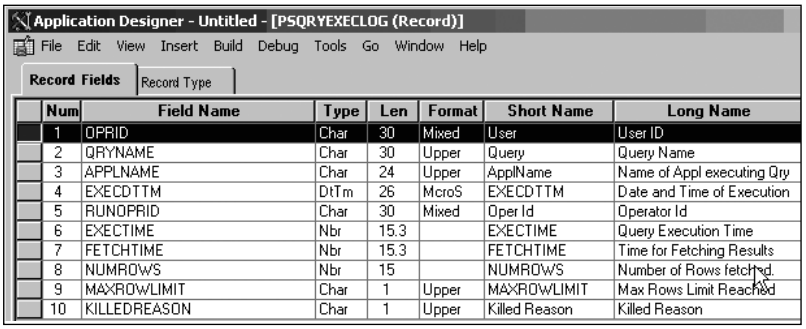
The results of the query are shown in Listing 10-3.

Listing 10-3. top10qry.lst

```
QRYRANK OPRID      QRYNAME                          TOTEXEC    TOTTIME PCTTIME
------- ---------- ------------------------------ ---------- ---------- -------
      1            DMKPUB                                4        2.8    51.3
      2 PS         DMK                                   7        2.4    43.6
      3            PER701__DEPT_TBL                      1        0.3     5.1
```

Query Logging

Also from PeopleTools 8.4, you can choose to log each execution of particular queries. Logging can be enabled for selected queries in the Query Administration component (shown in Figure 10-1). Every time the query runs, a row is written to the table PSQRYEXECLOG, as shown in Figure 10-2.



Num	Field Name	Type	Len	Format	Short Name	Long Name
1	OPRID	Char	30	Mixed	User	User ID
2	QRYNAME	Char	30	Upper	Query	Query Name
3	APPLNAME	Char	24	Upper	ApplName	Name of Appl executing Qry
4	EXECDTTM	DtTm	26	McroS	EXECDTTM	Date and Time of Execution
5	RUNOPRID	Char	30	Mixed	Oper Id	Operator Id
6	EXECTIME	Nbr	15.3		EXECTIME	Query Execution Time
7	FETCHTIME	Nbr	15.3		FETCHTIME	Time for Fetching Results
8	NUMROWS	Nbr	15		NUMROWS	Number of Rows fetched
9	MAXROWLIMIT	Char	1	Upper	MAXROWLIMIT	Max Rows Limit Reached
10	KILLEDREASON	Char	1	Upper	Killed Reason	Killed Reason

Figure 10-2. *PSQRYEXECLOG as shown by the Application Designer*

You could reasonably choose to enable logging for every query in the system. You can do this in the Query Administration component, or you can set the flag on the query definition directly with the SQL in Listing 10-4.

Listing 10-4. qrylogall.sql*: Enable query logging for all queries*

```
UPDATE pslock
SET    version = version + 1
WHERE  objecttypename IN('SYS','QDM');

UPDATE psversion
SET    version = version + 1
WHERE  objecttypename IN('SYS','QDM');

UPDATE psqrydefn
SET    execlogging = 'Y'
,      version = (SELECT version FROM pslock WHERE objecttypename = 'QDM')
WHERE  execlogging != 'Y';
```

PSQRYEXEC records the Operator ID that ran the query in RUNOPRID, and hence you could find out not only which queries are consuming the most time, but also which operators are spending the most time running queries.

The SQL query in Listing 10-5 will produce a similar report to that based on PSQRYSTATS in Listing 10-2, but now you know when the query was run, so you can look at a particular time window, which in this example is the last 7 days.

Listing 10-5. `top10logqry.sql`*: Top ten queries in the last 7 days*

```
SELECT *
FROM   (
       SELECT RANK() OVER (ORDER BY tottime DESC ) as qryrank
       ,      oprid, qryname, totexec, tottime
       ,      100*RATIO_TO_REPORT(tottime) OVER () as pcttime
       FROM   (SELECT oprid, qryname
              ,       COUNT(*) totexec
              ,       SUM(exectime) tottime
              FROM    psqryexeclog
              WHERE   execdttm > SYSDATE - 7
              GROUP BY oprid, qryname
              ) a
       )
WHERE  qryrank <= 10
/
```

PeopleSoft Ping

The PeopleSoft Ping utility was introduced in PeopleTools 8.42 (and it has been backported to PeopleTools 8.19, although the PIA graph capability is not available in PeopleTools 8.1*x*). It is a PIA component (shown in Figure 10-3) that runs a standard transaction and collects the elapsed times for database, application server, web server, and browser response times.

The PeopleSoft documentation[2] explains how to run Ping. Here I want to discuss how it can be used, what it tells you and, more important, what it does not tell you.

■ **Note** Throughout this chapter, unless otherwise stated, *Ping* refers to the PeopleSoft Ping utility, and not the network ping utility.

---

2. See PeopleTools PeopleBook (select Server Tools ➤ PeopleTools Utilities).

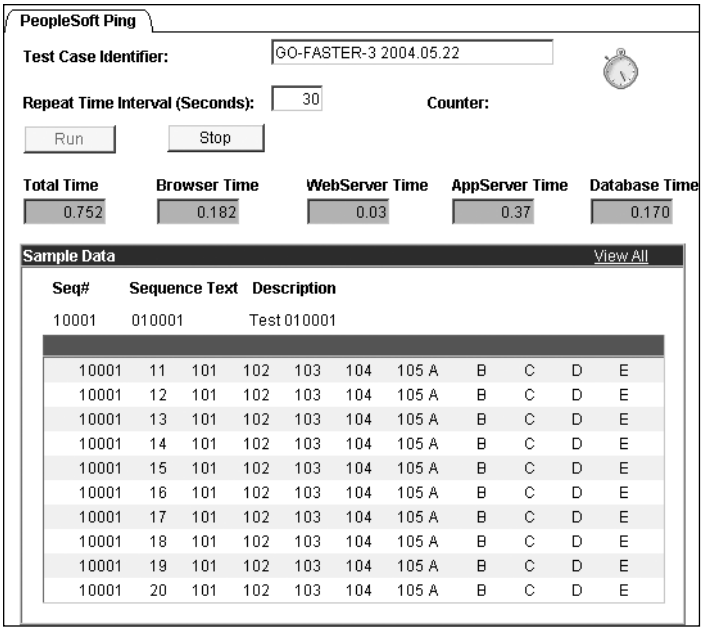


Figure 10-3. *Ping component*

The time between Ping transactions can be set on the page (it defaults to 30 seconds). The page will then repeatedly Ping at that frequency until it is stopped. The times are then stored in the database in the table PS_PTP_TST_CASES, whose structure is described in Table 10-1.

Table 10-1. *Columns on PS_PTP_TST_CASES*

Column Name	Description
PTP_TEST_CASE_ID	Test case identifier
PTP_DTTM_A1	Start time of the Ping transaction in PIA JavaScript on the client browser (according to the clock on the client)
PTP_DTTM_A2	End time of the Ping transaction in PIA JavaScript on the client browser (according to the clock on the client)
PTP_DTTM_B1	Start time of the Ping transaction in the application server
PTP_DTTM_B2	End time of the Ping transaction in the application server
PTP_DTTM_C1	Start time of the Ping transaction on the database
PTP_DTTM_C2	End time of the Ping transaction on the database
PTP_TIME_D	Elapsed database server time
PTP_AS_TIME	Elapsed application server time (including database time)
PTP_TOTAL_TIME	Total duration of Ping as measured on PIA, including application server time
PTP_WS_TIME	Elapsed web server duration (including application server time)

You can obtain a graph of the data in PeopleTools, but once it has any large spikes, the scale of the y-axis is automatically adjusted to include the whole range, and you can't see much detail. I prefer to extract the Ping data directly from the database into an Excel spreadsheet and produce a chart (see Figure 10-4). This has the advantage of my being able to contol the format of the chart and ranges exactly as desired.

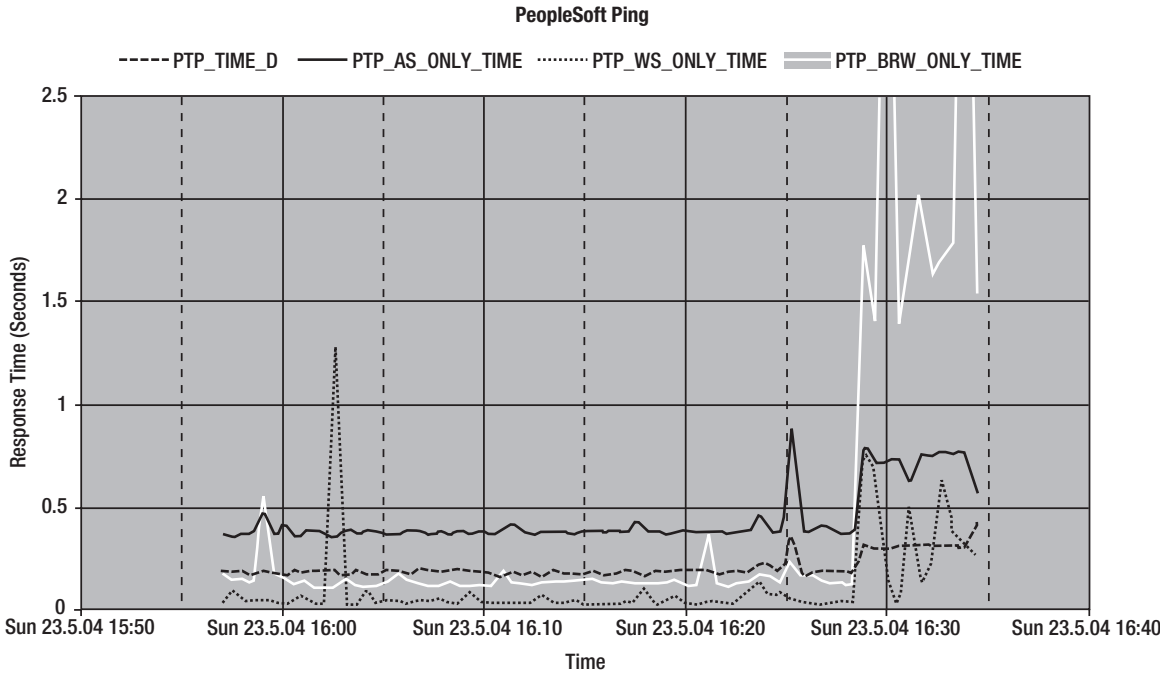


Figure 10-4. *Sample Excel chart of Ping data (*`psping.xls`*)*[3]

What Does Ping Measure?

It is important to understand how, and therefore just what, PeopleSoft Ping measures. Figure 10-5 illustrates the PIA, with the performance measures described in Chapter 9. The four measures collected by the Ping utility have been added.

3. This set of data was produced on a PeopleSoft system running entirely on a stand-alone laptop. At 16.28, a long-running query with a large sort was started. Both the disk and CPU were heavily loaded, so the response time of all components in the system degraded.

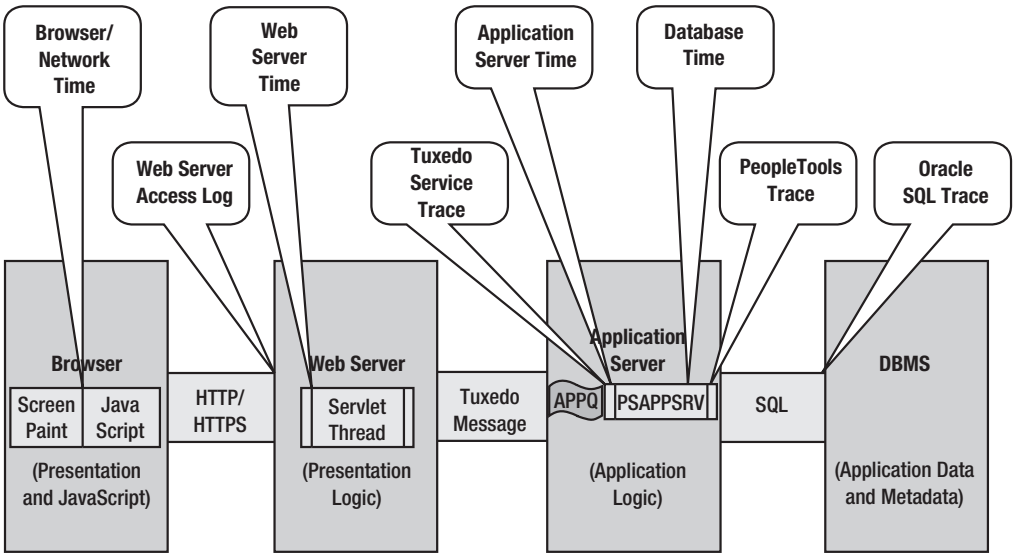


Figure 10-5. *Ping measurements on the PIA infrastructure*

PeopleSoft has added the instrumentation for Ping without major changes to the underlying technology. As the diagram in Figure 10-5 suggests, Ping does not measure exactly the same thing as the metrics described in Chapter 9. This is best illustrated by examining a single Ping result and comparing the Ping times with the times obtained from the other sources of metrics. These times are collected by a JavaScript script that has been built into the component, and they are then stored in the database in a table called PS_PTP_TST_CASES (see Listing 10-6).

On Oracle, the timestamp columns are of type DATE, so they are accurate only to 1 second. It is necessary to use the time columns for more precise timings.

Listing 10-6. *PSPing data stored in the database*

```
SELECT * FROM ps_ptp_tst_cases
WHERE ptp_test_case_id = 'GO-FASTER-3 2004.05.22'
AND ptp_dttm_a1 = TO_DATE('20040522 230448','YYYYMMDD HH24MISS');

PTP_TEST_CASE_ID               PTP_DTTM_A1         PTP_DTTM_A2
------------------------------ ------------------- -------------------
PTP_DTTM_B1         PTP_DTTM_B2         PTP_DTTM_C1         PTP_DTTM_C2
------------------- ------------------- ------------------- -------------------
PTP_TIME_D PTP_AS_TIME PTP_TOTAL_TIME PTP_WS_TIME
---------- ----------- -------------- -----------
GO-FASTER-3 2004.05.22         23:04:48 22/05/2004 23:04:49 22/05/2004
23:04:48 22/05/2004 23:04:49 22/05/2004 23:04:48 22/05/2004 23:04:49 22/05/2004
       .21         .64           .852        .722
```

Database Time

The Ping transaction is defined in the DERIVED_PTP.HTMLCTLEVENT.FieldChange PeopleCode (see Listing 10-7). Queries on the records PTP_TABLE1 and PTP_TABLE2 are run and timed in the PeopleCode. The PeopleCode system variable %PerfTime returns the time on the application server, rather than the database server, and is used to calculate the duration of the FILL and SELECT operations.

Listing 10-7. *Extract from* DERIVED_PTP.HTMLCTLEVENT.FieldChange *PeopleCode*

```
REM USE ROWSET TO READ RESULTS FROM A JOIN WITH BIND VARIABLE

&nbr = 10001;
&Table1_vw_rs = CreateRowset(Record.PTP_TABLE1_VW);

&startTime = %PerfTime;
&Table1_vw_rs.Fill("WHERE PTP_SEQ_NBR >= :1", &nbr);
&timeTaken = &timeTaken + (%PerfTime - &startTime);

&Rs = GetRowset(Scroll.PTP_TABLE1);
&Rs.Flush();
&startTime = %PerfTime;
&Rs.Select(Record.PTP_TABLE1, "WHERE PTP_SEQ_NBR <= 10010");
&timeTaken = &timeTaken + (%PerfTime - &startTime);

DERIVED_PTP.PTP_TIME_D = &timeTaken;
```

The resultant SQL can be seen in the PeopleTools trace (see Listing 10-8). The queries return 10 and 100 rows, respectively.

Listing 10-8. *PSPing SQL*

```
PSAPPSRV.2564   1-22899  23.04.49    0.190 Cur#1.2564.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
PTP_SEQ_NBR, PTP_SEQ_CHAR, DESCR, PTP_INT01, PTP_INT02, PTP_INT03, PTP_INT04, PTP_INT05,
PTP_INT06, PTP_INT07, PTP_INT08, PTP_INT09, PTP_INT10, PTP_INT11, PTP_INT12, PTP_INT13,
PTP_INT14, PTP_INT15, PTP_INT16, PTP_INT17, PTP_INT18, PTP_INT19, PTP_INT20, PTP_INT21,
PTP_INT22, PTP_INT23, PTP_INT24, PTP_CHAR01, PTP_CHAR02, PTP_CHAR03, PTP_CHAR04,
PTP_CHAR05, PTP_CHAR06, PTP_CHAR07, PTP_CHAR08, PTP_CHAR09, PTP_CHAR10, PTP_CHAR11,
PTP_CHAR12, PTP_CHAR13, PTP_CHAR14, PTP_CHAR15, PTP_CHAR16, PTP_CHAR17, PTP_CHAR18,
PTP_CHAR19, PTP_CHAR20, PTP_CHAR21, PTP_CHAR22, PTP_CHAR23 FROM PS_PTP_TABLE1 WHERE
PTP_SEQ_NBR <= 10010 ORDER BY PTP_SEQ_NBR
PSAPPSRV.2564   1-22900  23.04.49    0.010 Cur#1.2564.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
PTP_SEQ_NBR, PTP_LINE, PTP_INT01, PTP_INT02, PTP_INT03, PTP_INT04, PTP_INT05, PTP_INT06,
PTP_INT07, PTP_INT08, PTP_INT09, PTP_INT10, PTP_INT11, PTP_INT12, PTP_INT13, PTP_INT14,
PTP_INT15, PTP_INT16, PTP_INT17, PTP_INT18, PTP_INT19, PTP_INT20, PTP_INT21, PTP_INT22,
PTP_INT23, PTP_INT24, PTP_CHAR01, PTP_CHAR02, PTP_CHAR03, PTP_CHAR04, PTP_CHAR05,
PTP_CHAR06, PTP_CHAR07, PTP_CHAR08, PTP_CHAR09, PTP_CHAR10, PTP_CHAR11, PTP_CHAR12,
PTP_CHAR13, PTP_CHAR14, PTP_CHAR15, PTP_CHAR16, PTP_CHAR17, PTP_CHAR18, PTP_CHAR19,
PTP_CHAR20, PTP_CHAR21, PTP_CHAR22, PTP_CHAR23, PTP_CHAR24 FROM PS_PTP_TABLE2 WHERE
EXISTS  (SELECT 'X' FROM PS_PTP_TABLE1  WHERE PTP_SEQ_NBR <= 10010  AND
PS_PTP_TABLE2.PTP_SEQ_NBR = PS_PTP_TABLE1.PTP_SEQ_NBR )  ORDER BY PTP_SEQ_NBR, PTP_LINE
```

There is already a discrepancy in the timings. The Ping reports the database time as 0.21s (see Listing 10-6), while the trace claims 0.20s (see Listing 10-8) for these two queries. Both of these times include any time spent in the SQL*Net layers and any network layers between the application and database servers.

PeopleSoft Ping is not a good test of database performance, because the same rows are repeatedly read by successive Pings, so they are likely to be cached by the database. Therefore, there is unlikely to be any physical I/O for these queries, and their performance is unlikely to be affected by the disk subsystem. If, however, the database server becomes CPU bound, I would expect an increase in database time.

Application Server Time

When a Ping occurs, an ICPanel service runs on a PSAPPSRV server process. The standard transaction copies the data queried into one row set to another (see Listing 10-9). This kind of operation is quite typical of PeopleSoft 8 applications, and so is a reasonable test transaction. This operation does not require any database activity, so it is CPU intensive for the application server process.

Listing 10-9. *Extract from* `DERIVED_PTP.HTMLCTLEVENT.FieldChange` *PeopleCode showing a rowset copy in PeopleSoft Ping*

```
REM  Copy using Rowset function from one RowSet to Another
&Table1_rs.CopyTo(&Table1_cpy_rs);
```

The duration of the service is recorded in the Tuxedo Service trace (see Listing 10-10).

Listing 10-10. *Extract from* `APPQ.stderr`

```
SERVICE    PID        SDATE        STIME        EDATE        ETIME
 -------    ---        -----        -----        -----        -----
@ICPanel    3096       1085263488   77964316     1085263489   77964987
```

The Tuxedo service duration is the difference between the ETIME and STIME,[4] which in this case is 0.671 seconds, whereas Ping reports the application server time as only 0.64 seconds.

The Tuxedo service trace reports the time at which the service starts and completes. This timing information is captured within the Tuxedo libraries that handle the incoming message before unpacking the message and passing it to PeopleSoft code.[5] The PeopleSoft instrumentation must be called from within the PeopleSoft own code, rather than any Tuxedo code. Hence the PeopleSoft timing must be less than the Tuxedo service time.

The discrepancy in the times must include the time taken to unpack the incoming service message and pass the data to the PeopleSoft functions, and to package the return message before placing it on the return queue. It also implies that the application server time in Ping does not include any time that the request spends on the queue waiting to get onto a server process, because the Tuxedo service time does not either.

4. In this example, the application server was on Windows, so the times are in milliseconds.
5. It is not clear from the Tuxedo documentation whether the service time includes the time taken to dequeue the incoming request and enqueue the outgoing request.

Web Server Time

The web server response time is reported in the access log (see Listing 10-11) as 0.742 seconds, whereas Ping reports 0.722 seconds.

Listing 10-11. *Extract of* `PIA_access.log`

```
2004-05-22  23:04:49  0.742  125316  10.0.0.8  go-faster-3  POST  200
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
/psc/ps/EMPLOYEE/HRMS/c/UTILITIES.PTPERF_TEST.GBL
2004-05-22  23:04:49  0.01  1217  10.0.0.8  go-faster-3  GET  200  "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)"  /cs/ps/cache/PTPFUNCLIB_1.js  -
2004-05-22  23:04:49  0.0  77  10.0.0.8  go-faster-3  GET  200  "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)"  /psc/ps/psping_result.gif
tot=0.852&brt=0.13&wbt=0.082&apt=0.43&dbt=0.210&PpmStateNum=48
```

The results from the previous Ping are encoded as a query string on the `psping_result.gif`. This is so that the JavaScript script can make the colors change when the values go over the thresholds that can be defined.

Any time that a Tuxedo service request spends on a queue in the application server is not part of the time on the application server process, and so it will be included in the web server time.

Browser Time

The browser time is calculated from JavaScript that executes on the client browser itself. It is not clear whether this fully takes account of the time taken by the browser to "paint" the screen. Nonetheless, this is certainly a much closer approximation to the end user's actual experience than the access log from either the web server or any proxy server that might be in use.

The browser time is the difference between the browser and web server total response times. It includes the network transmission time between the web server and the browser. It must also include the discrepancy between the web server time as reported by PeopleSoft Ping and the servlet response time as reported in the web server access log.

PeopleSoft Ping Case Study 1: Desktop/Browser Performance

One of the most valuable uses of Ping is to provide an indication of browser and network performance, without the need to introduce third-party measurement software at additional expense.

The graph in Figure 10-6 shows the performance of a 384MHz PC running Microsoft Internet Explorer. It had 64MB of physical memory, but the Task Manager initially indicated that the commit change (total memory usage) is 125MB. Therefore, Windows is likely to be paging memory to and from the page file on the local disk.

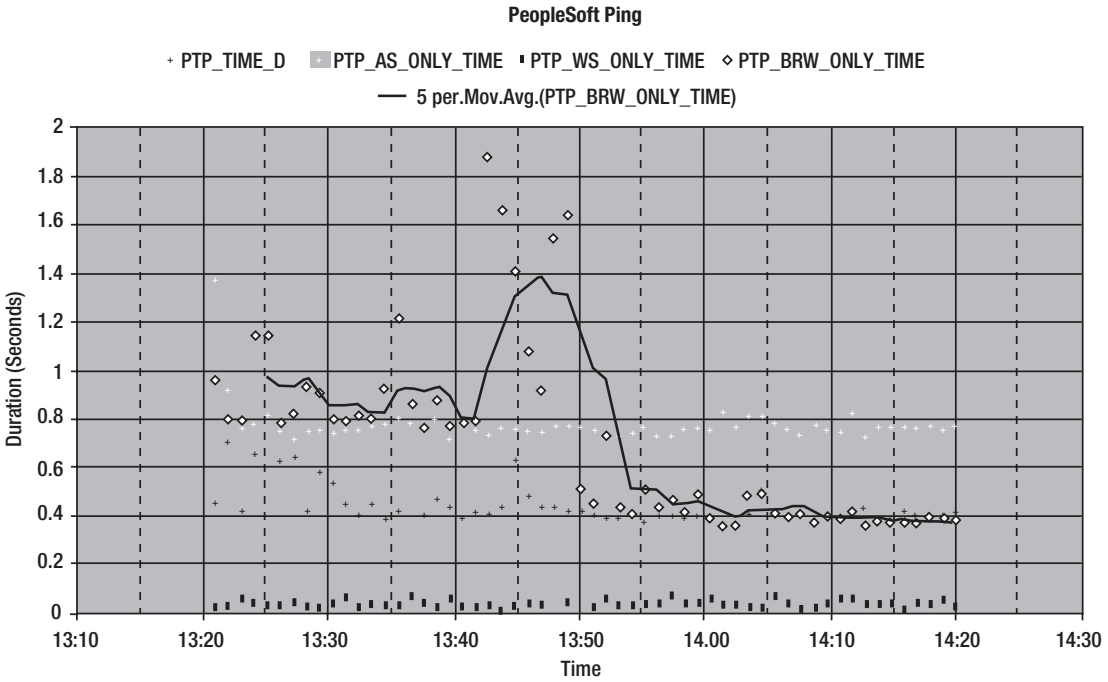


Figure 10-6. *Browser-only Ping time*

I shut down all other windows and as many services as possible. The act of starting the services window and shutting these processes down caused Windows to page memory, so there was an increase in browser time. I reduced the commit charge to 89MB, and the browser time fell from 0.8 seconds to 0.4 seconds.

From this, we can reach the following conclusions:

- Paging memory from disk is a slow business and generally degrades Windows performance, not just the browser. Make sure that the desktop has enough physical memory.
- PeopleSoft will run in a browser on any PC, but the specification of the PC can significantly affect the user experience.

PeopleSoft Ping Case Study 2: Application Server CPU Speed

Figure 10-7 shows Ping times from two different application servers connected to the same database.

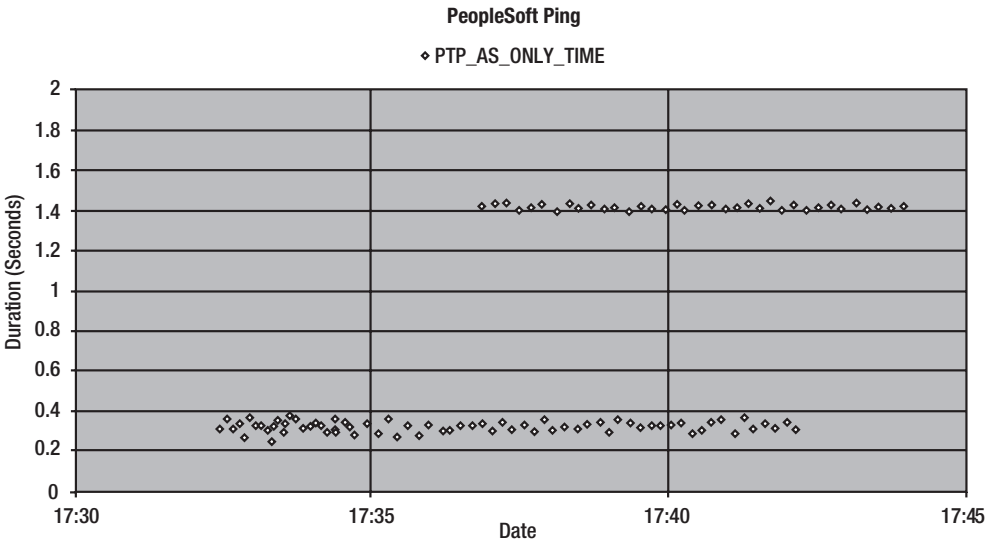


Figure 10-7. *Application server–only Ping time*

One application server is running on a Unix server with 750MHz processors and reports an average application server Ping time of 1.43 seconds. The other is on a Windows server with 3.2GHz processor and reports an average Ping time of 0.33 seconds. The processors on the Windows server are slightly more than four times faster, and the Ping is slightly more than four times faster.

The Ping transaction for the application server time is CPU intensive. It does not include any database activity, so the application server is simply executing PeopleSoft's C code.

This test suggests that processor speed is important to the performance of the PIA and PeopleCode code (excluding database access) in the application server. Maybe a clock cycle is a clock cycle is a clock cycle!

Conclusion

PeopleSoft Ping is not a fundamental indicator of system performance, but it will indicate the presence of contention in the various PIA components. Since all PeopleSoft systems execute the same Ping transaction, it is also a useful standard measure for comparison, either of different desktops within an organization or of different PeopleSoft systems.

Performance Monitor

PeopleSoft Performance Monitor, introduced in PeopleTools 8.44, represents a huge effort by PeopleSoft to fully instrument the application and produce a "wait interface." To put it in terms that an Oracle DBA would recognize, it provides for the application what an Oracle SQL trace with the wait interface (event 10046, level 8) does for the database.

At the time of this writing, I am working with PeopleTools 8.44, and I have yet to use Performance Monitor in a real production environment, so the comments in this section are based

upon some limited experiments that I have performed. One of the most significant unknowns remains what the overhead (measurement intrusion effect) is of using this facility on an active production system.

In this section I provide only a brief overview of the utility. The product documentation in *PeopleTools 8.44 PeopleBook: PeopleSoft Performance Monitor* is very detailed and well worth reading.

Architectural Overview

PeopleSoft has separated the job of obtaining the metrics from the various components being monitored from the job of storing those metrics in a database. So the overhead—especially the database processing and space overhead—of inserting and manipulating the performance metrics data can be separated from the production database. Thus if a company runs more than one PeopleSoft system that it wishes to monitor, it could build another, perhaps PeopleTools-only, system just to hold the metrics (although this may have a licensing implication).

So there is a concept of both a *monitoring system* and a *monitored system* (see Figure 10-8). A system can be configured to self-monitor, but there are various warnings against doing so on a production system due to the associated overhead. However, self-monitoring cannot lead to a recursive loop.

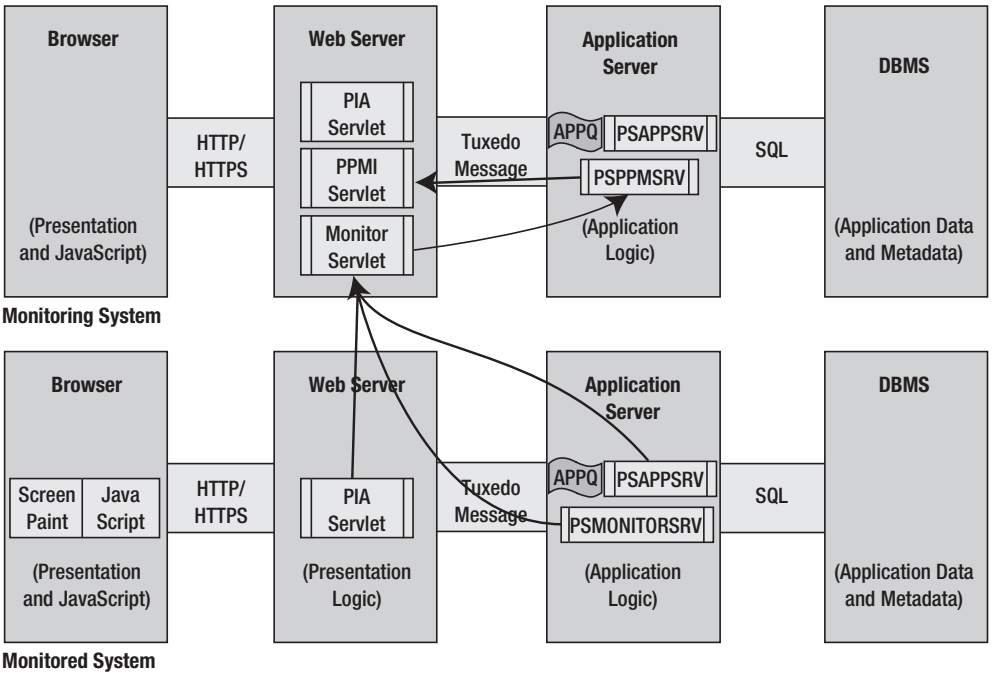


Figure 10-8. *PeopleSoft performance monitoring architecture*

To obtain the metrics from the monitored system, PeopleSoft has added code to the PIA servlet, and to the various servers in the application server and Process Scheduler. The monitor server (PSMONITORSRV) was added in PeopleTools 8.44 to collect host resource usage metrics. It also connects to the Tuxedo Bulletin Board to collect information about the Tuxedo domain (see Chapter 2 for a discussion of the Tuxedo architecture). In the Performance Monitor package, these various sources of metrics are referred to as *agents*. The agents send the metrics to the monitor servlet in the monitoring system. The `netstat` command reports a number of persistently established connections between the agents and the web server.

■ **Note** It is common for PeopleSoft systems to have multiple application servers and multiple web servers. Metrics must be collected from each application server and each web server.

On the monitoring system, the metrics are collected by the monitoring servlet and sent on to the performance collator server, PSPPMSRV, which then inserts the metrics into the database.

Metrics

The metrics collected by Performance Monitor break down into two major groups:

- **Transactions** consist of the activities that occur when the operator does something in the PIA that causes communication with the servlet, such as clicking a Save button in a component.
- **Events** are usually initiated periodically by the monitoring agents to collect metrics, such as CPU or Tuxedo metrics.

I discuss each group in further detail in the sections that follow.

Transactions

Transactions are the activities that occur in the web server and application server when a user initiates a conversation with the web server by clicking a link or button, or navigating between fields in the PIA.

A transaction can be composed of other transactions. For example, a PIA request can comprise one or more Jolt requests, the Jolt request(s) will give rise to a Tuxedo service, and so on. Each of these elements is a separate transaction. This is similar to an Oracle trace, where user SQL statements can give rise to recursive SQL statements, or a fetch operation can include sequential and/or scattered read operations. All of the transactions that occur in response to a single user action are collectively called a *Performance Monitoring Unit* (PMU). The transactions in a PMU exist in a hierarchy and can be displayed as a tree structure in the PIA. Figure 10-9 shows the PMU for a single PeopleSoft Ping. The level of detail in a PMU is controlled by the filtering level, which is discussed later in this chapter.

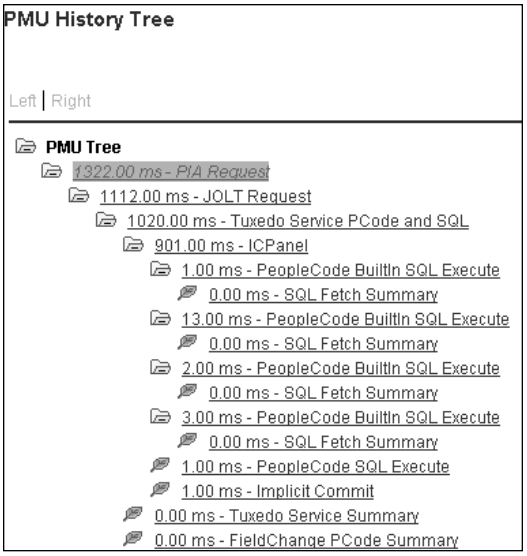


Figure 10-9. *PMU History Tree for a PeopleSoft Ping with verbose agent filtering*

Transactions are pieces of processing that take a certain amount of time to execute, so they all have a duration in addition to the various other metrics.

All of the output from Performance Monitor is stored in the database of the monitoring system.[6] Current or open PMUs are temporarily placed in the table PSPMTRANSCURR, and when they are completed they are moved to PSPMTRANSHIST. Old PMUs can be deleted or archived to PSPMTRANSARCH. These three tables have similar structures (see Table 10-2). It is worth taking some time to understand the structure of these tables, so that you can build your own queries.

Table 10-2. *Structure of PSPMTRANS% Tables*

Column Name	Description
PM_INSTANCE_ID	Instance. Uniquely identifies a transaction or event.
PM_INSTANCE_SEQ	Only on PSPMTRANSCURR.
PM_TRANS_DEFN_SET	See PM_TRANS_DEFT. Currently always 1.
PM_TRANS_DEFN_ID	With PM_TRANS_DEFN_SET, specifies transaction type defined on PSPMTRANSDEFN, which also holds attributes that describe the contexts and metrics for the transaction.
PM_AGENTID	Specifies agent (and hence database, application server, and web server) that collected transaction information. The value is looked up on PSPMAGENT.
OPRID	PeopleSoft operator that generated transaction.

(Continues)

6. An ERD diagram of the performance monitoring tables (PSPM%) can be downloaded from the Peoplesoft Connection website under Documentation Updates ➤ Enterprise ➤ PeopleTools ➤ Performance Monitor. There are also two use-case flowcharts for troubleshooting.

Column Name	Description
PM_PERF_TRACE	Name of the performance trace. Simply a string attribute set in the user's PIA session that identifies the session so that the transactions for that operator can be easily recovered. Defaults to `<OPRID>: YYYY-MM-DD HH24:MI:SS`.
PM_ACTION	Only on PSPMTRANSCURR.
PM_TRANS_DELETED	Only on PSPMTRANSCURR.
PM_CONTEXT_VALUE1	Identifies what component, PeopleCode, etc., generated the transaction. Not all transactions (e.g., Tools Controlled Commit) have contexts, and you may need to look at the context of the parent transaction to clarify from where the transaction originated.
PM_CONTEXT_VALUE2	Identifies what component, PeopleCode, etc., generated the transaction.
PM_CONTEXT_VALUE3	Identifies what component, PeopleCode, etc., generated the transaction.
PM_CONTEXTID_1	Copied from PSPMTRANSDEFN.PM_CONTEXTID_1. Identifier for PM_CONTEXT_VALUE1 that can be looked up on PSPMCONTEXTDEFN.
PM_CONTEXTID_2	Copied from PSPMTRANSDEFN.PM_CONTEXTID_2. Identifier for PM_CONTEXT_VALUE2 that can be looked up on PSPMCONTEXTDEFN.
PM_CONTEXTID_3	Copied from PSPMTRANSDEFN.PM_CONTEXTID_3. Identifier for PM_CONTEXT_VALUE3 that can be looked up on PSPMCONTEXTDEFN.
PM_AGENT_STRT_DTTM	Timestamp when agent first updated the transaction.
PM_AGENT_LAST_DTTM	Timestamp when agent last updated the transaction. Only on PSPMTRANSCURR.
PM_MON_STRT_DTTM	Timestamp when monitor first received the transaction.
PM_MON_LAST_DTTM	Timestamp when monitor last received the transaction. Only on PSPMTRANSCURR.
PM_TIMEOUT_DTTM	Timestamp when the transaction timed out. Only on PSPMTRANSCURR.
PM_TRANS_DURATION	Duration of transaction (milliseconds). Not on PSPMTRANSCURR; only on PSPMTRANSHIST and PSPMTRANSARCH.
PM_PARENT_INST_ID	Transaction instance of parent transaction. The transaction that spawned this transaction. Zero if this is the top (originating) transaction.
PM_TOP_INST_ID	Transaction instance that originated the series of transactions.
PM_PROCESS_ID	Operating system process ID of originating server ID or agent.
PM_METRIC_VALUE1	Transaction metric 1. Metric type defined by PSPMTRANSDEFN. PM_METRICID_1. Description from PSPMMETRICDEFN.PM_ METRICLABEL1.
PM_METRIC_VALUE2	Transaction metric 2.
PM_METRIC_VALUE3	Transaction metric 3.
PM_METRIC_VALUE4	Transaction metric 4.
PM_METRIC_VALUE5	Transaction metric 5.
PM_METRIC_VALUE6	Transaction metric 6.
PM_METRIC_VALUE7	Transaction metric 7 (a character string).
PM_ADDTNL_DESCR	Can contain extra information, such as a SQL statement and bind variable values.

Transactions can have up to three contexts, which help to identify the action that generated the transaction. The PM_CONTEXTIDs are stored on the transaction, and the contexts are defined on the table PSPMCONTEXTDEFN.

Each transaction can hold up to seven metrics, one of which can be a character string. The PM_METRICID can be looked up on PSPMMETRICDEFN. Each metric definition has a metric type (PM_METRICTYPE):

- **1: Counter**. For example, metric 4 is "Total servlet request time (ms)". The metric accumulates the time and reports it each time transaction 152 occurs. In order to find the servlet time accumulated since the last transaction 152 occurred, you need to subtract the value of the metric on the last transaction record from the current value.
- **2: Gauge**. For example, metric 102 is "%CPU Used". This type of metric is a scalar quantity. That is to say it contains a measurement of a particular quantity. The value can be understood in isolation, as opposed to a counter that must be compared.
- **3: Numeric identifier**. For example, metric 20 is the HTTP response code. This is a number that has a meaning, but you don't do any arithmetic with it. However, there are some type 3 metrics that should really be type 2, such as metric 37, "Number of Tuxedo Servers".
- **4: String identifier**. For example, metric 27 is a file name.

The interrelationship of the various tables that hold and describe transactions is shown in Figure 10-10. If you are collecting information for multiple monitored systems on the one monitoring system, then the database name is related to the agent.

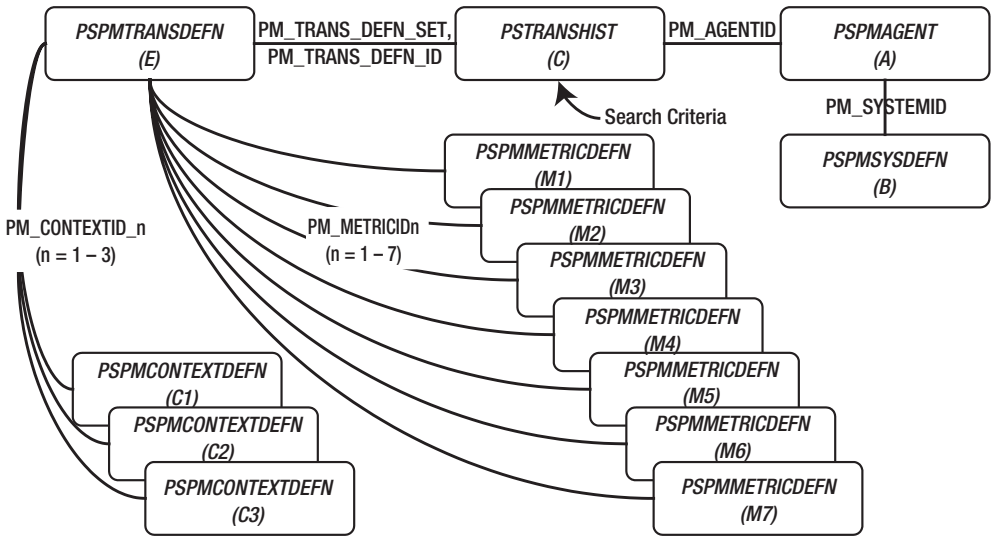


Figure 10-10. *Transactions structure*

Listing 10-12 shows an example of how to extract that data for a specific transaction. PeopleSoft also delivers a few similar ad hoc queries that interrogate the performance metrics directly.

Listing 10-12. `transaction_metrics.sql`*: SQL query to report on transactions in a PMU*

```
SELECT C.PM_TOP_INST_ID, C.PM_INSTANCE_ID, C.PM_PARENT_INST_ID
,      B.DBNAME
,      A.PM_HOST_PORT, A.PM_DOMAIN_NAME, A.PM_AGENT_TYPE, A.PM_INSTANCE
,      C.PM_AGENT_STRT_DTTM, C.PM_MON_STRT_DTTM
,      C.OPRID, C.PM_PERF_TRACE, C.PM_PROCESS_ID
,      E.PM_TRANS_DEFN_ID, E.DESCR60
,      'Context1:'||c.pm_contextid_1||'-'||C1.PM_CONTEXT_LABEL
                                       ||'='||C.PM_CONTEXT_VALUE1
,      'Context2:'||c.pm_contextid_2||'-'||C2.PM_CONTEXT_LABEL
                                       ||'='||C.PM_CONTEXT_VALUE2
,      'Context3:'||c.pm_contextid_3||'-'||C3.PM_CONTEXT_LABEL
                                       ||'='||C.PM_CONTEXT_VALUE3
,      C.PM_TRANS_DURATION
,      'Metric1:'||M1.PM_METRICLABEL||'='||C.PM_METRIC_VALUE1
,      'Metric2:'||M2.PM_METRICLABEL||'='||C.PM_METRIC_VALUE2
,      'Metric3:'||M3.PM_METRICLABEL||'='||C.PM_METRIC_VALUE3
,      'Metric4:'||M4.PM_METRICLABEL||'='||C.PM_METRIC_VALUE4
,      'Metric5:'||M5.PM_METRICLABEL||'='||C.PM_METRIC_VALUE5
,      'Metric6:'||M6.PM_METRICLABEL||'='||C.PM_METRIC_VALUE6
,      'Metric7:'||M7.PM_METRICLABEL||'='||C.PM_METRIC_VALUE7
,      C.PM_ADDTNL_DESCR
FROM   PSPMAGENT A, PSPMSYSDEFN B, PSPMTRANSHIST C, PSPMTRANSDEFN E,
       PSPMMETRICDEFN M1, PSPMMETRICDEFN M2, PSPMMETRICDEFN M3,
       PSPMMETRICDEFN M4, PSPMMETRICDEFN M5, PSPMMETRICDEFN M6,
       PSPMMETRICDEFN M7,
       PSPMCONTEXTDEFN C1, PSPMCONTEXTDEFN C2, PSPMCONTEXTDEFN C3
WHERE B.PM_SYSTEMID = A.PM_SYSTEMID
AND   A.PM_AGENTID = C.PM_AGENTID
AND   C.PM_TOP_INST_ID = 824633721163 /*specify PMU and all child transactions*/
AND   C.PM_TRANS_DEFN_SET = E.PM_TRANS_DEFN_SET
AND   C.PM_TRANS_DEFN_ID = E.PM_TRANS_DEFN_ID
AND   M1.PM_METRICID(+) = E.PM_METRICID_1
AND   M2.PM_METRICID(+) = E.PM_METRICID_2
AND   M3.PM_METRICID(+) = E.PM_METRICID_3
AND   M4.PM_METRICID(+) = E.PM_METRICID_4
AND   M5.PM_METRICID(+) = E.PM_METRICID_5
AND   M6.PM_METRICID(+) = E.PM_METRICID_6
AND   M7.PM_METRICID(+) = E.PM_METRICID_7
AND   C1.PM_CONTEXTID(+) = C.PM_CONTEXTID_1
AND   C2.PM_CONTEXTID(+) = C.PM_CONTEXTID_2
AND   C3.PM_CONTEXTID(+) = C.PM_CONTEXTID_3
;
```

Listing 10-13 shows the output from Listing 10-12 for the root transaction shown in Figure 10-9.

Listing 10-13. *Sample output from* `transaction_metrics.sql`

```
PM_TOP_INST_ID PM_INSTANCE_ID PM_PARENT_INST_ID DBNAME
PM_HOST_PORT
PM_DOMAIN_NAME                   PM_AGENT_TYPE
PM_INSTANCE                      PM_AGENT_STRT_DTTM  PM_MON_STRT_DTTM
OPRID                           PM_PERF_TRACE                  PM_PROCESS_ID
PM_TRANS_DEFN_ID DESCR60
'CONTEXT1:'||C.PM_CONTEXTID_1||'-'||C1.PM_CONTEXT_LABEL||'='||C.PM_CONTEXT_VALUE
'CONTEXT2:'||C.PM_CONTEXTID_2||'-'||C2.PM_CONTEXT_LABEL||'='||C.PM_CONTEXT_VALUE
'CONTEXT3:'||C.PM_CONTEXTID_3||'-'||C3.PM_CONTEXT_LABEL||'='||C.PM_CONTEXT_VALUE
PM_TRANS_DURATION
'METRIC1:'||M1.PM_METRICLABEL||'='||C.PM_METRIC_VALUE1
'METRIC2:'||M2.PM_METRICLABEL||'='||C.PM_METRIC_VALUE2
'METRIC3:'||M3.PM_METRICLABEL||'='||C.PM_METRIC_VALUE3
'METRIC4:'||M4.PM_METRICLABEL||'='||C.PM_METRIC_VALUE4
'METRIC5:'||M5.PM_METRICLABEL||'='||C.PM_METRIC_VALUE5
'METRIC6:'||M6.PM_METRICLABEL||'='||C.PM_METRIC_VALUE6
'METRIC7:'||M7.PM_METRICLABEL||'='||C.PM_METRIC_VALUE7
PM_ADDTNL_DESCR
--------------------------------------------------------------------------------
  824633721163   824633721163                0 HR88
go-faster-3:7201:7202
ps                               WEBSERVER
-1                               16:12:07 14.06.2004 16:12:09 14.06.2004
PS                              PS: 2004-06-14 16:01:11                    0
             101 Reported at entry and exit of PIA servlet
Context1:3-Session ID=AN7tpzSwpZc4kt9k8QNaCcYUWWh9FaFt!1963244185!1087224685145
Context2:2-IP Address=10.0.0.3
Context3:1-Action=View Page
             1322
Metric1:Response Size (bytes)=17613
Metric2:Response Code=200
Metric3:Static Content Count=0
Metric4:Is this a Pagelet?=0
Metric5:=0
Metric6:=0
Metric7:=
http://go-faster-3:7201/psc/ps/EMPLOYEE/HRMS/c/UTILITIES.PTPERF_TEST.GBL
```

In Listing 10-13, transaction 101 is "Reported at entry and exit of PIA servlet". It is the root transaction of the PMU shown in Figure 10-9. It has three contexts:

- Context 1: Action=View Page
- Context 2: IP Address=10.0.0.3
- Context 3: Session ID=AN7tpzSwpZc4kt9k8 ...

However, the additional description is often more useful. In this case, it is the URL of the component:

```
http://go-faster-3:7201/psc/ps/EMPLOYEE/HRMS/c/UTILITIES.PTPERF_TEST.GBL
```

Transaction 101 has four metrics:

- **Metric 19**: "Response Size (bytes)"
- **Metric 20**: "Response Code"
- **Metric 22**: "Static Content Count"
- **Metric 23**: "Is this a Pagelet?"

Events

Certain *events* are defined that also cause the monitor agents to collect various related metrics. These events can occur

- On a regular cycle, such as host resource or Tuxedo server metrics
- In response to a user action, such as a PeopleSoft Ping
- On an exception, such as a Jolt exception or query timeout

Events do not have contexts but are instead related to an agent, either a web server JVM, or an application server or Process Scheduler domain server process. They are stored in the database in PSPMEVENTHIST (described in Table 10-3) and are archived to PSPMEVENTARCH.

Table 10-3. *Structure of the PSPMEVENTHIST and PSPMEVENTARCH Tables*

Column Name	Description
PM_INSTANCE_ID	Instance. Uniquely identifies a transaction or event.
PM_EVENT_DEFN_SET	See PM_EVENT_DEFN_ID. Currently always 1.
PM_EVENT_DEFN_ID	With PM_EVENT_DEFN_SET, specifies event type defined on PSPMEVENTDEFN, which also holds attributes that describe the metrics for the event.
PM_AGENTID	Specifies agent (and hence database, application server, and web server) that collected transaction information. The value is looked up on PSPMAGENT.
PM_AGENT_DTTM	Timestamp of event at the agent.
PM_MON_DTTM	Timestamp of event when received by the monitor process.
PM_PROCESS_ID	Operating system process ID of the originating server ID or agent.
PM_FILTER_LEVEL	Filter level of the agent (see the next section, "Agent Filter Levels").
PM_METRIC_VALUE1	Transaction metric 1. Metric type defined by PSPMEVENTDEFN. PM_METRICID_1. Description from PSPMMETRICDEFN.PM_ METRICLABEL1.
PM_METRIC_VALUE2	Event metric 2.
PM_METRIC_VALUE3	Event metric 3.
PM_METRIC_VALUE4	Event metric 4.
PM_METRIC_VALUE5	Event metric 5.

Column Name	Description
PM_METRIC_VALUE6	Event metric 6.
PM_METRIC_VALUE7	Event metric 7 (a character string).
PM_ADDTNL_DESCR	Can contain extra information, such as a SQL statement and bind variable values.

The structure for events (see Figure 10-11) is less complicated than transactions because there are no contexts.

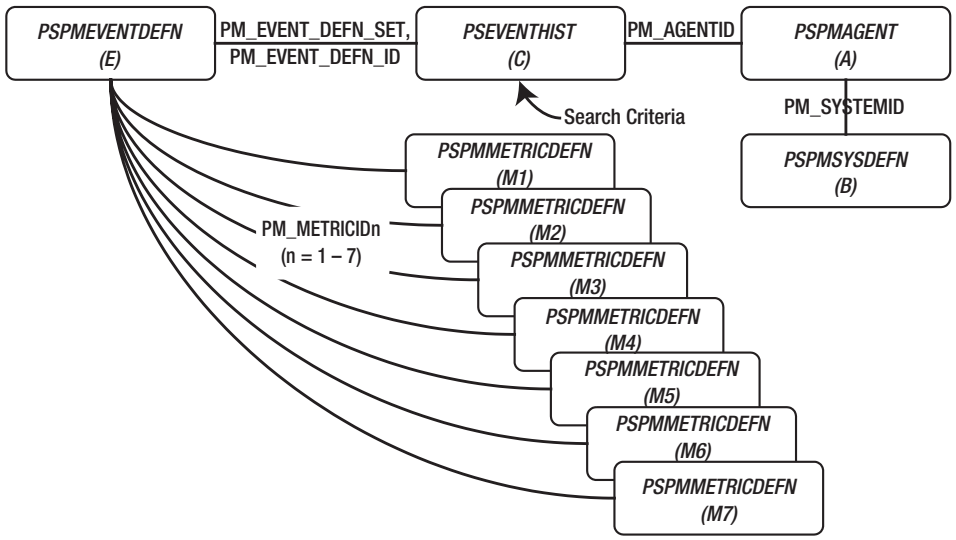


Figure 10-11. *Events structure*

Listing 10-14 shows a simple query to retrieve the events and metrics that occurred in a particular time window.

Listing 10-14. *Extract from* `event_metrics.sql`

```
SELECT B.DBNAME,
       A.PM_HOST_PORT, A.PM_AGENT_TYPE, A.PM_DOMAIN_NAME, A.PM_INSTANCE,
       C.PM_AGENT_DTTM, C.PM_INSTANCE_ID,
       E.PM_EVENT_DEFN_ID, E.DESCR60,
       'Metric1:'||M1.PM_METRICLABEL||'='||C.PM_METRIC_VALUE1,
       'Metric2:'||M2.PM_METRICLABEL||'='||C.PM_METRIC_VALUE2,
       'Metric3:'||M3.PM_METRICLABEL||'='||C.PM_METRIC_VALUE3,
       'Metric4:'||M4.PM_METRICLABEL||'='||C.PM_METRIC_VALUE4,
       'Metric5:'||M5.PM_METRICLABEL||'='||C.PM_METRIC_VALUE5,
       'Metric6:'||M6.PM_METRICLABEL||'='||C.PM_METRIC_VALUE6,
       'Metric7:'||M7.PM_METRICLABEL||'='||C.PM_METRIC_VALUE7,
       C.PM_ADDTNL_DESCR
```

```
FROM   PSPMAGENT A, PSPMSYSDEFN B, PSPMEVENTHIST C, PSPMEVENTDEFN E,
       PSPMMETRICDEFN M1, PSPMMETRICDEFN M2, PSPMMETRICDEFN M3,
       PSPMMETRICDEFN M4, PSPMMETRICDEFN M5, PSPMMETRICDEFN M6,
       PSPMMETRICDEFN M7
WHERE  B.PM_SYSTEMID = A.PM_SYSTEMID
AND    A.PM_AGENTID = C.PM_AGENTID
AND    B.DBNAME IN ('HR88' ,'hr88')
AND    C.PM_AGENT_DTTM
              BETWEEN TO_DATE('2004-06-14 16:12:06','YYYY-MM-DD HH24.MI.SS')
                  AND TO_DATE('2004-06-14 16:12:11','YYYY-MM-DD HH24.MI.SS')
AND    C.PM_EVENT_DEFN_SET = E.PM_EVENT_DEFN_SET
AND    C.PM_EVENT_DEFN_ID = E.PM_EVENT_DEFN_ID
AND    M1.PM_METRICID(+) = E.PM_METRICID_1
AND    M2.PM_METRICID(+) = E.PM_METRICID_2
AND    M3.PM_METRICID(+) = E.PM_METRICID_3
AND    M4.PM_METRICID(+) = E.PM_METRICID_4
AND    M5.PM_METRICID(+) = E.PM_METRICID_5
AND    M6.PM_METRICID(+) = E.PM_METRICID_6
AND    M7.PM_METRICID(+) = E.PM_METRICID_7
;
```

Listing 10-15 shows a sample of the output from the query in Listing 10-14, the event generated by a PSPing.

Listing 10-15. *Sample output from event metrics*

```
DBNAME    PM_HOST_PORT
PM_AGENT_TYPE                    PM_DOMAIN_NAME
PM_INSTANCE                      PM_AGENT_DTTM       PM_INSTANCE_ID
PM_EVENT_DEFN_ID DESCR60
'METRIC1:'||M1.PM_METRICLABEL||'='||C.PM_METRIC_VALUE1
'METRIC2:'||M2.PM_METRICLABEL||'='||C.PM_METRIC_VALUE2
'METRIC3:'||M3.PM_METRICLABEL||'='||C.PM_METRIC_VALUE3
'METRIC4:'||M4.PM_METRICLABEL||'='||C.PM_METRIC_VALUE4
'METRIC5:'||M5.PM_METRICLABEL||'='||C.PM_METRIC_VALUE5
'METRIC6:'||M6.PM_METRICLABEL||'='||C.PM_METRIC_VALUE6
'METRIC7:'||M7.PM_METRICLABEL||'='||C.PM_METRIC_VALUE7
PM_ADDTNL_DESCR
--------------------------------------------------------------------------------
HR88     go-faster-3:7201:7202
WEBSERVER                        ps
-1                               16:12:08 14.06.2004   824633721166
             600 PSPING metrics forwarded from browser
Metric1:Network Latency (ms)=435
Metric2:WebServer Latency (ms)=100
Metric3:AppServer Latency (ms)=561
Metric4:DB Latency (millisecs)=451
```

```
Metric5:=0
Metric6:=0
Metric7:IP Address=10.0.0.3
PS;AN7tpzSwpZc4kt9k8QNaCcYUWWh9FaFt!1963244185!1087224685145
```

Metrics of type 1 are numeric counters that increment from event to event. For this type of metric, you will need to know the difference between the value of the metric for successive events. Oracle's analytic LEAD() function could be helpful. Listing 10-16 shows how you might code a query to do this.

Listing 10-16. *Extract from* `event_metrics2.sql` *showing the use of analytic functions for counters*

```
SELECT ...
       PM_METRICLABEL1,
       CASE
         WHEN PM_METRICTYPE1 = '1' AND PM_METRIC_VALUE1<PM_METRIC_LAST1
                                   THEN PM_METRIC_VALUE1
         WHEN PM_METRICTYPE1 = '1' THEN PM_METRIC_VALUE1-PM_METRIC_LAST1
         ELSE PM_METRIC_VALUE1
       END AS PM_METRIC_VALUE1,
...
FROM   (
SELECT ...
       M1.PM_METRICLABEL PM_METRICLABEL1 ,C.PM_METRIC_VALUE1,
       LEAD(C.PM_METRIC_VALUE1,1)
           OVER(PARTITION BY E.PM_EVENT_DEFN_SET, E.PM_EVENT_DEFN_ID,
                             C.PM_AGENTID, C.PM_METRIC_VALUE7
           ORDER BY C.PM_AGENT_DTTM DESC) AS PM_METRIC_LAST1,
       M1.PM_METRICTYPE PM_METRICTYPE1,
...
FROM   PSPMAGENT A, PSPMSYSDEFN B, PSPMEVENTHIST C, PSPMEVENTDEFN E,
...
```

Agent Filter Levels

The event and transaction definitions include a filter level (see Figure 10-12). The agents generate the metrics only if the filter level of the event is less than or equal to the filter level for the agent. The default agent filter level is 4 for all agents.

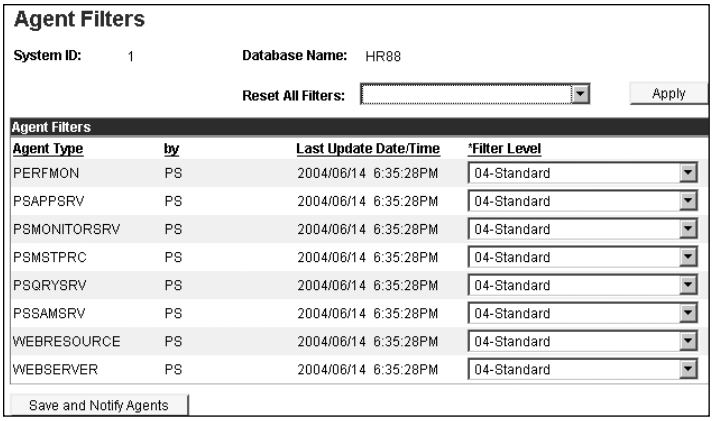


Figure 10-12. *Agent Filters panel*

If the Ping PMU shown in Figure 10-9 had been monitored with agents only at the standard filter level, it would not have shown any detail below the ICPanel service. By setting the filter level to 5 (verbose), the SQL statement and fetch operations were reported.

However, there are warnings in both the Agent Filters component and the documentation that increasing the filter level to 5 (verbose) or 6 (debug) can have a significant performance impact.

Performance Trace

The *performance trace* is essentially a way of grouping an operator's PMUs. So an operator can enable a trace session and perform a series of actions, giving rise to a series of PMUs. When the performance trace is enabled, a default name composed of the operator's ID and the current date and time is generated (see Figure 10-13), but this can be overridden. The performance trace name is also written to those PMU records. Later, just the transactions for that operator's session can be retrieved and analyzed. Thus you can find out where and how much time that operator spent waiting for the system to respond.

If the operator overrides the filter settings for their trace, event 601 is also generated.

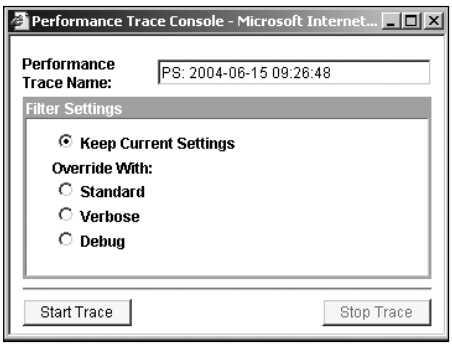


Figure 10-13. *Performance Trace Console dialog*

Transaction and Events

Performance Monitor collects and stores some of the same (or at least similar) metrics that are discussed in Chapter 9, but without the need to process trace files. All the metrics are described in the Performance Monitor PeopleBook, but in this section I want to look at some of those transactions and events, and suggest how they could be used.

All the events described in this section (except 600 and 601) are collected on a regular cycle (by default every 300 seconds).

Transaction ID 115: JOLT Request

The Jolt request transaction (see Table 10-4) shows the size of each incoming and outbound Tuxedo message. It provides an easier way to determine how many Tuxedo messages exceed the IPC message and queue size limits (see Chapter 13).

Table 10-4. *Jolt Request to the Application Server*

Metric	Description
Context 1	Session ID
Context 2	IP Address
Context 3	Tuxedo Service Name
Metric 1	Jolt Send Buffer (bytes)
Metric 2	Jolt Receive Buffer (bytes)
Metric 3	Jolt Return Code
Metric 4	Jolt Request Retried

The Error status code is stored in the additional information column, PM_ADDTNL_DESCR.

Individual transactions and events can also be examined in the PIA in the Performance Monitor ➤ History menu. All of the contexts and metrics can be viewed (see Figure 10-14).

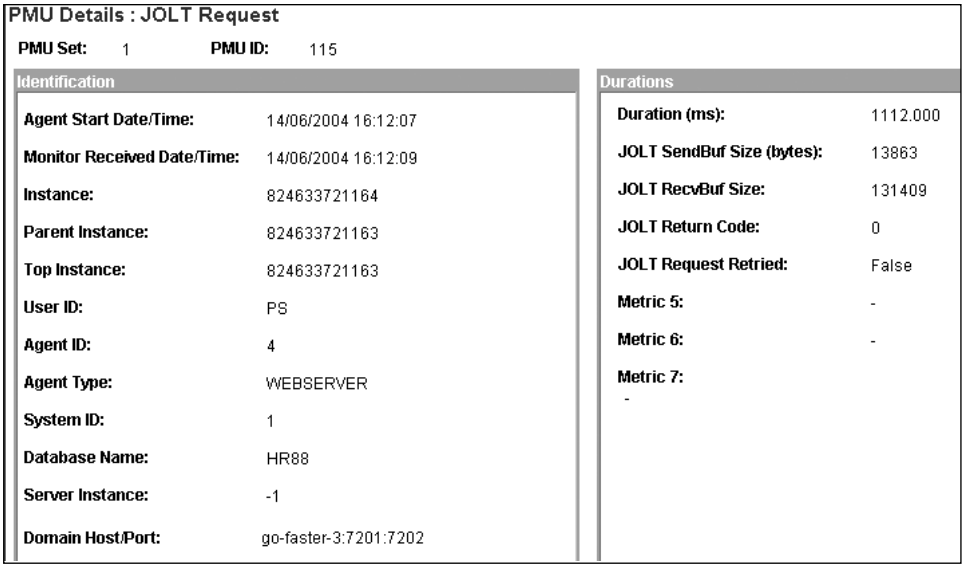


Figure 10-14. *Jolt request transaction as seen in Performance Monitor's History menu*

Previously, the message sizes could be seen only in the enhanced Tuxedo trace, but enabling this trace has a significant performance overhead. You can see that the message sizes reported in the transaction appear as parameters to the Tuxedo service calls in the enhanced Tuxedo log file, as shown in Listing 10-17.

Listing 10-17. *Enhanced Tuxedo log*

```
161207.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:  { tpservice({"ICPanel", 0x0,
0x2413040, 13863, 0, -1, {1087224688, 0, 37}})
161207.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    { tpalloc("CARRAY", "", 8192)
161207.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    } tpalloc = 0x24252e8
[tperrno TPEOS]
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    { tprealloc(0x24252e8, 131072)
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    } tprealloc = 0xfae90e0
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    { tprealloc(0xfae90e0, 196608)
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    } tprealloc = 0xfdca0e0
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:   { tpreturn(2, 0, 0xfdca0e0, 131409,
0x0)
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:    } tpreturn [long jump]
161208.GO-FASTER-3!PSAPPSRV.2264.2260.0: TRACE:at:  } tpservice
```

Event ID 150: JVM Status

The metrics reported by this event (see Table 10-5) are available in the web server console; however, there is no easy way to capture that information.

Table 10-5. *JVM Status*

Metric	Description	Comment
1	% of JVM Memory Used	
2	Max JVM Memory Available	
3	Sessions in Web Server/App Server	This is therefore the actual number of browser sessions concurrently connected to this web server.
4	Execute Threads	
5	Busy Threads	A JVM thread will be busy because it is either serving a request from a user itself or it is waiting for the application server to respond.
6	Domain Count	

The `printclient` command in the Tuxedo `tmadmin` utility will also report all the clients of the Bulletin Board and their status. However, it includes the Tuxedo handler processes and any instances of `tmadmin`, which this event does not. Hence, it is possible to use `printclient` to count the number of users connected to the PIA. This is not a perfect measure because the Tuxedo Jolt Listener and the PIA servlet have separate timeout settings. A session might timeout on the application server before it does so on the servlet. Therefore, the Performance Monitor metrics provide a better way to determine the number of concurrent users on a system.

Event ID 152: Web Site Status

This event reports metrics for the PIA servlet, as shown in Table 10-6.

Table 10-6. *Web Site Status Reported by WebServer Resource Monitor*

Metric	Description	Comment
1	Cumulative number of requests to all servlets	This metric corresponds to the number of entries found in the access log.
2	Servlet requests (last minute)	
3	Avg Request Time (last minute)	
4	Cumulative time in all Servlets (ms)	This metric can be slightly less than the sum of the time taken to serve the requests recorded in the web server access log.
5	Current Sessions	This does not match with the number web/application server session on event 150.
7	Site Path	A single web server may host several PeopleSoft systems. Each system will have a unique site name in the path to the servlet in the URL.

Event ID 153: Web Servlet Status

This event is similar to event 152, except that the statistics are broken down by servlet (see Table 10-7).

Table 10-7. *Web Servlet Status Reported by WebServer Resource Monitor*

Metric	Description	Comment
1	Cumulative request to this Servlet	This metric corresponds to the number of entries found in the access log.
2	Servlet requests (last minute)	
3	Avg Request Time (last minute)	
4	Cumulative time in this Servlet (ms)	This metric can be slightly less than the sum of the time taken to serve the requests recorded in the web server access log.
7	Servlet Name	

Events 152 and 153 give a good indication of the total amount of response time consumed in the PIA, although the sum of the durations in the access log may add up to slightly more than the time recorded for the event. For example, Listing 10-18 shows the results of querying, for a 5-minute interval, the event 153 data for the `psc` servlet.

Listing 10-18. *Details of part of event 153*

```
DBNAM PM_HOST_PORT          PM_AGENT_DTTM       PM_EVENT_DEFN_ID
DESCR60
PM_METRICLABEL1                PM_METRIC_VALUE1
PM_METRICLABEL2                PM_METRIC_VALUE2
PM_METRICLABEL3                PM_METRIC_VALUE3
PM_METRICLABEL4                PM_METRIC_VALUE4
PM_METRICLABEL7                PM_METRIC_VALUE7
------------------------------ --------------------
HR88  go-faster-3:7201:7202 16:57:29 24.08.2004              153
Web servlet status reported by WebServer Resource Monitor
Requests to this Servlet                       5
Servlet Requests (last minute)                 0
Avg Request Time (last minute)                 0
Time in this Servlet (ms)                  25637
Servlet Name                   /psc
```

Meanwhile, Listing 10-19 shows, for the same period, the entries in the access log for the `psc` servlet.

Listing 10-19. *Corresponding web server access log entries*

```
2004-08-24   16:52:35   0.762   3650   10.0.0.8   GO-FASTER-3   GET
/psc/ps/EMPLOYEE/HRMS/s/WEBLIB_PT_NAV.ISCRIPT1.FieldFormula.IScript_PT_NAV_PAGELET
Bodyid=true&c=YN5k5WtxjcM%3d
2004-08-24   16:54:33   0.28   0   10.0.0.8   GO-FASTER-3   GET
200   /psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL...
2004-08-24   16:54:37   2.123   0   10.0.0.8   GO-FASTER-3   POST
200   /psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL   -
2004-08-24   16:54:43   2.984   0   10.0.0.8   GO-FASTER-3   POST
200   /psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL   -
2004-08-24   16:55:06   19.498   0   10.0.0.8   GO-FASTER-3
POST   200   /psc/ps/EMPLOYEE/HRMS/c/ADMINISTER_WORKFORCE_(GBL).JOB_DATA.GBL   -
```

The entries in the web server access log total 25.647 seconds, but the Process Monitor reports only 25.637 seconds. It should also be possible to correlate the cumulative servlet time reporting by this event with the CPU consumption recorded in event 300.

If you do configure a system to self-monitor, this event will also report the time spent in the monitoring servlets. It will give an indication of the overhead of self-monitoring.

Event ID 200: Resources per Process

Each instrumented server process generates this event to record operating system resource consumption by those servers (see Table 10-8).

Table 10-8. *Events Sent by All C++ Processes*

Metric	Description	Comment
1	%CPU Used	
2	CPU Time (secs)*	This is the amount of CPU time consumed by the server process since it started.
3	VM (bytes)	
4	Working Set (bytes)	

** CPU Time (metric 103) is delivered as type 3 (numeric identifier) when it should actually be type 1 (counter). This does not affect the behavior of the Performance Monitor, but counters should be compared with the previously recorded value for the same agent.*

Event ID 300: Host Resource Status

The resource status events are generated by the PSMONITORSRV (see Table 10-9). Each application server and Process Scheduler Tuxedo domain will include an instance of this server. If they are all running on the same physical server, then they will all report similar data.

Table 10-9. *Host Resource Status*

Metric	Description	Comment
1	%CPU Used	
2	%Memory Used	
3	Hard Page Faults/Second	
4	Total Tuxedo Connections	Closely corresponds to number of web/application server connections in event 150.
5	Total Tuxedo Requests Queued	

The total number of connections could also be obtained by counting the rows returned from the `printclient` command in the Tuxedo `tmadmin` utility.

Event ID 301: Tuxedo "pq" Row

This event retrieves the same information as the `printqueue` command in the Tuxedo `tmadmin` utility (see Chapter 9). One event is written for each queue in the domain for application server and Process Scheduler domains. The PSMONITORSRV server reads this information directly from the Tuxedo Bulletin Board. Table 10-10 contains the metrics and their descriptions.

Table 10-10. *Reported in Groups to Simulate a Tuxedo Command-Line "pq"*

Metric	Description	Comment
1	Server Count	Number of server processes on the queue.
2	Queue Length	Number of requests currently on the queue.
3	Avg. Queue Length	Not reported by `tmadmin` unless load balancing is enabled, in which case it returns –1.
7	Queue Name	Name of Tuxedo queue. If the queue name is not defined in Tuxedo, it defaults to a string containing the Tuxedo group and server IDs.

Event ID 302: Tuxedo "psr" Row

This event retrieves the same information as the `printserver` command in the Tuxedo `tmadmin` utility (see Chapter 9). One event is written for each server process in the domain for application server and Process Scheduler domains. The PSMONITORSRV server reads this information directly from the Tuxedo Bulletin Board. Table 10-11 contains the metrics and their descriptions.

Table 10-11. *Reported in Groups to Simulate a Tuxedo Command-Line "psr"*

Metric	Description	Comment
1	Server Instance	Tuxedo server ID. This value is only unique within the queue.
2	Total Requests*	Total number of requests that this server ID has handled.
3	PID	Operating system process ID of the server process
7	Server Name	Name of server process

** Total Request (metric 115) is delivered as type 3 (numeric identifier) when it should actually be type 1 (counter). This does not affect the behavior of the Performance Monitor, but counters should be compared with the previously recorded value for the same agent.*

The additional description shows the name of the Tuxedo service that the server process is currently handling.

The metrics collected by events 150, 300, 301, and 302 effectively contain the same information as the `Tuxmon` scripts described in Chapter 13.

Event ID 350: Master Scheduler Status

This event provides an overview of each Process Scheduler configured on the database (see Table 10-12).

Table 10-12. *Master Scheduler Status*

Metric	Description
1	Active Processes
2	Queued Processes
3	Blocked Processes
4	Unused Process Slots

Note that the Process Scheduler name is not recorded on this event. It would be advisable to give the Tuxedo domain ID that includes the scheduler name.

Event ID 351: Master Scheduler Detail

An event is generated for each process type configured for each Process Scheduler (see Table 10-13). This event does include the server name.

Table 10-13. *Master Scheduler Detail per Process Type*

Metric	Description
1:120	Active Processes
4:123	Unused Process Slots
7:141	Server Name

Events 350 and 351 provide an indication of the level of batch activity in a system at any time. Users of systems with a significant simultaneous batch and online load sometimes report that online performance degrades when there is a heavy batch load. These metrics should provide the ability to detect any correlation.

Event ID 600: PSPING

When you use the PeopleSoft Ping utility, the Ping also generates an event. The Ping metrics are stored in PS_PTP_TST_CASES on the monitored system by the Ping component itself, but an event is generated and sends the same metrics to the monitoring system as well (see Table 10-14).

Table 10-14. *Ping Metrics Forwarded from the Browser*

Metric	Description
1	Network Latency (ms)
2	Web server Latency (ms)
3	Application Server Latency (ms)
4	DB Latency (ms)
7	IP Address

The Operator ID and Java Session ID are stored in the additional information.

Event ID 601: Override

This event provides a record of the performance monitoring traces performed on the system (see Table 10-15).

■ **Caution** If a user enables event monitoring and increases the current agent filter level, this could degrade system performance.

Table 10-15. *User Event: Monitoring Level Override*

Metric	Description
1	Agent Filter Mask
7	User Initiated PMU Name

Instrumentation

Table 10-16 shows all the metrics collected from various sources for the same PeopleSoft Ping transaction shown in Figure 10-9.

Table 10-16. *Metrics from a PeopleSoft Ping*

Duration (ms)	Description
1547	Total duration of Ping.
1342	Duration of servlet request as reported by the web server access log.
1322	Transaction 101: PIA view page request.
1112	Web server duration (including application server and database durations) of Ping.
1112	Transaction 115: Jolt request. This transaction is produced by the PIA servlet in the web monitor, so you would expect it to correspond to the duration of the Ping as also measured in the servlet. However, sometimes the duration of Ping can be 10 or 20 ms higher than this metric.
1082	Duration of Tuxedo ICPanel service as reported by the Tuxedo service trace.
1020	Transaction 400: Tuxedo service.
1012	Application server duration of Ping.

Duration (ms)	Description
901	Transaction: 401 ICPanel service.
861	Transaction 400: Duration of PeopleCode component only.
451	Database duration of Ping.
30	Active database time reported in Oracle SQL trace.

■ **Note** PeopleTools SQL trace was enabled when these measurements were taken. The overhead of this trace is responsible for a large discrepancy in the time reported by the database and the duration of the database call as measured by the application server.

We can see that nearly all the durations are different from each other. Figure 10-15 shows these metrics superimposed upon the now familiar PIA architecture diagram.

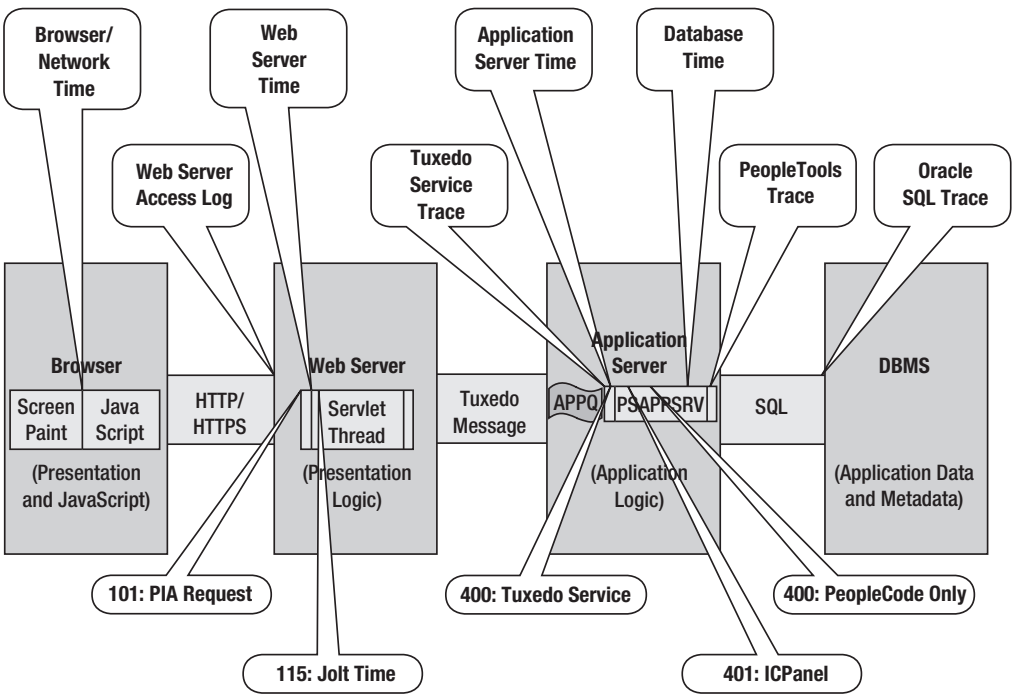


Figure 10-15. *Sources of metrics with Process Monitor transactions added*

The differences indicate that the measurements are collected at slightly different points in the PeopleSoft technology chain. Within the four tiers, the performance measurement instrumentation has been placed in different layers of software. Of course, each of these layers makes its contribution to the overall response time, as seen by the human operator in front of the browser!

Summary

Before the release of PeopleTools 8.4, getting performance metrics out of a PeopleSoft system required the variety of techniques discussed in Chapter 9, involving varying degrees of processing. On Unix systems, this often included writing scripts that were scheduled to periodically run utilities. On Windows, this approach was generally not possible without additional software.

As demonstrated in this chapter, PeopleSoft has made much more performance information readily available in PeopleTools 8.4. The metrics are stored in the database, so it is possible to answer more sophisticated questions by constructing SQL queries.

Performance Monitor represents a huge step forward in collecting performance for PeopleSoft systems. If its overhead is not too severe, it will be much easier to determine where in the system users are spending the most time waiting. Events can be periodically scheduled within the PeopleSoft system without the need for scripting or assistance from operating system administrators to schedule the task.

Performance issues are instinctively routed to the DBA. This is another area of PeopleSoft in which the DBA is going to need to be involved.

CHAPTER 11

SQL Optimization Techniques in PeopleSoft

The previous two chapters discussed how to obtain the performance metrics that can indicate the most time-consuming processes or activities on your system. In the case of the Application Engine and COBOL processes, traces can identify the particular step in the process that is costing the most time. Although significant amounts of time can be spent executing code in the client or application server processes, you'll often find that the most time-consuming step turns out to involve the execution of SQL.

When SQL performance is not the most significant part of a performance issue, then the problem is likely to be much more complicated to address. It may come down to the design of a process, which may require considerable customization, or it may be a feature of the PeopleSoft-supplied program that you either cannot see or cannot change. At least if the issue is SQL performance then, as the DBA, you can see what is going on, and perhaps you can do something about it, possibly without having to make a code change.

This chapter has several purposes:

- To demonstrate how to enable Oracle SQL trace to capture information about the SQL being issued by the application
- To explain how to find where in the application the SQL is defined
- To determine whether, and to what extent, it is possible to change the code

It will not explain how to optimize SQL statements, nor how and why the various techniques for optimization work (there many Oracle-specific books on that subject).

Enabling Oracle SQL Trace

Performance metrics may direct you to a particular process and sometimes a particular part of a process. If SQL performance is the problem, then the next question is, which SQL statement or statements are consuming the most time and why?

The timings reports for the COBOL and Application Engine processes can provide valuable information that can direct you to a particular step or statement. Furthermore, for each step, the Application Engine timings can distinguish between time spent on SQL and time spent executing PeopleCode.

The timings reports can sometimes identify a particular SQL statement. You could then use the `EXPLAIN PLAN` command to generate an execution plan. However, this command may generate a different execution plan from the one that was actually generated and used when the process ran. The reasons for this include, but are not limited to, a change in statistics, environment, or indexing. For instance, PeopleSoft batch processes often truncate and repopulate working storage tables and then refresh cost-based optimizer statistics on those tables.

The SQL trace of a process will record all the SQL statements that were issued and how long they took to execute. The execution plan for each SQL statement is written to the trace in the STAT lines when the cursor is closed. It is the only way to know for certain exactly how the SQL was executed when it ran in that process. So, the question is how to enable SQL trace.

In the sections that follow, I discuss various techniques to enable SQL trace. There are ways to add code to the application so that it issues the commands to enable and disable tracing. However, in practice, this is rarely helpful because performance problems usually arise in production and final test environments, where code change is simply not an option. For processes initiated via the Process Scheduler, a trigger can be used to issue the commands to enable SQL trace. For online processing, it is necessary to identify the database session and enable trace in that session.

Oracle Initialization Parameters and SQL Trace

Before you enable SQL trace, you need consider several Oracle initialization parameters:

- `TIMED_STATISTICS` must be set to TRUE in order to obtain timings in the SQL trace file or from any of the dynamic performance views (such as V$WAITSTAT). Without this parameter enabled, you cannot determine the longest running statements because they will all report a duration of zero. I generally recommend that this parameter should always be enabled. The overhead of this parameter is low, but without it you have no idea what is going on. It can, at least, be enabled dynamically.
- The trace files are written to the directory specified by `USER_DUMP_DEST`. If the output directory fills up, no error is generated. Oracle merely stops writing the trace files. However, if that directory is on the same file system as the archive log destination, the database could hang.
- The trace file size is limited by `MAX_DUMP_FILE_SIZE`. The parameter is expressed in operating system blocks (usually 512 bytes). From Oracle 9, it can also be expressed in kilobytes or megabytes, or it can be set to UNLIMITED. When a trace file exceeds this size, the Oracle process writes the warning message "DUMP FILE SIZE IS LIMITED" and then stops writing to it. PeopleSoft processes can easily produce large trace files. If you work with a truncated trace file, you are not analyzing the whole process, and you may not be working on the biggest performance problem.

- On Unix systems, the trace files can be read only by a member of the `dba` group. But if you set `_TRACE_FILES_PUBLIC` to TRUE, the files can be read by any Unix user. However, trace files can contain data values embedded in SQL statements even if bind variables are not logged, so you should be aware of the possible security implications of making these files viewable by anybody with a Unix account on the database server.
- The `STATISTICS_LEVEL` parameter was introduced in Oracle 9.2. Setting it to ALL will produce row source details in the STAT lines of the trace file. This parameter defaults to TYPICAL.
- `TRACEFILE_IDENTIFIER` can be set in a session to a string, which will be included in the trace file. If it is set or changed after the tracing is enabled, the trace will switch to a new file, named with the new identifier. However, be careful when doing this. If you switch trace files while you have open cursors, then `tkprof` may not generate accurate results for either trace file.

Analyzing SQL Trace Files with TKPROF

Oracle provides the `TKPROF` utility to translate raw trace file data into a readable report. It is documented in the *Oracle Performance Tuning Guide*. In most situations, I find it adequate to list the top ten SQL statements in a trace by total execution time, and work through them. Listing 11-1 shows a sample `tkprof` command to produce such a report.

Listing 11-1. `tkprof` *command*

```
tkprof hr88_ora_1748.trc hr88_ora_1748.ela sort=fchela,prsela,exeela print=10 sys=no
```

The various command parameters are described as follows:

- `SORT`: In this case, the SQL statements are sorted by the sum of the elapsed parse, execution, and fetch times for each distinct statement.
- `PRINT`: The output file lists the top ten statements according to the sort order.
- `SYS`: Recursive SQL is not reported.
- `EXPLAIN`: `tkprof` connects to the database and generates an execution plan with the `EXPLAIN PLAN` command, in addition to reporting the execution plan in the STAT lines in the trace.

■ **Caution** The additional execution plan generated by the EXPLAIN PLAN option should be approached with caution, because it may be different from the plan that was used when the SQL was traced and that was recorded in the STAT lines.

From Oracle 9.2, this option is usually unnecessary because the STAT lines contain both table and index names when the row source statistics are collected (see Listing 11-2).

Listing 11-2. *STAT lines in an Oracle 9.2.0.5 trace*

```
STAT #15 id=1 cnt=1 pid=0 pos=1 obj=0 op='UPDATE  (cr=1 r=0 w=0 time=146 us)'
STAT #15 id=2 cnt=1 pid=1 pos=1 obj=124398 op='INDEX UNIQUE SCAN PS_PSPRCSQUE
(cr=1 r=0 w=0 time=15 us)'
```

However, in Oracle8*i* the STAT lines do not contain index names (see Listing 11-3), so `tkprof` reports table names but only index object numbers. The `EXPLAIN PLAN` command will produce index names, but you should check that the execution plan is the same as the one in the STAT lines.

Listing 11-3. *STAT lines in an Oracle 8.1.7.4 trace*

```
STAT #15 id=1 cnt=1 pid=0 pos=0 obj=0 op='UPDATE PSPRCSRQST '
STAT #15 id=2 cnt=1 pid=1 pos=1 obj=26955 op='TABLE ACCESS BY INDEX ROWID PSPRCSRQST '
STAT #15 id=3 cnt=1 pid=2 pos=1 obj=27133 op='INDEX UNIQUE SCAN '
```

STAT lines are only emitted to the trace file when the cursor closes. If a session terminates abnormally with open cursors, or if the trace file reaches the `MAX_DUMP_FILE_SIZE` before a cursor is closed, then `tkprof` will not find STAT lines and will not report an execution plan.

Enabling SQL Trace for PeopleTools Clients

PeopleSoft has built its own sophisticated trace facility into all its processes that will report nearly everything that happens. This trace facility was principally intended as a debugging tool, but it also includes the amount of time that the operation took to execute and the time since the last traced operation completed. The timings can be affected by network performance and some operations not traced, for example:

- SQL that updates the operator's password stored in the database is not logged.
- If `DBMonitoring` is enabled, PeopleTools uses the DBMS_APPLICATION_INFO PL/SQL package to register the Operator ID with the database. This is not logged in the PeopleTools trace.

The Windows run-time client was not a supported run-time option in PeopleTools 8.1, and it has disappeared in PeopleTools 8.4. Only nVision and Query remain. It is possible to enable Oracle SQL trace, usually from another session, on the Windows client processes, but since PeopleTools 7.*x* this has not worked well. The Windows processes are multithreaded and, in 2-tier mode, maintain multiple connections to the database. There is also an inactivity timeout that disconnects the client from the database, thus terminating the session and disabling trace. When the user returns, the client will silently reconnect to the database, but SQL trace will not resume, because the client creates a brand-new database session.

Enabling SQL Trace for Application Server Processes

On startup, each PeopleSoft application server process[1] creates and maintains a single persistent database session that is used for every request it handles. To generate an Oracle SQL trace of

1. Except PSWATCHSRV, which was introduced in PeopleTools 8.4.

a single PIA client's activity, it would be necessary to provide a PIA domain and an application server with only a single PSAPPSRV process exclusively for that one client. Thus all of the activity goes through a single database session.

If you wish to separately trace Query or nVision activity, then you should configure the PSQRYSRV process, which is dedicated to running queries from these products. There is now one easily identifiable process for which trace can be enabled and disabled from another session.

Enabling Trace in Another Session

You can enable tracing of another session using the PL/SQL packages supplied by Oracle. The following commands enable tracing:

```
EXECUTE sys.dbms_system.set_sql_trace_in_session(<sid>,<serial#>,TRUE);
EXECUTE sys.dbms_support.start_trace_in_session(<sid>,<serial#>
                                              ,waits=>TRUE, binds=>FALSE);
EXECUTE sys.dbms_system.set_ev(<sid>,<serial#>,10046,8,'');
EXECUTE sys.dbms_monitor.session_trace_enable(<sid>,<serial#>
                                             ,waits=>TRUE,binds=>FALSE);
```

To disable tracing, use one of the following:

```
EXECUTE sys.dbms_system.set_sql_trace_in_session(<sid>,<serial#>,FALSE);
EXECUTE sys.dbms_support.stop_trace_in_session(<sid>,<serial#>);
EXECUTE sys.dbms_system.set_ev(<sid>,<serial#>,10046,0,'');
EXECUTE sys.dbms_monitor.session_trace_enable(<sid>,<serial#>,
                                              waits=>FALSE, binds=>FALSE);
```

Note that

- SID and SERIAL# are obtained from V$SESSION.
- The DBMS_SYSTEM package is only accessible by SYSDBA or DBA accounts—and for good reason, as the package also contains procedures you may not want the average user to have (i.e., KSDWRT writes trace files or to the `alert.log`). It is best to encapsulate these procedures within another custom-written stored procedure, and then grant access on that.
- The Oracle-supplied DBMS_SUPPORT package, created by the script `$ORACLE_HOME/rdbms/admin/dbmssupp.sql`, is not loaded by default when the database catalogue is created.
- Up to Oracle 9, if you also want to enable SQL trace on another session with the wait events or bind variables, then you must use `set_ev`. This method is not supported by Oracle.
- The package DBMS_MONITOR is new in Oracle 10. It is documented in the *PL/SQL Packages Reference*, whereas DBMS_SUPPORT and DBMS_SYSTEM are not. It is the supported method to set higher levels of trace in other sessions.

Listing 11-4 shows a PL/SQL script that will enable SQL tracing on the PSAPPSRV processes connected to the database. It identifies application server processes from the CLIENT_INFO string that is written by the application server processes when `DBMonitoring` (see Chapters 9 and 12) is enabled in the application server configuration file. CLIENT_INFO includes the name of the server program. The script can, of course, be adjusted to further restrict the sessions traced by adding conditions to the driving query. For example, you may wish to trace only the application servers on a particular machine.

Listing 11-4. traceallon.sql*: Enabling trace on application server processes*

```
set serveroutput on buffer 1000000000 echo on
spool traceallon
DECLARE
   CURSOR c_appsess IS
   SELECT *
   FROM   v$session
   WHERE  type = 'USER'
-- AND    program like '%PSAPPSRV%'
   AND    client_info like '%,PSAPPSRV%';
   p_appsess c_appsess%ROWTYPE;
BEGIN
   OPEN c_appsess;
   LOOP
      FETCH c_appsess INTO p_appsess;
      EXIT WHEN c_appsess%NOTFOUND;
--    sys.dbms_system.set_sql_trace_in_session(
--                    p_appsess.sid, p_appsess.serial#,TRUE);
--    sys.dbms_support.start_trace_in_session(p_appsess.sid, p_appsess.serial#);
      sys.dbms_system.set_ev(p_appsess.sid, p_appsess.serial#,10046,8,'');
      sys.dbms_output.put_line('Enable:'
                               ||p_appsess.sid||','||p_appsess.serial#);
   END LOOP;
   CLOSE c_appsess;
END;
/
```

Trace can be disabled in a similar way.

Using an AFTER LOGON Trigger

SQL trace can also be enabled from an AFTER LOGON database trigger. However, unless the client program name is properly registered with the database, so that it appears in V$SESSION.PROGRAM, it can be difficult to work out what program is connecting to the database. The program name registers for IPC connections on all platforms, and via SQL*Net on Windows and AIX, but not on HP-UX or Solaris.

The trigger in Listing 11-5 will trace all processes that are called something beginning with "PSAPPSRV" in any case.

Listing 11-5. trace_connect_trigger.sql*: Enabling SQL trace after database logon*

```
CREATE OR REPLACE TRIGGER sysadm.connect_trace
AFTER LOGON
ON sysadm.schema
DECLARE
    l_tfid VARCHAR2(64);
```

```
BEGIN
-- if this query returns no rows an exception is raised and trace is not set
   SELECT SUBSTR(TRANSLATE('''
                ||TO_CHAR(sysdate,'YYYYMMDD.HH24MISS')
                ||'.'||s.program
                ||'.'||s.osuser
                ||''''
               ,' \/','___'),1,64)
   INTO   l_tfid
   FROM   v$session s
   WHERE  s.sid IN(
              SELECT sid
              FROM   v$mystat
              WHERE  rownum = 1)
   AND    UPPER(s.program) LIKE '%PSAPPSRV%';

   EXECUTE IMMEDIATE 'ALTER SESSION SET TIMED_STATISTICS = TRUE';
   EXECUTE IMMEDIATE 'ALTER SESSION SET MAX_DUMP_FILE_SIZE = UNLIMITED';
   EXECUTE IMMEDIATE 'ALTER SESSION SET TRACEFILE_IDENTIFIER = '||l_tfid;
   EXECUTE IMMEDIATE 'ALTER SESSION SET EVENTS ''10046 TRACE NAME CONTEXT
FOREVER, LEVEL 8''';

   EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

Remember that CLIENT_INFO is still NULL at connect time because it is set by the application only after the AFTER LOGON trigger completes.

Enabling SQL Trace on the Process Scheduler

We have already seen in Chapter 9 how processes that are run via the Process Scheduler log update their request record on PSPRCSRQST, with the processes' start and end times, and also the processes' status.

When the batch process starts, it updates its own RUNSTATUS on the table PSPRCSRQST to 7, to indicate that it is processing. When the process terminates, it updates the status to 9, indicating success. A trigger configured to fire on this change of status will be run in the batch process's own database session. Now you have a way to introduce additional code that runs very close to the start or end of the process, without changing any of the program code.

Enabling SQL Trace for Batch Processes with a Trigger

The trigger shown in Listing 11-6 will enable Oracle SQL trace when a process starts.

Listing 11-6. `trace_trigger.sql`

```
rem requires following grants to be made explicitly by sys
rem GRANT ALTER SESSION TO sysadm;
rollback;
```

```
CREATE OR REPLACE TRIGGER sysadm.set_trace
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.runstatus = 7 AND old.runstatus != 7
AND   new.prcstype IN('Application Engine','COBOL SQL','Crystal',
        'SQR Process','SQR Report','SQR Report For WF Delivery')
)
BEGIN
   EXECUTE IMMEDIATE 'ALTER SESSION SET TIMED_STATISTICS = TRUE';
   EXECUTE IMMEDIATE 'ALTER SESSION SET MAX_DUMP_FILE_SIZE = 2048000';
   EXECUTE IMMEDIATE 'ALTER SESSION SET STATISTICS_LEVEL=ALL';
   EXECUTE IMMEDIATE 'ALTER SESSION SET TRACEFILE_IDENTIFIER = '''||
      REPLACE(:new.prcstype,' ','_')||'_'||
      REPLACE(:new.prcsname,' -','__')||'_'||
      :new.prcsinstance||'''';
-- levels: 1=basic trace, 4=bind variable, 8=wait statistics
   EXECUTE IMMEDIATE 'ALTER SESSION SET EVENTS ''10046 TRACE NAME CONTEXT
FOREVER, LEVEL 8''';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

■ **Tip** The exception at the end of the trigger prevents the UPDATE statement from failing, even if the trigger does not function. Otherwise, it can crash the Process Scheduler.

The trace file name will contain the process type, process name, and process instance number. For example, the trace file `hr88_ora_3752_application_engine_prcsyspurge_205.trc` was produced when the Application Engine process called PRCSYSPURGE ran on database HR88 with process instance 205. The ID of the Oracle shadow process was 3752. In addition to restricting the trigger to certain process types, you can enhance the WHEN clause further, so that the trigger fires only for certain processes, operators, process schedulers, time periods, or run controls.

For instance, in the example in Listing 11-7, the trigger will trace only the GL Journal Edit process when it is run by operator VP1 on the PSUNX Process Scheduler, and then only when the operator uses a run control called TRACEME. Furthermore, the trigger, although still enabled, ceases to fire after a specified time.

Listing 11-7. *WHEN clause of a trigger*

```
WHEN (new.runstatus = 7 AND old.runstatus != 7
AND new.prcstype = 'COBOL SQL'
AND new.prcsname = 'GLPJEDIT'
AND new.oprid = 'VP1'
AND new.servernamerqst = 'PSUNX'
AND new.runcntlid = 'TRACEME'
AND new.rqstdttm <= TO_DATE('200404231700','YYYYMMDDHH24MI')
)
```

This trigger will not work in some circumstances:

- It should not be extended for use with the process type PSJobs. A PSJob is not an actual process, but a reference to several processes run as a group. The status of the PSJob is updated by the Process Scheduler. The trigger would trace the Process Scheduler.
- In PeopleTools 7, if a Crystal Reports or nVision process is run from a Windows client in 3-tier mode, the status of the request record is updated by the PSSAMSRV process in the application server. So that process would be traced rather than the PSQRYSRV process that executes the query SQL.

The other Oracle events shown in Table 11-1 can also be used to further enhance the content of the SQL trace file.

Table 11-1. *Other Oracle Events*

Event	Level	Description
10128	2	Partition elimination
10053	1	CBO decision-tree analysis
10104	1	Hash activity
10032	1	Sort statistics
10033	1	Sort I/O

In the Application Engine from PeopleTools 8.4, the Process Scheduler is fully integrated with Tuxedo. There are Application Engine server processes, PSAESRV, that respond to Tuxedo service requests from the Process Scheduler server. Like other PeopleSoft Tuxedo processes, the PSAESRV processes each maintain a single persistent connection to the database.

If you enable session trace with the trigger, then you will continue to trace subsequent Application Engine process requests handled by the same server process. This is discussed in more detail in Chapter 14. The solution is to create a second trigger to disable trace when the process completes, and update the status on the Process Scheduler request to something other than processing.

Enabling SQL Trace Programmatically

Occasionally you may want to trace a specific part of a batch process rather than use the trigger to trace the entire process. In order to do this, SQL or PL/SQL commands to enable and disable trace must be added to the program in appropriate places.

Application Engine

Any valid SQL command can simply be coded as a free-format SQL statement in an Application Engine SQL step. Thus, it is possible to add code to enable and disable session trace, as required, within a process.

Figure 11-1 shows an Application Engine program in the Application Designer. In this example, I have added a step to an Application Engine program that issues the `ALTER SESSION` command.

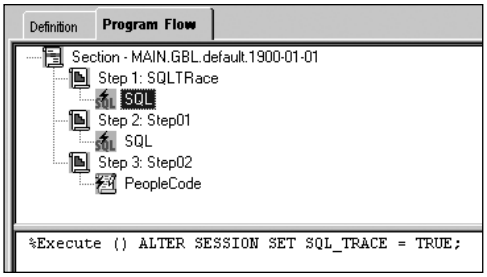


Figure 11-1. *Part of an Application Engine program*

SQR

In SQR, it is simply a matter of coding the ALTER SESSION statement. You might even want to have an SQC file that contains a standard function, as in Listing 11-8. It can then simply be #INCLUDEd and called. The code can be removed later.

Listing 11-8. `session.sqc`*: Standard procedures to enable and disable trace in SQR*

```
!****************************************
!  session.sqc:  oracle session tracing  *
!****************************************
!*************************************************************************
! Function:    enable_session_trace                                      *
!                                                                        *
! Description: enable oracle session trace                               *
!                                                                        *
!*************************************************************************
begin-procedure enable_session_trace

display 'Enabling Oracle Session Trace'

begin-sql
ALTER SESSION SET SQL_TRACE=TRUE
end-SQL

end-procedure

!*************************************************************************
! Function:    disable_session_trace                                     *
!                                                                        *
! Description: disable oracle session trace                              *
!                                                                        *
!*************************************************************************
begin-procedure disable_session_trace

display 'Disabling Oracle Session Trace'
```

```
begin-sql
ALTER SESSION SET SQL_TRACE=FALSE
end-SQL

end-procedure
```

These procedures can be called as shown in Listing 11-9.

Listing 11-9. *Calling procedures in an SQR*

```
begin-REPORT
  do enable_session_trace
...
...
...
  do disable_session_trace
end-REPORT

#include 'session.sqc'
...
```

Remote Call

Some batch processing, mostly COBOL programs, can be initiated synchronously by the PIA. Behind the scenes, the application server initiates the batch process and waits for it to complete. For example, in Financials, a single voucher can be edited and posted "online" by clicking a button on a page, at which point the application server runs a COBOL process, GLPJEDIT. This is the same process that is run by the Process Scheduler to process many vouchers in a single execution. When a process is executed by remote call, there is no request record for the process on the PSPRCSRQST table to be updated, so the trigger described for scheduled processes (see Listing 11-6) will not fire.

However, when a batch process starts, one of the first things it does is insert a row to the PS_MESSAGE_LOG table, and then it commits the insert so that the message can be seen in the Process Monitor. A trigger can be created to execute on this insert (see Listing 11-10).

Listing 11-10. *Trigger to trace a* `RemoteCall` *process*

```
CREATE OR REPLACE TRIGGER sysadm.trace_remotecall
BEFORE INSERT ON sysadm.ps_message_log
FOR EACH ROW
WHEN (new.process_instance = 0
AND   new.message_seq      = 1
AND   new.program_name     = 'GLPJEDIT'
AND   new.dttm_stamp_sec  <= TO_DATE('200407231500','YYYYMMDDHH24MI')
)
BEGIN
   EXECUTE IMMEDIATE 'alter session set TIMED_STATISTICS = TRUE';
   EXECUTE IMMEDIATE 'alter session set MAX_DUMP_FILE_SIZE = 2048000';
   EXECUTE IMMEDIATE 'alter session set TRACEFILE_IDENTIFIER = '''||
                              replace(:new.program_name,' -','__')||'''';
```

```
   /* disk waits(8) and bind variables(4)*/
   EXECUTE IMMEDIATE 'alter session set events ''10046 trace name context
forever, level 8''';

EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

There are a number of things to note about this trigger:

- Processes executed by `RemoteCall` always have a process instance of 0, so the WHEN clause is restricted to this value.
- The trigger fires only on the first message to be inserted, which has a MESSAGE_SEQ of 1.
- The process name is written to the message log, so the trigger can be restricted to particular processes.
- The messages are timestamped, so the trigger can be restricted to fire in a particular time window.

Where Does This SQL Come From?

Many techniques and tools that are employed in performance tuning are very good at identifying problem SQL statements that consume large amounts of time and resources. Sometimes, a performance problem can be resolved by making changes to the database (parameters, indexes, statistics, or physical storage options). However, in many cases, the solution involves changing the code (hints, FROM clause order, or additional joins). So it is valuable to know where a SQL statement is defined in the application.

PeopleSoft does not use the DBMS_APPLICATION_INFO.SET_MODULE PL/SQL packaged procedure supplied by Oracle to register the program name, so there is no direct way of associating a SQL statement with a particular piece of the application.

However, the SQL issued by PeopleSoft applications is stored in various ways and is often generated dynamically. In the following sections I explain how the way in which a piece of SQL is formatted or structured can give clues as to how you can find it in the PeopleSoft development tools.

If you can find it, you can change it.

Component Processor

The SQL issued by the PeopleTools client (either by the application server or by the Windows client) is dynamically generated by the component processor, which interprets and executes the application stored in the PeopleTools tables.

The SQL code, as shown for example in Listing 11-11, is generally all uppercase. Each scroll in a page relates to a single PeopleSoft record, and there will be a query of the single table or view corresponding to that record. There will be a WHERE clause condition for each key column, which will also appear in the ORDER BY clause. Date columns will be converted to a character string in the format `'YYYY-MM-DD'`.

Listing 11-11. *PIA component SQL*

```
SELECT EMPLID, PER_STATUS, TO_CHAR(BIRTHDATE,'YYYY-MM-DD'), BIRTHPLACE, BIRTHCOUNTRY,
BIRTHSTATE, TO_CHAR(DT_OF_DEATH,'YYYY-MM-DD'), TO_CHAR(ORIG_HIRE_DT,'YYYY-MM-DD'),
HIGHLY_COMP_EMPL_C, HIGHLY_COMP_EMPL_P FROM PS_PERSON WHERE EMPLID=:1 ORDER BY EMPLID
```

There are a number of variations:

- Queries generated from search dialogs are always DISTINCT. This cannot be suppressed.
- It is common to see a criterion on a security column in search dialog SQL, such as ROWSECCLASS, as in the example in Listing 11-12. This criterion is added automatically when one of a number of special columns, such as ROWSECLASS, is present in the search record.
- Anything that the user types into the search criteria, in either a search dialog or a related display search, will appear as a literal value in a WHERE clause condition.
- The `UPPER()` functions are applied (as shown in Listing 11-12) when searching mixed-case fields (as defined in the Application Designer) and when case-insensitive searching is permitted in the PeopleTools options (see Chapter 5).

Listing 11-12. *Case-insensitive search*

```
SELECT DISTINCT EMPLID, EMPL_RCD, NAME, LAST_NAME_SRCH, SETID_DEPT, DEPTID,
NAME_AC, PER_STATUS FROM PS_PERS_SRCH_GBL WHERE ROWSECCLASS=:1 AND UPPER(NAME)
LIKE UPPER('Smith') || '%' ESCAPE '\' ORDER BY NAME, EMPLID
```

From PeopleTools 8.43, the SQL statements that generate lists of translate values (see Listing 11-13) have a FIRST_ROWS hint. This query fetches all the rows in one operation without waiting any longer, and then it loads them into the component buffer. I don't understand why PeopleSoft has chosen to instruct the optimizer to retrieve the first row as cheaply as possible.

Listing 11-13. *Getting translate values*

```
SELECT  /*+ FIRST_ROWS */ NAME_TYPE, ORDER_BY_SEQ, NAME_TYPE_DESCR
FROM PS_NAME_TYPE_TBL A ORDER BY NAME_TYPE
```

PeopleCode

PeopleCode is PeopleSoft's proprietary programming language. Explicit SQL statements can be executed in PeopleCode. Certain PeopleCode functions that populate a scroll in a page can accept a WHERE clause that is built into the SQL executed by the command. PeopleCode is executed by the component processor, and from PeopleTools 8.1 it can also be executed by the Application Engine.

A mixed-case SQL statement from a PIA trace, such as in Listing 11-14, indicates SQL embedded in PeopleCode. This is simply because a lot of PeopleSoft developers are not rigorous about how they write SQL code. However, some SQL embedded in PeopleCode is all uppercase because that is how it was coded, and that SQL can be confused as generated by the component processor.

Any complex SQL, or SQL that joins several tables, is also likely to be PeopleCode. For example, the SQL shown in Listing 11-14 was extracted from a trace file.

Listing 11-14. *SQL submitted by a* `SQLExec()` *function*

```
Select A.BEN_STATUS from PS_ACTN_REASON_TBL A where A.ACTION = :1 and A.ACTION_REASON =
(Select min(AA.ACTION_REASON) from PS_ACTN_REASON_TBL AA where AA.ACTION = A.ACTION) and
A.EFFDT = (Select max(AAA.EFFDT) from PS_ACTN_REASON_TBL AAA where AAA.ACTION = A.ACTION
and AAA.ACTION_REASON = A.ACTION_REASON)
```

The SQL comes from the PeopleCode `SQLExec()` function shown in Listing 11-15. This function passes the SQL in the string parameter through to the database. What you code is what you get. The statement is also in mixed case and has multicharacter table aliases, both of which suggest a `SQLExec()` PeopleCode function.

Listing 11-15. `SQLExec()` *function in PeopleCode*

```
   SQLExec("Select A.BEN_STATUS from PS_ACTN_REASON_TBL A where A.ACTION = :1 and
A.ACTION_REASON = (Select min(AA.ACTION_REASON) from PS_ACTN_REASON_TBL AA where AA.ACTION
= A.ACTION) and A.EFFDT = (Select max(AAA.EFFDT) from PS_ACTN_REASON_TBL AAA where
AAA.ACTION = A.ACTION and AAA.ACTION_REASON = A.ACTION_REASON)", &ACTION, &FETCH_STATUS);
```

An uppercase SELECT and FROM statement followed by a mixed-case WHERE clause suggests a function in PeopleCode populating a scroll on a page or a rowset. Where the alias of the table is "FILL," as in Listing 11-16, it is almost certainly a rowset being filled.

Listing 11-16. *SQL generated by a rowset* `Fill()` *function*

```
PSAPPSRV.2564   1-4321   01.52.36    0.551 Cur#1.2564.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
FILL.PNLNAME,FILL.PNLFLDID,FILL.FIELDNUM,FILL.PNLFIELDNAME,FILL.FIELDTYPE,FILL.RECNAME,
FILL.FIELDNAME,FILL.LBLTYPE,FILL.GOTOPORTALNAME,FILL.GOTONODENAME,FILL.GOTOMENUNAME,
FILL.GOTOPNLGRPNAME,FILL.GOTOMKTNAME,FILL.GOTOPNLNAME,FILL.GOTOPNLACTION
FROM PS_CO_PNLFIELD_VW FILL  WHERE PNLNAME = :1 and FIELDTYPE = 16 and LBLTYPE = 7
AND RECNAME = :2 and FIELDNAME = :3
```

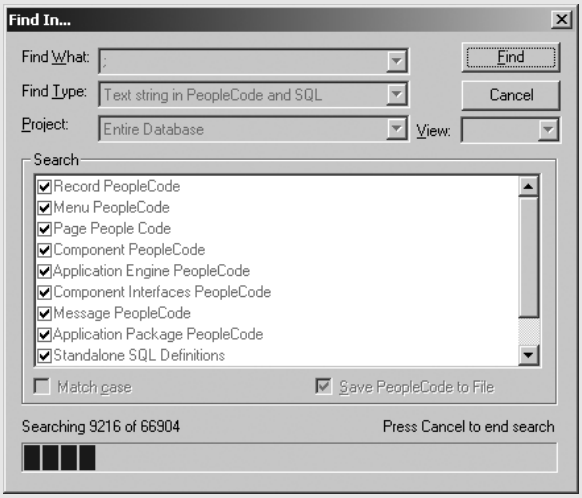
The SQL in Listing 11-16 is generated by the PeopleCode shown in Listing 11-17. The table name is obtained from the PeopleSoft record that is the parameter to the `CreateRowset()` function. The WHERE clause is applied to the `Fill()` function.

Listing 11-17. *Rowset function in PeopleCode*

```
    &PnlField_Rs = CreateRowset(Record.CO_PNLFIELD_VW);
      &PnlField_Rs.Flush();
      &PnlField_Rs.Fill("WHERE PNLNAME = :1 and FIELDTYPE = 16 and LBLTYPE = 7 AND
RECNAME = :2 and FIELDNAME = :3", %Page, &LinkRecName, &LinkFieldName);
```

SEARCHING PEOPLECODE

If you are trying to find the PeopleCode in which a particular SQL statement appears, the Application Designer provides a utility to search the PeopleCode, as shown in the following image. However, using this utility can be time consuming. Alternatively, you can generate a text file of all the PeopleCode in the system by searching for a semicolon (;) and saving the results to a flat file. You can then search the flat file relatively quickly with a text editor.



Query

The PeopleSoft Query utility is used for ad hoc queries and reporting. The tool is described in the PeopleTools Query PeopleBook. PeopleSoft positions the Query tool for "end users" as well as developers. Most of these end users will be completely nontechnical and have probably never heard of SQL. I am not going to comment on the wisdom or otherwise of permitting this, but I have seen PeopleSoft systems brought to their knees by unrestricted access to the ad hoc query tool!

Each Crystal Report has a corresponding query that feeds data to the report. Queries are also used in nVision and some view definitions in the Application Designer.

The Query utility generates monolithic SQL query statements that frequently join many tables. They can include effective date/sequence and tree subqueries that are automatically generated by PeopleTools. It is relatively easy to write queries that are functionally correct. It is usually more challenging to write a query that generates SQL and also executes efficiently.

The tables' aliases in a query are single letters allocated sequentially along the FROM clause. Gaps can appear in the sequence if a record is deleted from the query. When a record is added to a query, the first unused letter in the alphabet is used as the alias, hence the letter can get out of sequence.

If a record has a query security record defined, then the query security record is automatically added to the query when the record is selected. It is given the same alias as the table it secures, with a "1" appended. A record is joined to its query security records by the key columns they

have in common. For example, in Listing 11-18 PERSONAL_DTA_VW is given the alias D. It has a query security record PERS_SRCH_QRY, which is therefore given the alias D1. The two records are joined only by EMPLID, which is the only key column defined on both records in the Application Designer (hence the significance of defining key fields on a view, even though there is no corresponding index).

If multiple records have the same query security record, as shown in Listing 11-18, the security record appears only once in the FROM clause after the first table it secures, but it will be joined to both tables.

Listing 11-18. *Query TRN003__COURSE_WAITING_LIST*

```
SELECT A.EMPLID, A.ATTENDANCE, A.COURSE, B.DESCR, D.NAME, A.SESSION_NBR,
TO_CHAR(A.STATUS_DT,'YYYY-MM-DD'),B.COURSE
  FROM PS_TRAINING A, PS_COURSE_TBL B, PS_PERSONAL_DTA_VW D, PS_PERS_SRCH_QRY D1
  WHERE D.EMPLID = D1.EMPLID
    AND D1.ROWSECCLASS = 'HCDPALL'
    AND ( A.COURSE = :1
     AND A.ATTENDANCE IN ('S','W')
     AND A.COURSE = B.COURSE
     AND A.EMPLID = D.EMPLID )
```

Like other PeopleTools objects, queries are stored in the PSQRY% PeopleTools tables. If you have the SQL, it is possible to work out which query you are looking at by querying the PSQRYRECORD table, as shown in Listing 11-19. This table holds one row per record selected in each query.

Listing 11-19. `findqry.sql`

```
SELECT a.oprid, a.qryname
FROM   psqryrecord a
,      psqryrecord b
,      psqryrecord d
WHERE  a.oprid = b.oprid
AND    a.qryname = b.qryname
AND    a.oprid = d.oprid
AND    a.qryname = d.qryname
AND    a.corrname = 'A'
AND    a.recname = 'TRAINING'
AND    b.corrname = 'B'
AND    b.recname = 'COURSE_TBL'
AND    d.corrname = 'D'
AND    d.recname = 'PERSONAL_DTA_VW'
/
```

On a demo HR database, this query returns two rows (see Listing 11-20), indicating that there are two queries with at least similar FROM clauses.

Listing 11-20. *Results of* `findqry.sql`

```
OPRID                          QRYNAME
------------------------------ ------------------------------
                               TRN002__SESSION_ROSTER
                               TRN003__COURSE_WAITING_LIST
```

Query TRN002__SESSION_ROSTER is shown in Listing 11-21. You can see that the aliases on the tables in the FROM clauses are as specified in the query in Listing 11-19.

Listing 11-21. *Query TRN002__SESSION_ROSTER*

```
SELECT A.EMPLID, A.COURSE, A.SESSION_NBR, D.NAME, B.DESCR,
TO_CHAR(A.COURSE_START_DT,'YYYY-MM-DD'), A.ATTENDANCE,B.COURSE
  FROM PS_TRAINING A, PS_COURSE_TBL B, PS_PERSONAL_DTA_VW D, PS_PERS_SRCH_QRY D1
  WHERE D.EMPLID = D1.EMPLID
    AND D1.ROWSECCLASS = 'HCDPALL'
    AND ( ( A.ATTENDANCE = 'E'
     AND A.COURSE = B.COURSE
     AND D.EMPLID = A.EMPLID
     AND A.COURSE = :1
     AND ( A.SESSION_NBR = :2
     OR A.COURSE_START_DT = TO_DATE(:3,'YYYY-MM-DD'))) )
```

■ **Note** On some systems where operators have the right to develop ad hoc PeopleSoft queries, I have found that some operators habitually take a public query and save their own private version of it, and possibly adjust it slightly. The result is that when you find one problem SQL statement that is a query, you discover that there are sets of similar queries that all need to be addressed.

COBOL

SQL statements submitted by PeopleSoft COBOL programs are either read in from the stored statements table, PS_SQLSTMT_TBL, or dynamically generated from COBOL code. Stored statements will only use bind variables; the dynamically generated statements may contain literal values.

It can be difficult to distinguish between the two classes in the COBOL statement timings report (see Listing 9-43 in Chapter 9).[2] The point being that it is possible to change or hint the statements on the stored statements table, but to change the dynamic statements is likely to

2. The PeopleTools trace indicates whether a statement is a static statement stored in the database

   `GETSTMT Stmt=FSPJCOMB_S_COMGRP, length=297 COM Stmt=SELECT A.PROCESS_GROUP ...`

   or a dynamic statement

   `DYNAMIC Stmt=FSPJECHF_U_CFERROR COM Stmt=UPDATE PS_PSA_ACCTDSTGL SET ...`

   Trace has a significant run-time overhead; it is really intended as debugging tool. You would have to search the trace for the each statement.

require code changes. The PeopleSoft COBOL is extremely complicated, and making changes should be a last resort.

In the Global Payroll engine, some of the dynamic statements are derived from the payroll rules, which are stored in the database as metadata. These statements can be changed to a limited extent by the way the payroll rules are coded.

PeopleSoft delivers the stored statements as Data Mover scripts (in `$PS_HOME/src/cbl/base`) that are loaded into the database (see Listing 11-22). The scripts should be considered as the source for the stored statements. If you make changes and a new version of the script is delivered, you can use file compare utilities to detect and resolve the differences.

Listing 11-22. *Extract from the* `gppcancl.dms` *Data Mover script*

```
STORE GPPCANCL_D_WRKSTAT
DELETE FROM PS_GP_PYE_STAT_WRK
 WHERE CAL_RUN_ID=:1
   AND EMPLID BETWEEN :2 AND :3
;
```

Identifying Stored Statements

It is often useful to know which stored statement is which. The PL/SQL script in Listing 11-23 will add the name of the statement as a comment to identify the SQL.

Listing 11-23. `stmtid.sql`*: Script to add an identification comment to stored statements*

```
spool stmtid
set serveroutput on buffer 1000000000 echo on verify on feedback on

DECLARE
   CURSOR stmt_cursor IS
   SELECT *
   FROM   ps_sqlstmt_tbl;

   c_stmt stmt_cursor%ROWTYPE;
   l_stmt_text VARCHAR2(32767); /*for stmt text so can use text functions*/
   l_stmt_id   VARCHAR2(18);    /*PS stmt ID string*/
   l_len       INTEGER;         /*length of stmt text*/
   l_spcpos    INTEGER;         /*position of first space*/
   l_compos    INTEGER;         /*position of first comment*/
   l_compos2   INTEGER;         /*end of first comment*/
   l_idpos     INTEGER;         /*position of statement id*/
BEGIN
   OPEN stmt_cursor;
   LOOP
      FETCH stmt_cursor INTO c_stmt;
      EXIT WHEN stmt_cursor%NOTFOUND;

      l_stmt_id := c_stmt.pgm_name||'_'||c_stmt.stmt_type
                                   ||'_'||c_stmt.stmt_name;
```

```
      l_stmt_text := c_stmt.stmt_text;
      l_spcpos := INSTR(l_stmt_text,' ');
      l_compos := INSTR(l_stmt_text,'/*');
      l_compos2 := INSTR(l_stmt_text,'*/');
      l_idpos := INSTR(l_stmt_text,l_stmt_id);

--    sys.dbms_output.put_line(l_stmt_id);
--    sys.dbms_output.put_line(SUBSTR(l_stmt_text,1,100));
--    sys.dbms_output.put_line('Space at '||l_spcpos);
--    sys.dbms_output.put_line('Comment at '||l_compos);
--    sys.dbms_output.put_line('Comment End at '||l_compos2);
--    sys.dbms_output.put_line('ID at '||l_idpos);

      IF (l_idpos = 0 AND l_spcpos > 0 AND LENGTH(l_stmt_text)<=32000) THEN
         /*no id comment in string and its not too long so add one*/
         IF (l_compos = 0) THEN /*no comment exists*/
            l_stmt_text := SUBSTR(l_stmt_text,1,l_spcpos) ||'/*'||
                           l_stmt_id||'*/'||SUBSTR(l_stmt_text,l_spcpos);
         ELSE /*insert into existing comment*/
            l_stmt_text := SUBSTR(l_stmt_text,1,l_compos2-1)||
                           ' '||l_stmt_id||SUBSTR(l_stmt_text,l_compos2);
         END IF;

         UPDATE ps_sqlstmt_tbl
         SET    stmt_text = l_stmt_text
         WHERE  pgm_name  = c_stmt.pgm_name
         AND    stmt_type = c_stmt.stmt_type
         AND    stmt_name = c_stmt.stmt_name;

--       sys.dbms_output.put_line(SUBSTR(l_stmt_text,1,100));
      END IF;

   END LOOP;
   CLOSE stmt_cursor;
END;
/
```

Every time the COBOL program submits the SQL, it will contain the comment (see Listing 11-24), and when you find the statement in a trace, you will know where it came from. Dynamic statements will then be easily distinguished because they will not have a comment.

Listing 11-24. *Stored statement with an identifying comment*

```
DELETE /*GPPCANCL_D_WRKSTAT*/ FROM PS_GP_PYE_STAT_WRK
WHERE CAL_RUN_ID=:1 AND EMPLID BETWEEN :2 AND :3
```

SQR

PeopleSoft uses SQR as both a batch and reporting language. In SQR programs, SQL statements are just embedded in the SQR or SQC source files. These files can be edited with any text editor. Finding the SQL is usually just a matter of searching the source code.

However, many PeopleSoft SQR programs also dynamically generate all or part of a SQL statement in a string variable, and then use that string in the SQL statement, as shown in Listing 11-25.

Listing 11-25. *Example of SQR code with a variable embedded in the SQL*

```
...
FROM  PS_GP_CAL_RUN_DTL A,
      PS_GP_CALENDAR B,
      PS_GP_CAL_PRD C
WHERE A.CAL_RUN_ID = $Cal_Run_ID
[$Paygroup_Where]
AND   B.GP_PAYGROUP = A.GP_PAYGROUP
AND   B.CAL_ID = A.CAL_ID
AND   C.CAL_PRD_ID = B.CAL_PRD_ID
...
```

In Listing 11-25, `$Cal_Run_ID` is a normal bind variable. However, the square brackets around `$Paygroup_Where`, in Listing 11-26, indicate that the variable is not a bind variable, but that the contents of the variable are to be embedded in the SQL. As in this example, these variables often contain SQL keywords as well as the literal values of variables.

If the values of the variables change with each execution of the statement, the database will reparse the statement for each new value.

Listing 11-26. *SQR code to build the dynamic part of a WHERE clause*

```
    let $Where = ''
    if not isblank($Paygroup)
        let $Where = ' AND GRP.GP_PAYGROUP = ''' || $Paygroup || ''''
        let $Where_B = ' AND B.GP_PAYGROUP = ''' || $Paygroup || ''''
    End-If
```

Identifying comments and hints can be added manually to SQL statements in SQR programs. They must be placed in a line on their own, as shown in Listing 11-27.

Listing 11-27. *SQR code with an embedded hint and comment*

```
Begin-Procedure Check-Leavers-Float-Balance($Employee_Selection, $Effdt,
$GP_Paygroup, $GP_Cal_Run_ID)
...
Begin-Select On-Error=SQL-Error
/*+ORDERED Check-Leavers-Float-Balance*/
A.EMPLID            &EMPLID
...
```

When an SQR process starts, it parses and validates all the static SQL it finds in the program, including any included SQC files, before beginning execution.[3] Hence you may have parsed cursors that are never executed because the SQL is never reached during the SQR program logic.

SQR will generate a cursor status report of the static SQL statements if it is run with the `-S` flag. SQR is not executed directly by the Process Scheduler, but via a PeopleSoft wrapper program called `PSSQR`. Additional SQR flags can be appended to the SQR command line with the `-AP` parameter. This can be configured in the Process Type definition, as shown in Figure 11-2.

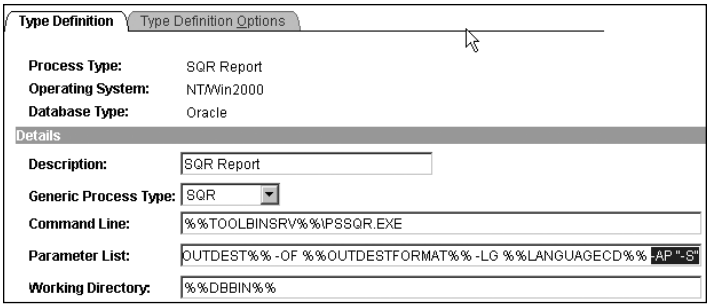


Figure 11-2. *Appending the* `-S` *parameter to the SQR command line*

The Cursor Status report (see Listing 11-28) is generated when the SQR program completes.

Listing 11-28. *Extract of* `PTSQRTST_370.out` *showing the SQR Cursor Status report*

```
Cursor Status:

 Cursor #1:
   SQL = ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY'
 Compiles = 1
 Executes = 1
 Rows     = 0
...
Cursor #34:
   SQL = SELECT substr(C.PRCSNAME, 1, 5), C.PRCSNAME  FROM PS_PRCSDEFN C
         WHERE C.PRCSTYPE like 'SQR%'
 Compiles = 2
 Executes = 1
 Rows     = 11
...
```

3. It is possible to compile a run-time version of the SQR. The queries are validated during compilation instead of execution. PeopleSoft does not use this option, and it is not recommended, because compile-time code such as `ASK` and `#IF` commands are not executed.

Application Engine

In the Application Engine, SQL is usually explicitly coded, but it can also include PeopleTools macros, called *meta-SQL*, that either dynamically construct parts of the statement or include blocks of code from elsewhere.

By default, %BIND fields are resolved to literals by the Application Engine before the SQL statements are submitted to the database. However, if the `ReuseStatement` attribute is set to `yes` on the Application Engine step, then %BIND references to columns in the state record are resolved to Oracle bind variables. In PeopleSoft-delivered applications, `ReuseStatement` is set to `no` on the vast majority of steps.

Listing 11-29 shows a SQL statement in an Application Engine step. The %BIND meta-SQL extracts a value from the column SERVERNAMERUN on the state record PRCSPURGE_AET, and %SELECT populates the column SERVERNAME on the same state record with the value selected by the query.

Listing 11-29. *PRCSYSPURGE.SchdlSrv.Step01 Do Select*

```
%Select(PRCSPURGE_AET.SERVERNAME)
 SELECT SERVERNAME
  FROM PSSERVERSTAT
 WHERE SERVERNAME <> %Bind(PRCSPURGE_AET.SERVERNAMERUN)
   AND SERVERSTATUS = '3'
   AND ( %DateTimeDiff(LASTUPDDTTM, %CurrentDateTimeIn) < 10)
```

ReuseStatement is not enabled for this set, so it produces the SQL query in Listing 11-30 (as reported by the PeopleTools trace[4]).

Listing 11-30. *PeopleTools trace of the Application Engine step PRCSYSPURGE.SchdlSrv.Step01 Do Select*

```
%Select(PRCSPURGE_AET.SERVERNAME) SELECT SERVERNAME FROM PSSERVERSTAT WHERE
SERVERNAME <> 'PSNT' AND SERVERSTATUS = '3' AND ( ROUND((( SYSDATE) -
(LASTUPDDTTM)) * 1440, 0) < 10)
/
```

From version 8, Application Engine can also execute PeopleCode. Bind variables in PeopleCode are not resolved to literals, even when executed within Application Engine programs.

Techniques for SQL Optimization

Once you have found a piece of problem SQL, there are various techniques you can employ to improve its performance. This section discusses the techniques you can apply (but without explaining either how or why they work) and then how to apply them to specific PeopleSoft Tools.

4. The `%Select` macro indicates that the column SERVERNAME is selected into the Application Engine bind variable `SERVERNAME` on the Application Engine working storage record PRCS_PURGE_AET. An SQL trace will reveal that the Application Engine actually updates the columns on the working storage record unless the restart option is disabled. This allows Application Engine processes to be restarted from the point of their last commit if they crash.

Hints

The cost-based optimizer was introduced in Oracle 7. Hints were introduced as a means to give specific instructions to the optimizer on how to execute a statement. Oracle has introduced more hints with every release.

A hint is a not just a hint, but a directive to the optimizer.[5] Most hints reduce the options open to the optimizer. The optimizer may appear to ignore the hint either when it is not properly specified or when it chooses an alternate plan that is not precluded by the hints.

In the Listing 11-31, a FULL hint has been used to force the optimizer to perform a full table scan on the PS_PERSON table, instead of using a FAST FULL SCAN on the PS_PERSON unique index.

Listing 11-31. *A FULL hint*

```
SELECT EMPLID FROM PS_PERSON A
  INDEX (FAST FULL SCAN) OF 'PS_PERSON' (UNIQUE) (Cost=2 Card=889 Bytes=5334)

SELECT /*+FULL(A)*/ EMPLID FROM PS_PERSON A
  TABLE ACCESS (FULL) OF 'PS_PERSON' (Cost=4 Card=889 Bytes=5334)
```

■ **Tip** When you add hints to code in PeopleSoft, always use the multiline `/* */` notation because the SQL query can be rewrapped by editors or within PeopleSoft.

Indexes

All index changes should be specified in the Application Designer (as described in Chapter 5), otherwise they are likely to be lost if the table is rebuilt by a PeopleTools alter script for any reason.

Disabling Indexes Without Using Hints

It is possible, by use of hints, to make a particular query either use or not use a particular index. However, it is not always possible to add the hint to certain SQL statements in PeopleSoft. An alternative strategy is to disable an index on a particular column by adjusting a SQL query but without changing its function.

Consider the query in Listing 11-32, for example.

Listing 11-32. *A query using an index*

```
SELECT EMPLID, NAME
FROM   PS_NAMES
WHERE  EMPLID >= 'PA001';
```

5. A hint must be a directive because plan stability (stored outlines) works by using a set of hints that guarantees a particular plan for a given SQL statement. See also "Oracle Urban Legends #10," presented by Connor McDonald at the UKOUG2002 Conference (www.ukoug.org), and Oracle MetaLink Document 69992.1: "Why Is My Hint Ignored?" (http://metalink.oracle.com).

```
Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE (Cost=4 Card=179 Bytes=3759)
  TABLE ACCESS (BY INDEX ROWID) OF 'PS_NAMES' (Cost=4 Card=179 Bytes=3759)
    INDEX (RANGE SCAN) OF 'PS_NAMES' (UNIQUE) (Cost=3 Card=179)

Statistics
----------------------------------------------------------
        137  consistent gets
```

This query uses the index on PS_NAMES to retrieve the data. However, this might not be desirable. If the query was changed as shown in Listing 11-33, then the index on the column EMPLID could not be used, because there the EMPLID column is part of an expression in the query.

Listing 11-33. *Index disabled*

```
SELECT EMPLID, NAME
FROM   PS_NAMES
WHERE  ''||EMPLID >= 'PA001';

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE (Cost=5 Card=179 Bytes=3759)
  TABLE ACCESS (FULL) OF 'PS_NAMES' (Cost=5 Card=179 Bytes=3759)

Statistics
----------------------------------------------------------
         87  consistent gets
```

The result is that although the full scan of the PS_NAMES table had a higher cost, it required much less logical I/O.

FROM Clause Ordering

I sometimes discover confusion among developers about how the order of tables in the FROM clause of a SQL query does—and more usually does not—affect its execution, so I'd like to take this opportunity to clarify this point.

- Under the cost-based optimizer, the order of tables in the FROM clause of a query does not affect the execution plan unless the ORDERED hint is used. Then the tables are accessed in the order specified in the FROM clause.
- In the rule-based optimizer, the tables are accessed in the *reverse* order to that specified in the FROM clause,[6] although other factors can override this.

6. The rule-based optimizer is deprecated in Oracle9*i* and not supported in Oracle 10*g*.

Here is a very simple example. The three queries shown in Listing 11-34 all produce the same execution plan. The optimizer decides to full scan the PS_NAMES table and then do an indexed lookup on PS_JOB.

Listing 11-34. *Three queries with the same execution plan*

```
SELECT A.EMPLID, A.NAME, B.DEPTID
FROM   PS_NAMES A, PS_JOB B
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';

SELECT A.EMPLID, A.NAME, B.DEPTID
FROM   PS_JOB B, PS_NAMES A
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';

SELECT /*+ ORDERED*/ A.EMPLID, A.NAME, B.DEPTID
FROM   PS_NAMES A, PS_JOB B
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';

SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=3 Bytes=99)
  TABLE ACCESS (BY INDEX ROWID) OF 'PS_JOB' (Cost=2 Card=3 Bytes=36)
    NESTED LOOPS (Cost=7 Card=3 Bytes=99)
      TABLE ACCESS (FULL) OF 'PS_NAMES' (Cost=5 Card=1 Bytes=21)
      INDEX (RANGE SCAN) OF 'PS_JOB' (UNIQUE) (Cost=1 Card=3)
```

In Listing 11-35, the order of the tables in the FROM clause has been reversed. However, the ORDERED hint forces the optimizer to start with the PS_JOB table.

Listing 11-35. *The ORDERED hint forces a different execution plan*

```
SELECT /*+ORDERED*/ A.EMPLID, A.NAME, B.DEPTID
FROM   PS_JOB B, PS_NAMES A
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';

SELECT STATEMENT Optimizer=CHOOSE (Cost=9 Card=3 Bytes=99)
  HASH JOIN (Cost=9 Card=3 Bytes=99)
    INDEX (FAST FULL SCAN) OF 'PSOJOB' (NON-UNIQUE) (Cost=3 Card=1753 Bytes=21036)
    TABLE ACCESS (FULL) OF 'PS_NAMES' (Cost=5 Card=1 Bytes=21)
```

In Listing 11-36, the execution plans under the rule-based optimizer start with whichever table is last in the WHERE clause.

Listing 11-36. *The rules-based optimizer takes the tables in the reverse order of the FROM clause.*

```
SELECT /*+RULE*/ A.EMPLID, A.NAME, B.DEPTID
FROM   PS_NAMES A, PS_JOB B
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';
```

```
SELECT STATEMENT Optimizer=HINT: RULE
  TABLE ACCESS (BY INDEX ROWID) OF 'PS_NAMES'
    NESTED LOOPS
      TABLE ACCESS (FULL) OF 'PS_JOB'
      INDEX (RANGE SCAN) OF 'PS_NAMES' (UNIQUE)

SELECT /*+RULE*/ A.EMPLID, A.NAME, B.DEPTID
FROM   PS_JOB B, PS_NAMES A
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%';

SELECT STATEMENT Optimizer=HINT: RULE
  TABLE ACCESS (BY INDEX ROWID) OF 'PS_JOB'
    NESTED LOOPS
      TABLE ACCESS (FULL) OF 'PS_NAMES'
      INDEX (RANGE SCAN) OF 'PSBJOB' (NON-UNIQUE)
```

Explicitly Coding Implicit Joins

If A = B and B = C, it implies that A = C. Mathematicians call this *transitive closure*. However, the Oracle optimizer does not work that out for itself on column predicates.[7] Explicitly adding the implicit joins permits the optimizer to consider additional join permutations without changing the function of the statement. Consequently, the optimizer tends to follow the join conditions.

The example in Listing 11-37 is taken from Global Payroll. It became a performance problem because Oracle chose an inappropriate execution plan.

Listing 11-37. *Original "problem" SQL*

```
SELECT ...
  FROM PS_GP_PYE_STAT_WRK S
      ,PS_GP_PYE_PRC_STAT P2
      ,PS_GP_RSLT_ACUM RA
 WHERE S.RUN_CNTL_ID=:1
   AND S.OPRID=:2
   AND S.EMPLID BETWEEN :3 AND :4
   AND S.EMPLID=RA.EMPLID
   AND P2.EMPLID=RA.EMPLID
   AND S.EMPL_RCD=RA.EMPL_RCD
   AND P2.EMPL_RCD=RA.EMPL_RCD
   AND S.GP_PAYGROUP=RA.GP_PAYGROUP
   AND P2.GP_PAYGROUP=RA.GP_PAYGROUP
   AND S.CAL_ID=RA.CAL_ID
   AND P2.CAL_ID=RA.CAL_ID
```

7. Oracle can do transitive closure on value predicates. If A = 1 and A = B, then adding the condition B = 1 will not make any difference because Oracle does do transitive closure on value predicates. Other databases, such as Sybase and DB2, can do transitive closure on column predicates.

```
   AND P2.CAL_RUN_ID=RA.CAL_RUN_ID
   AND P2.ORIG_CAL_RUN_ID=RA.ORIG_CAL_RUN_ID
   AND S.PRD_TYPE='R'
   AND S.RSLT_SEG_NUM=RA.RSLT_SEG_NUM
   AND S.PRIOR_VER_NUM=P2.RSLT_VER_NUM
   AND S.PRIOR_REV_NUM=P2.RSLT_REV_NUM
   AND RA.ACM_PRD_OPTN='1'
ORDER BY ...
;
```

If you draw out the SQL statement, as shown in Figure 11-3, you can see how the tables are joined. All the selective conditions are on PS_GP_PYE_STAT_WRK, so the optimizer starts with that table. PS_GP_RSLT_ACUM is one of the main payroll result tables and will usually hold many millions of rows. PS_GP_PYE_PRC_STAT is also a result table, but it is much smaller than PS_GP_RSLT_ACUM. The result table has only a single unique index, as shown in Listing 11-38.

Listing 11-38. *Index on a very large result table*

```
CREATE UNIQUE INDEX PS_GP_RSLT_ACUM ON PS_GP_RSLT_ACUM
(EMPLID, CAL_RUN_ID, EMPL_RCD, GP_PAYGROUP, CAL_ID, ORIG_CAL_RUN_ID,
RSLT_SEG_NUM, PIN_NUM, EMPL_RCD_ACUM, ACM_FROM_DT, ACM_THRU_DT, SLICE_BGN_DT,
 SLICE_END_DT,
 SEQ_NUM8) ...
```

The optimizer starts with PS_GP_PYE_STAT_WRK; follows the links on the key columns (see Figure 11-3) and accesses the very large result table PS_GP_RSLT_ACUM second, using the first five columns of the index; and then finally goes to PS_GP_PYE_STAT.

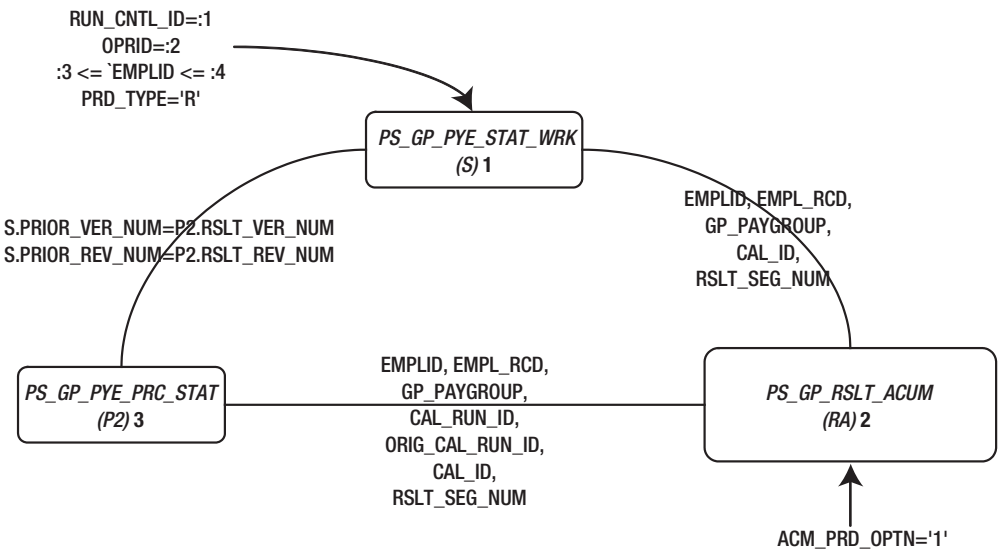


Figure 11-3. *Diagram of the original SQL*

The joins in Listing 11-39 are implied from other joins in the query.

Listing 11-39. *Implied joins*

```
AND P2.EMPLID=S.EMPLID
AND P2.EMPL_RCD=S.EMPL_RCD
AND P2.GP_PAYGROUP=S.GP_PAYGROUP
AND P2.CAL_ID=S.CAL_ID
```

By explicitly adding the implicit joins, the optimizer joins the tables in a different order, as shown in Figure 11-4.

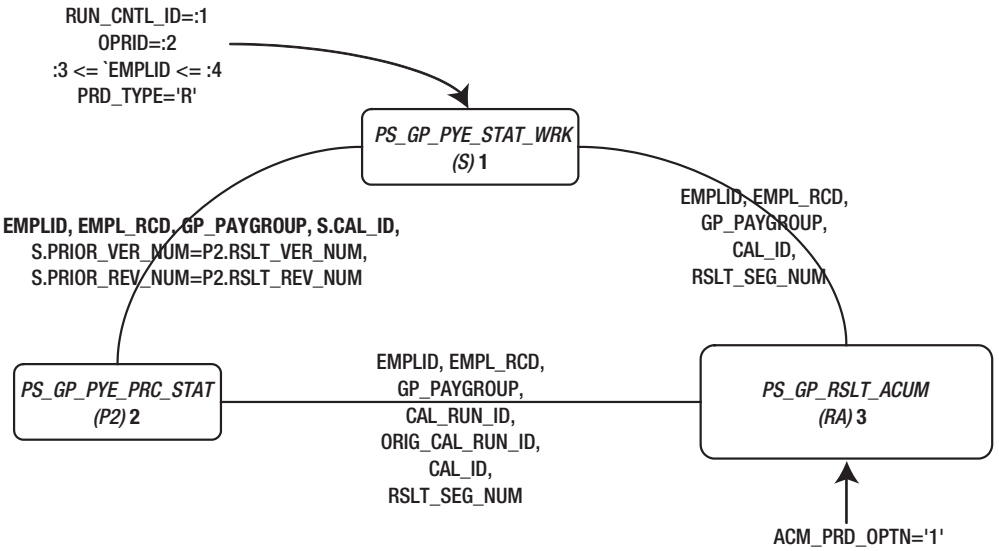


Figure 11-4. *SQL with implicit joins added*

The optimizer still starts with PS_GP_PYR_STAT_WRK, but it can now join PS_GP_PYE_PRC_STAT second, before accessing PS_GP_RSLT_ACUM using the first seven keys of the index. In this case, using the extra two columns on the index produced a dramatic improvement in performance.

■ **Tip** If you ever have trouble seeing what is going on in a query, or why the optimizer takes a particular path through the tables, draw it in the manner shown in this section. Doing so also helps to identify when implicit joins could be explicitly coded.

Plan Stability (or Stored Outlines)

Plan stability is a feature in Oracle that is designed to prevent environmental factors from affecting performance characteristics of the application. The execution plans of SQL statements are held in the database as *stored outlines*. An outline is a set of hints that unambiguously describe a particular execution plan and, if applied to the statement, will produce the same execution plan.

Stored outlines provide a way to apply hints to a SQL statement that cannot be changed in the application code. There are various scenarios in PeopleSoft in which a DBA may be tempted to consider introducing plan stability.

A SQL statement may be generated from metadata, and there is no opportunity to edit it directly. This is frequently the case in the component processor. The COBOL batch programs in the Financials product and the Global Payroll module of HCM also dynamically generate SQL.

DBAs might also be tempted to use stored outlines instead of using PeopleSoft development tools to make changes. However, I consider that plan stability is unlikely to be an effective tuning tool in a PeopleSoft environment, and there are other, better options. The dynamic nature of PeopleSoft SQL makes it difficult to use stored outlines. The SQL statement stored in the outline must exactly match the one submitted to the database for the outline to be applied.

In Chapter 8, we saw how SQL code generated by the component processor when saving application data changes in the PIA changed depending upon which columns were updated by the user. Each combination of columns would produce a different SQL statement and would require a different outline.

In Chapter 4, I showed how SQL generated by the search dialog includes user input in literal values. Earlier in this chapter, I described how variables referenced by `%BIND` in the Application Engine are also resolved to literal values. Each literal value produces a different SQL statement. If CURSOR_SHARING is set to FORCE or SIMILAR, then the literal values are converted to bind variables, and the statements are considered to be the same.[8]

While cursor sharing can be effective in reducing the amount of hard parsing in a PeopleSoft system, it can also prevent the use of histograms, although bind variable peeking, introduced in Oracle9*i,* mitigates this problem. As mentioned in Chapter 5, much of the processing in the Financials product is based on searching for flags set to a particular value, so histograms are needed to make Oracle use indexes on the flag columns. If necessary, the CURSOR_SHARING_EXACT hint, introduced in Oracle9*i*, can be applied to specific statements to prevent conversion of literals to bind variables.

I consider stored outlines to be an option of last resort when working on PeopleSoft systems. They are also an additional administrative overhead. The DBA must manage them outside the PeopleSoft development environment and migrate from database to database in synchronization with program code.

I have never been faced with a performance issue where there is no option left other than stored outlines and that is so severe it justifies the additional administrative overhead. Most of the batch processing source code is accessible either in flat files or via the Application Designer. Most of the code generated by the component processor references tables by their primary key columns and is rarely a performance problem.

Instead, I have always found that suitable indexes with realistic optimizer statistics, including histograms when necessary and occasional hints in the source code, are sufficient.

---

8. The stored outline contains bind variables if the outline is created by enabling CREATE_STORED_OUTLINES, but not if the `CREATE OUTLINE DDL` command is used.

Implementing SQL Optimization Techniques

This section discusses how to implement various optimization and other coding tips into SQL generated by PeopleTools. The examples presented here only illustrate the techniques. I am not suggesting that any or all of the changes will actually improve the performance of the specific examples.

Views

Most views in PeopleSoft are defined in the Application Designer, in the usual way, as a block of SQL. All of the standard techniques for improving a query can be brought to bear upon views. Some views are defined as a query, and have to be edited via the Query design tool, but they could always be changed to normal views.

Views can also be a useful way to introduce a hint via the back door. A reference to one record that is a table can be replaced in PeopleSoft with a different—but identical—record that is a view. That view might simply select the original table (in which case you can still update the table via the view), but now you can add a hint to the view. The scope of the hint is not limited to the view, but it can affect the query that references the view.

■ **Caution** Use this technique with care because every statement that references the view could be affected by the hint, and that may not be desirable.

Component Processor

There is no way to directly alter the SQL generated by the component processor to load and save data in the PIA. However, this SQL is rarely a problem. Most of the queries use criteria on the key columns, so the database naturally uses the unique key index. However, it is possible to change the scroll in the page to reference a view instead.

Case-Insensitive Search Dialogs and Function-Based Indexes

The SQL behind the search dialog (see Listing 11-40) is generated from the definition of the search record and the fields in the dialog that the user fills in. You cannot change the SQL. In particular, as noted earlier, you cannot remove the DISTINCT keyword.

However, for search records that are tables, indexing will usually resolve most problems. When the search record is a view, you can always change the view.

If case-insensitive searching is enabled, a function-based index on a frequently used underlying column is often helpful (also see Chapter 6).

Listing 11-40. *Search dialog SQL and a suggested function-based index*

```
SELECT DISTINCT EMPLID, EMPL_RCD, NAME, LAST_NAME_SRCH, SETID_DEPT, DEPTID,
NAME_AC, PER_STATUS
FROM PS_PERS_SRCH_GBL
WHERE ROWSECCLASS=:1
AND UPPER(NAME) LIKE UPPER('Smith') || '%' ESCAPE '\'
ORDER BY NAME, EMPLID;

CREATE INDEX PSZNAMES ON PS_NAMES
(UPPER(NAME)
) ...
/
```

Search Dialog PeopleCode

Sometimes, if you don't like the answer you should change the question. Search dialogs are sometimes slow because the user has not put in enough data to restrict the amount of data returned (PeopleTools restricts the dialog to show only the first 300 rows queried). PeopleCode could be added to the search record, to force at least some of the name to be entered.

In the vanilla HCM application, there is SearchSave PeopleCode on the search record (see Listing 11-41) that forces the operator to enter something in the search dialog; otherwise, the operator gets an error message and cannot start the search.

Listing 11-41. *Extract of PERS_SGBL_SBR.SearchSave PeopleCode*

```
If Not RecordChanged(PERS_SGBL_SBR.EMPLID) Then
   Error MsgGet(1000, 168, "At least one key field must be entered.")
End-If;
```

It is possible to tighten the search restriction a lot more to save the users from themselves. In Listing 11-42, the SearchSave PeopleCode has been enhanced to force the operators to enter at least the first two characters of the name or last name, or at least five characters of the EMPLID, unless they are also searching by one of the name fields. Forcing operators to put in better search criteria can produce better performance.

Listing 11-42. *Additional validation added to PERS_SGBL_SBR.SearchSave PeopleCode*

```
/* if searching on a name then at least two characters must be entered*/
If All(PERS_SGBL_SBR.NAME) And
      Len(PERS_SGBL_SBR.NAME) < 2 Then
   Error MsgGet(21000, 28, "Enter at least 2 characters of name");
End-If;
/* if searching on a name then at least two characters must be entered*/
If All(PERS_SGBL_SBR.LAST_NAME_SRCH) And
      Len(PERS_SGBL_SBR.LAST_NAME_SRCH) < 2 Then
   Error MsgGet(21000, 28, "Enter at least 2 characters of last name");
End-If;
```

```
/* if entry anything in EMPLID then enter at least 9 characters unless
searching on something else as well */
If All(PERS_SGBL_SBR.EMPLID) And
      Len(PERS_SGBL_SBR.EMPLID) < 5 Then
   If None(PERS_SGBL_SBR.NAME) And
         None(PERS_SGBL_SBR.LAST_NAME_SRCH) Then
      Error MsgGet(21000, 28, "Enter whole of employee ID, not just a part, if not
entering and other search information");
   End-If;
```

PeopleCode

Pieces of PeopleCode are attached to events in the PIA. When a user clicks a link or saves a piece of data, PeopleCode can be coded to fire and perform processing. SQL statements appear in PeopleCode in various forms, as described in the sections that follow.

SQLExec()

What you see is what you get. A SQL statement passed as a string to a `SQLExec()` function, as shown in Listing 11-43, will be passed to the database exactly as coded, although it can use certain PeopleSoft macros to format values.

Listing 11-43. *SQL statement in a* `SQLExec()` *PeopleCode function*

```
      SQLExec("Select A.BEN_STATUS from PS_ACTN_REASON_TBL A where A.ACTION = :1
and A.ACTION_REASON = (Select min(AA.ACTION_REASON) from PS_ACTN_REASON_TBL AA
where AA.ACTION = A.ACTION) and A.EFFDT = (Select max(AAA.EFFDT) from
PS_ACTN_REASON_TBL AAA where AAA.ACTION = A.ACTION and AAA.ACTION_REASON =
A.ACTION_REASON)", &ACTION, &FETCH_STATUS);
```

Sometimes statements are built dynamically before being submitted, as in Listing 11-44.

Listing 11-44. *Dynamically generated statement used in a* `SQLExec()` *function*

```
&SEL = "SELECT 'X' FROM PS_" | &RECN | " WHERE PSARCH_ID = :1";
SQLExec(&SEL, PSARCH_ID, &EXIST);
```

Changes to the SQL can be made by simply editing the PeopleCode.

ScrollSelect() and Other Scroll Functions

The PeopleCode scroll functions are used to programmatically load data into scrolls on a page. The `ScrollSelect()` function (see Listing 11-45) queries data from a record into a scroll. The SELECT and FROM clauses of the query are generated automatically by PeopleTools. The fourth parameter of the function is a string, which contains a WHERE clause that is appended to the query generated by the rest of the command.

Listing 11-45. *A* `ScrollSelect()` *function*

```
ScrollSelect(1, Record.ENCUMB_TRIGGER, Record.ENCUMB_TRIGGER, "Where
TRIGGER_RECORD = 'J' and emplid = :1 and EMPL_RCD = :2 and PROCESSED = 'N'",
&EMPLID, &EMPL_RCD);
```

The scroll function in Listing 11-45 generates the SQL in Listing 11-46.

Listing 11-46. *Query generated by* `ScrollSelect()`

```
Stmt=SELECT SETID, DEPTID, POSITION_POOL_ID, SETID_JOBCODE, JOBCODE,
POSITION_NBR, EMPLID, EMPL_RCD, JOB_REQ_NBR, TRIGGER_RECORD, TIME_STAMP,
TO_CHAR(TIME_STAMP,'YYYY-MM-DD-HH24.MI.SS."000000"'), PROCESSED FROM PS_ENCUMB_TRIGGER
Where TRIGGER_RECORD = 'J' and emplid = :1 and EMPL_RCD = :2 and PROCESSED = 'N'
ORDER BY SETID, DEPTID, POSITION_POOL_ID, SETID_JOBCODE, JOBCODE, POSITION_NBR,
EMPLID, EMPL_RCD, JOB_REQ_NBR, TRIGGER_RECORD, TIME_STAMP
```

You can adjust the WHERE clause of the SQL in PeopleCode, but you would need to use a view to control the SELECT or FROM clause.

Rowsets

Scroll functions are still supported in PeopleTools 8 PeopleCode, but they have been superceded by rowset objects (see Listing 11-47).

Listing 11-47. *Rowset methods*

```
&Table1_vw_rs = CreateRowset(Record.PTP_TABLE1_VW);
&Table1_vw_rs.Fill("WHERE PTP_SEQ_NBR >= :1", &nbr);
&Rs = GetRowset(Scroll.PTP_TABLE1);
&Rs.Flush();
&Rs.Select(Record.PTP_TABLE1, "WHERE PTP_SEQ_NBR <= 10010");
```

The SQL generated by the `Fill()` and `Select()` methods (see Listing 11-48) is similar to that generated by the scroll functions, but it can be distinguished because the table always has the alias "FILL."

Listing 11-48. *SQL generated by rowset methods*

```
PSAPPSRV.3808   1-11990  14.28.59    0.471 Cur#1.3808.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
FILL.PTP_SEQ_NBR,FILL.PTP_SEQ_CHAR,FILL.DESCR,FILL.PTP_INT01,FILL.PTP_INT02,FILL.PTP_INT03
,FILL.PTP_INT04,FILL.PTP_INT05,FILL.PTP_INT06,FILL.PTP_INT07,FILL.PTP_INT08,FILL.PTP_INT09
,FILL.PTP_INT10,FILL.PTP_INT11,FILL.PTP_INT12,FILL.PTP_INT13,FILL.PTP_INT14,FILL.PTP_INT15
,FILL.PTP_INT16,FILL.PTP_INT17,FILL.PTP_INT18,FILL.PTP_INT19,FILL.PTP_INT20,FILL.PTP_INT21
,FILL.PTP_INT22,FILL.PTP_INT23,FILL.PTP_INT24,FILL.PTP_CHAR01,FILL.PTP_CHAR02,
FILL.PTP_CHAR03,FILL.PTP_CHAR04,FILL.PTP_CHAR05,FILL.PTP_CHAR06,FILL.PTP_CHAR07,
FILL.PTP_CHAR08,FILL.PTP_CHAR09,FILL.PTP_CHAR10,FILL.PTP_CHAR11,FILL.PTP_CHAR12,

FILL.PTP_CHAR13,FILL.PTP_CHAR14,FILL.PTP_CHAR15,FILL.PTP_CHAR16,FILL.PTP_CHAR17,
FILL.PTP_CHAR18,FILL.PTP_CHAR19,FILL.PTP_CHAR20,FILL.PTP_CHAR21,FILL.PTP_CHAR22,
FILL.PTP_CHAR23 FROM PS_PTP_TABLE1 FILL  WHERE PTP_SEQ_NBR <= 10001
PSAPPSRV.3808   1-11993  14.28.59    0.000 Cur#1.3808.HR88 RC=0 Dur=0.000 COM Stmt=SELECT
FILL.PTP_SEQ_NBR,FILL.PTP_SEQ_CHAR,FILL.DESCR,FILL.PTP_INT01 FROM PS_PTP_TABLE1_VW FILL
WHERE PTP_SEQ_NBR >= :1
```

Query

The monolithic nature of the SQL generated by PeopleSoft Query not only produces queries that perform poorly, but is also one the main reasons that queries can become a cause of systemwide performance problems.

It is possible to introduce Oracle hints in queries to control execution plans. Outer joins can avoid more complicated constructions. However, other, more esoteric features of SQL are not supported by Query because they are not implemented on all platforms. In such cases, the only option is to create a view as desired and query the view.

Hints in Expressions

For a hint to be recognized in a query, it must appear between the SELECT keyword and the first selected column (see Listing 11-49).

Listing 11-49. *A PeopleSoft query with a hint*

```
SELECT /*+ ALL_ROWS*/  A.EMPLID, A.COURSE, A.SESSION_NBR, D.NAME, B.DESCR,
TO_CHAR(A.COURSE_START_DT,'YYYY-MM-DD'), A.ATTENDANCE,B.COURSE
  FROM PS_TRAINING A, PS_COURSE_TBL B, PS_PERSONAL_DTA_VW D, PS_PERS_SRCH_QRY D1
  WHERE D.EMPLID = D1.EMPLID
    AND D1.ROWSECCLASS = 'HCDPALL'
    AND ( ( A.ATTENDANCE = 'E'
     AND A.COURSE = B.COURSE
     AND D.EMPLID = A.EMPLID
     AND A.COURSE = :1
     AND ( A.SESSION_NBR = :2
     OR A.COURSE_START_DT = TO_DATE(:3,'YYYY-MM-DD'))) )
```

To achieve this in Query, you need to create an expression, as shown in Figure 11-5, defined the same way as the first selected field that it will replace.

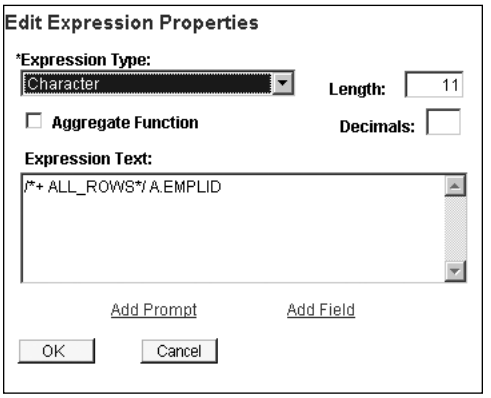


Figure 11-5. *A Query expression*

Then make it the first selected field, as shown in Figure 11-6.

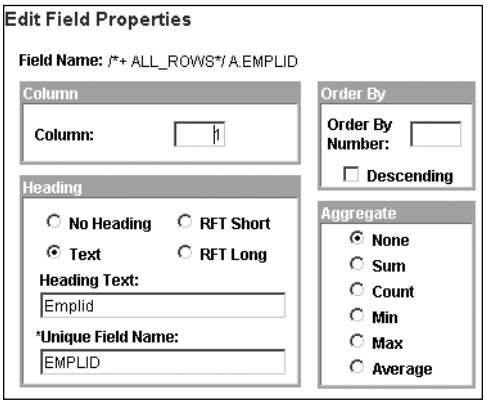


Figure 11-6. *Edit Field Properties panel*

This technique does not work if the query has been made DISTINCT (see Figure 11-7).

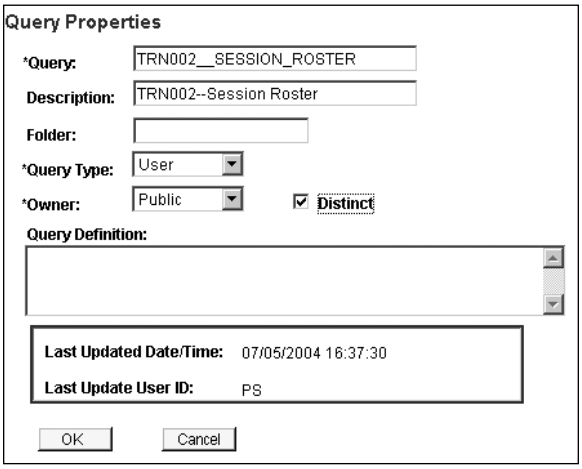


Figure 11-7. *Query Properties panel*

The hint will appear after the DISTINCT keyword, and the optimizer will ignore it because it is in the wrong place, as shown in Listing 11-50.

Listing 11-50. *A hint in the wrong place is just a comment*

```
SELECT DISTINCT /*+ ALL_ROWS*/  A.EMPLID, A.COURSE, ...
```

Instead, the Distinct check box should be unchecked and the DISTINCT keyword put in the expression. The result is shown in Listing 11-51.

Listing 11-51. *A hint in the correct position*

```
SELECT /*+ ALL_ROWS*/ DISTINCT  A.EMPLID, A.COURSE, ...
```

Hints in Views

Cost-based optimizer hints can also be enabled in a particular query by adding a view that contains an optimizer hint. The view in Listing 11-52 can be defined in the Application Designer (remember to put it on a Query tree).

Listing 11-52. *A hint in a view*

```
CREATE VIEW PS_FIRST_ROWS_VW (DUMMY_FIELD) AS
SELECT /*+ FIRST_ROWS */
 'x' FROM DUAL
```

Then the view can be joined into the query, as shown in Listing 11-53, thus introducing the hint.

Listing 11-53. *The view with a hint in a query*

```
SELECT A.EMPLID, A.COURSE, A.SESSION_NBR, D.NAME, B.DESCR,
TO_CHAR(A.COURSE_START_DT,'YYYY-MM-DD'), A.ATTENDANCE,B.COURSE
  FROM PS_TRAINING A, PS_COURSE_TBL B, PS_PERSONAL_DTA_VW D,
     PS_PERS_SRCH_QRY D1, PS_FIRST_ROWS_VW C
  WHERE D.EMPLID = D1.EMPLID
    AND D1.ROWSECCLASS = 'HCDPALL'
    AND ( ( A.ATTENDANCE = 'E'
     AND A.COURSE = B.COURSE
     AND D.EMPLID = A.EMPLID
     AND A.COURSE = :1
     AND ( A.SESSION_NBR = :2
     OR A.COURSE_START_DT = TO_DATE(:3,'YYYY-MM-DD'))) )
```

On systems where some operators are allowed to develop their own queries, this provides a relatively safe and easy way for users to add hints to views.

Outer Joins

There are genuinely occasions when it is necessary for an operator to code an outer join in Query. If it were not possible, the only alternative would be convoluted UNION queries such as the one shown in Listing 11-54.[9]

Listing 11-54. *Query functionally equivalent to an outer join*

```
SELECT A.EMPLID, A.NAME, B.COUNTRY, B.CITY, B.COUNTY
FROM   PS_PERSONAL_DATA A, PS_PERS_SRCH_QRY A1, PS_ADDRESSES B
WHERE  A.EMPLID = A1.EMPLID
```

9. This is exactly the kind of thing that PeopleTools does when an operator's language is not the base language for the system. If the translation in the operator's language is not available, PeopleTools retrieves the base language value.

```
AND    A1.ROWSECCLASS = 'DPALL'
AND A.EMPLID = B.EMPLID
AND B.EFFDT =
   (SELECT MAX(B_ED.EFFDT) FROM PS_ADDRESSES B_ED
   WHERE B.EMPLID = B_ED.EMPLID
   AND B.ADDRESS_TYPE = B_ED.ADDRESS_TYPE
   AND B_ED.EFFDT <= SYSDATE))
UNION
SELECT C.EMPLID, C.NAME, ' ', ' ', ' '
FROM   PS_PERSONAL_DATA C, PS_PERS_SRCH_QRY C1
WHERE  C.EMPLID = C1.EMPLID
AND    C1.ROWSECCLASS = 'DPALL'
AND NOT EXISTS(
   SELECT 'x' FROM PS_ADDRESSES D
   WHERE  C.EMPLID = D.EMPLID
   AND    D.EFFDT =
      (SELECT MAX(D_ED.EFFDT) FROM PS_ADDRESSES D_ED
      WHERE D.EMPLID = D_ED.EMPLID
      AND D.ADDRESS_TYPE = D_ED.ADDRESS_TYPE
      AND D_ED.EFFDT <= SYSDATE)))
```

PeopleTools 8.44

From PeopleTools 8.44, Query will generate the SQL92-compliant[10] outer join syntax that Oracle implemented, and now recommends,[11] in release 9*i*.

■ **Note** This syntax is not recognized by Oracle8*i*.

If you outer join to an effective dated table (in this example, PS_ADDRESSES), then Query will automatically add the effective date subquery. You cannot outer join to a subquery. If the effective date criterion is appropriate, then you need to add the `OR B.EFFDT IS NULL` criterion manually, as shown in Listing 11-55.

Listing 11-55. *Outer joining an effective dated table*

```
SELECT A.EMPLID, A.NAME, B.COUNTRY, B.CITY, B.COUNTY
  FROM (PS_NAMES A LEFT OUTER JOIN  PS_ADDRESSES B ON  A.EMPLID = B.EMPLID ),
PS_PERS_SRCH_QRY A1
  WHERE A.EMPLID = A1.EMPLID
    AND A1.ROWSECCLASS = 'HCDPALL'
```

10. When vendors claim that they are "SQL92 compliant," it is important to know at which level. Oracle 7.0 was certified for SQL92 entry-level compliance in 1993. The left outer join syntax is in a transitional level. There are also intermediate and full levels.
11. According to the *Oracle9*i *SQL Reference* (Release 2), "Oracle Corporation recommends that you use the FROM clause OUTER JOIN syntax rather than the Oracle join operator."

```
AND ( A.EFFDT =
    (SELECT MAX(A_ED.EFFDT) FROM PS_NAMES A_ED
    WHERE A.EMPLID = A_ED.EMPLID
      AND A.NAME_TYPE = A_ED.NAME_TYPE
      AND A_ED.EFFDT <= SYSDATE)
 AND A.EMPLID LIKE 'KUZ%'
 AND ( B.EFFDT =
    (SELECT MAX(B_ED.EFFDT) FROM PS_ADDRESSES B_ED
    WHERE B.EMPLID = B_ED.EMPLID
      AND B.ADDRESS_TYPE = B_ED.ADDRESS_TYPE
      AND B_ED.EFFDT <= A.EFFDT)
 OR B.EFFDT IS NULL) )
```

Figure 11-8 shows the output of this query in Excel.

	A	B	C	D	E
1		4			
2	ID	Name	Country	City	County
3	KUZ010	Lamoreaux,Nathalie Eve	USA	Rochester	
4	KUZ020	Pierce,Suzanne Marie	USA	Oklahoma City	
5	KUZ100	Alvarez,Angelica			
6	KUZ110	Knoelle,Ken			

Figure 11-8. *Results of Listing 11-55 in Excel*

PeopleTools 8.43 and Earlier

Up to PeopleTools 8.43, PeopleSoft Query does not generate any form of outer-join syntax. The only way to introduce it is to manually code the Oracle-specific syntax in an expression, as shown in Figure 11-9.

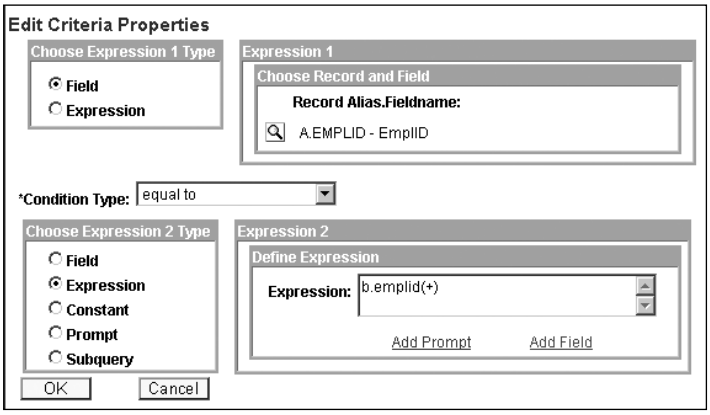


Figure 11-9. *PeopleTools 8.1x Query criteria properties with Oracle outer-join syntax*

The SQL in Listing 11-56 was generated by Query. It is still necessary to work around the effective date subquery on the outer-joined table by adding the OR ... IS NULL condition manually.

Listing 11-56. *A query with Oracle-specific outer-join syntax*

```
SELECT A.EMPLID, A.NAME, B.COUNTRY, B.CITY, B.COUNTY
FROM PS_NAMES A, PS_PERS_SRCH_QRY A1, PS_ADDRESSES B
WHERE A.EMPLID = A1.EMPLID
AND A1.ROWSECCLASS = 'DPALL'
AND ( A.EFFDT =
(SELECT MAX(A_ED.EFFDT) FROM PS_NAMES A_ED
WHERE A.EMPLID = A_ED.EMPLID
AND A.NAME_TYPE = A_ED.NAME_TYPE
AND A_ED.EFFDT <= SYSDATE)
AND A.EMPLID = B.EMPLID(+)
AND ( B.EFFDT =
(SELECT MAX(B_ED.EFFDT) FROM PS_ADDRESSES B_ED
WHERE B.EMPLID = B_ED.EMPLID
AND B.ADDRESS_TYPE = B_ED.ADDRESS_TYPE
AND B_ED.EFFDT <= SYSDATE)
OR B.EFFDT IS NULL)
AND A.EMPLID LIKE 'L%' )
```

Outer-Joining Query Security Records

The query security record is an attribute of the record defined in the Application Designer. Whenever a record is referenced in Query, it is automatically joined to its query security record, which is also added to the query. If records in a query have the same query security record, the query security record only appears once in the FROM clause of the query.

The newly implemented SQL92 syntax in PeopleTools 8.44 makes it possible to outer join a record with a query security record. In the example in Listing 11-57, both tables are secured by PS_EMPLMT_SRCH_QRY. PeopleTools automatically adds `OR B.EMPLID IS NULL` to the join to the query security record. The (+) syntax is Oracle specific and so is not used by PeopleTools.

Listing 11-57. *SQL92 outer-join syntax*

```
SELECT A.EMPLID, A.EMPL_RCD, TO_CHAR(A.EFFDT,'YYYY-MM-DD'), A.EFFSEQ, B.NAME
  FROM (PS_JOB A LEFT OUTER JOIN  PS_EMPLOYEES B
        ON  A.EMPLID = B.EMPLID AND A.EMPL_RCD = B.EMPL_RCD ), PS_EMPLMT_SRCH_QRY A1
  WHERE A.EMPLID = A1.EMPLID
    AND A.EMPL_RCD = A1.EMPL_RCD
    AND A1.ROWSECCLASS = 'HCDPALL'
    AND  (B.EMPLID = A1.EMPLID OR B.EMPLID  IS NULL )
    AND  (B.EMPL_RCD = A1.EMPL_RCD OR B.EMPL_RCD  IS NULL )
...
```

This would not have been possible up to PeopleTools 8.43, where the equi-join to the query security table could not be removed.

Listing 11-58. *System-generated equi-joins*

```
SELECT A.EMPLID, A.EMPL_RCD, TO_CHAR(A.EFFDT,'YYYY-MM-DD'), A.EFFSEQ
  FROM PS_JOB A, PS_EMPLMT_SRCH_QRY A1, PS_EMPLOYEES B
  WHERE A.EMPLID = A1.EMPLID
```

```
    AND A.EMPL_RCD = A1.EMPL_RCD
    AND A1.ROWSECCLASS = 'HCDPALL'
    AND B.EMPLID = A1.EMPLID
    AND B.EMPL_RCD = A1.EMPL_RCD
...
```

The workaround for this is to outer join the two tables in a view, apply the security record to the view, and then use the view in Query.

FROM Clause Ordering

As discussed earlier, the ORDERED hint forces Oracle to access tables in the order in which they appear in the FROM clause. This hint is often very effective. Simply by ensuring the tables are joined in the desired sequence, the optimizer can safely be left to work out the most efficient access methods.

However, the only *supported* way to control the order of the FROM clause in a query is to add tables to the query in that order. You cannot control the position of security records. They are placed immediately after the first table in the FROM clause to which they are joined.

LEADING Hint

The LEADING hint is an alternative to the ORDERED hint, where it is not possible to change the order of the FROM clause as desired. This hint was introduced in Oracle9*i*, and it lets you specify the first table that Oracle should access to execute the query, as demonstrated in Listing 11-59.

Listing 11-59. *The LEADING hint*

```
SELECT A.EMPLID, A.NAME, B.DEPTID
FROM   PS_JOB B, PS_NAMES A
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%'
/
Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=3 Bytes=99)
 TABLE ACCESS (BY INDEX ROWID) OF 'PS_JOB' (Cost=2 Card=3 Bytes=36)
  NESTED LOOPS (Cost=7 Card=3 Bytes=99)
   TABLE ACCESS (FULL) OF 'PS_NAMES' (Cost=5 Card=1 Bytes=21)
   INDEX (RANGE SCAN) OF 'PS_JOB' (UNIQUE) (Cost=1 Card=3)

SELECT /*+LEADING(B)*/ A.EMPLID, A.NAME, B.DEPTID
FROM   PS_JOB B, PS_NAMES A
WHERE  A.EMPLID = B.EMPLID
AND    A.NAME LIKE 'Smith%'
/
```

```
Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE (Cost=10 Card=3 Bytes=99)
 HASH JOIN (Cost=10 Card=3 Bytes=99)
  INDEX (FAST FULL SCAN) OF 'PSAJOB' (NON-UNIQUE) (Cost=4 Card=1753 Bytes=21036)
   TABLE ACCESS (FULL) OF 'PS_NAMES' (Cost=5 Card=1 Bytes=21)
```

It is not until Oracle 10*g* that the LEADING hint can take several tables and specify the order in which the tables are accessed in a query. Until then, you may still need to control the order of the FROM clause in a query, and use the ORDERED hint.

Manually Resequencing the FROM Clause

For a fairly simple ad hoc query, the overhead of rewriting it in Query and selecting the tables in the correct order is not acceptable. However, for complex queries, particularly where there is a corresponding Crystal Report, this could become a very time-consuming process.

For such cases, and as an absolutely last resort, there is an *unsupported* way to resequence the FROM clause without having to rebuild the whole query.

■ **Caution** The following technique for reordering the FROM clause in a query is completely unsupported by both PeopleSoft Inc. and Go-Faster Consultancy Ltd. It must be stressed that if a mistake is made during the following procedure, *any or all queries in the system could be corrupted*. If you attempt it, you are strongly recommended to work on a separate copy database and take a backup of all of the PSQRY% tables before starting. The adjusted queries can then be moved to other databases with an Application Designer upgrade project.

Remember, if it goes wrong, you are on your own!

In the example in Listing 11-60, I want to introduce an ORDERED hint and reverse the current order of the FROM clause.

Listing 11-60. *A sample query*

```
SELECT A.PORTAL_LABEL, A.PORTAL_OBJNAME,A.PORTAL_NAME,A.PORTAL_REFTYPE
  FROM PSPRSMDEFN A, PSPRSMPERM B, PSOPRDEFN C
  WHERE A.PORTAL_REFTYPE = 'C'
     AND A.PORTAL_CREF_USGT = 'TARG'
     AND A.PORTAL_NAME = B.PORTAL_NAME
     AND A.PORTAL_REFTYPE = B.PORTAL_REFTYPE
     AND A.PORTAL_OBJNAME = B.PORTAL_OBJNAME
     AND C.OPRCLASS = B.PORTAL_PERMNAME
     AND A.PORTAL_URI_SEG1 <> ' '
     AND A.PORTAL_URI_SEG2 <> ' '
     AND A.PORTAL_URI_SEG3 <> ' '
     AND C.OPRID = :1
     AND A.PORTAL_NAME = :2
  ORDER BY 1
```

The first stage (see Listing 11-61) is to identify the queries to be altered. Using a variation of the method of identifying a query discussed earlier, the table GFC_UPDQRY is created to hold a list of the queries that are to be updated by the rest of this procedure.

Often there are groups of queries that reference the same tables with the same aliases to which the same changes need to be made. Users sometimes make private copies of public queries and then adjust them. This query will find all similar queries. You should review the queries found.

Listing 11-61. *Beginning of* `qryupd.sql`

```
REM qryupd.sql
REM (c)Go-Faster Consultancy Ltd. 2004
REM ------------------------------------------------------------------------
REM STOP! DO NOT RUN THIS SCRIPT UNTIL YOU HAVE READ
REM CHAPTER 11 OF PEOPLESOFT FOR THE ORACLE DBA
REM THIS SCRIPT IS COMPLETELY UNSUPPORTED
REM IT COULD CORRUPT EVERY QUERY IN YOUR PEOPLESOFT DATABASE
REM ------------------------------------------------------------------------
REM review this script before you run it
REM you will need to edit literal values and adjust some statements.

ROLLBACK;
DROP TABLE gfc_qryupd;
DROP TABLE gfc_qryupdrec;
DROP TABLE gfc_qryupdrec2;

REM ------------------------------------------------------------------------
REM *** specify the records and correlation names                        ***
REM *** add another psqryrecord for each record/correlation name        ***
REM *** combination that you are looking for                            ***

CREATE TABLE gfc_qryupd AS
SELECT /*+ORDERED*/
       r1.oprid, r1.qryname
FROM   psqryrecord r1
,      psqryrecord r2
,      psqryrecord r3
WHERE  r1.recname = 'PSPRSMDEFN'
AND    r1.corrname  = 'A'
AND    r2.corrname  = 'B'
AND    r2.recname = 'PSPRSMPERM'
AND    r2.qryname = r1.qryname
AND    r2.oprid   = r1.oprid
AND    r3.corrname  = 'C'
AND    r3.recname = 'PSOPRDEFN'
AND    r3.qryname = r1.qryname
AND    r3.oprid   = r1.oprid
;
```

Four public queries (see Listing 11-62) are identified by this query on a vanilla HCM8.8 database.

Listing 11-62. *Public queries identified for reordering*

```
OPRID                          QRYNAME
------------------------------ ------------------------------
                               PT_SEC_USER_CREF
                               PT_SEC_USER_CREF_MENU_CMP_MKT
                               PT_SEC_USER_CREF_PORTAL
                               PT_SEC_USER_CRF_PRT_MN_CMP_MKT
```

If a query has been selected that you don't want to change, then simply delete it from GFC_QRYUPD.

The next statement (see Listing 11-63) will delete any query from the list where a field exists in the query on a record that is not in the query. Theoretically, this should never happen, but I have encountered this corruption.

Listing 11-63. *Checking for a corrupt query*

```
REM ------------------------------------------------------------------------
REM omit corrupt queries

DELETE FROM gfc_qryupd d
WHERE (d.oprid, d.qryname) IN (
      SELECT f.oprid, f.qryname
      FROM   psqryfield f
      WHERE  (f.oprid, f.qryname) IN
             (SELECT oprid, qryname
             FROM   gfc_qryupd)
      AND    NOT EXISTS(
             SELECT 'x'
             FROM   psqryrecord r
             WHERE  r.qryname = f.qryname
             AND    r.oprid = f.oprid
             AND    r.recname = f.recname)
             AND   (f.recname != ' '
             OR     f.fieldname != ' ')
             );
```

PSQRYRECORD contains one row per record per query. We need a copy of parts of it for the queries to be adjusted so that both the new and old record numbers are known (see Listing 11-64).

Listing 11-64. *Copying part of a query definition to a working storage table*

```
REM ------------------------------------------------------------------------
REM create working storage table with queries to be updated

CREATE TABLE gfc_qryupdrec as
SELECT oprid, qryname, corrname, recname, selnum
,      rcdnum oldrcdnum, rcdnum newrcdnum
FROM   psqryrecord
WHERE  (oprid, qryname) IN (
       SELECT oprid, qryname
       FROM   gfc_qryupd)
;

CREATE UNIQUE INDEX gfc_qryupdrec
ON gfc_qryupdrec(oprid, qryname, selnum, oldrcdnum, recname)
;
```

Within a query, the table alias (stored on CORRNAME) uniquely identifies a record. The record name alone is not good enough for this purpose, because a record could be referenced twice. The record number (RCDNUM) controls the order in which the records appear in the FROM clause of the query. The statement in Listing 11-65 sets the new record number for each record in the query according to its alias. You will need to change the following UPDATE statement depending on the change in sequence number you want to achieve.

Listing 11-65. *Setting the new FROM clause order*

```
REM ------------------------------------------------------------------------
REM *** apply new order using correlation names to id records          ***
REM *** specify the desired order of your query here                   ***
UPDATE gfc_qryupdrec
SET newrcdnum = DECODE(corrname,
                'A',3,
                'B',2,
                'C',1)
WHERE  (oprid, qryname) IN (
       SELECT oprid, qryname
       FROM   gfc_qryupd)
;
```

Be careful when determining a new selection number. If a query is composed of a number of queries UNIONed together, and contains subqueries, then each query is given a separate selection number (SELNUM). There is a separate record number sequence, beginning with 1, for each selection number. You may need to examine the content of GFC_QRYUPDREC to work out the new record numbers.

However, there is only ever one sequence of table aliases. A second working storage table is used (because you can't use the `rank()` function in an UPDATE statement) to produce a unique number for each table in the whole query (see Listing 11-66).

Listing 11-66. *Calculating new record numbers, handling subqueries and UNIONs*

```
REM -----------------------------------------------------------------------
REM if a query has a sequence of record numbers per select, but only one
REM sequence of table aliases. Therefore use rank() to produce a list

CREATE TABLE gfc_qryupdrec2 AS
SELECT oprid, qryname, corrname, recname, selnum, oldrcdnum, newrcdnum
,      RANK() OVER (PARTITION BY oprid, qryname
                    ORDER     BY selnum, newrcdnum) AS newrank
FROM   gfc_qryupdrec
;

CREATE UNIQUE INDEX gfc_qryupdrec2
ON gfc_qryupdrec2(oprid, qryname, selnum, oldrcdnum, recname)
;
```

The new record numbers must then be written to both PSQRYRECORD and PSQRYFIELD (see Listing 11-67). New table aliases are created using the rank saved to the second temporary table.

Listing 11-67. *Updating the record number on PSQRYRECORD and PSQRYFIELD*

```
REM -----------------------------------------------------------------------
REM apply new order to record definitions, simultaneously set new table
REM alias. If more than 12 tables extend list in decode statement

UPDATE psqryrecord r
SET    (rcdnum, corrname) = (
       SELECT l.newrcdnum
       ,      DECODE(newrank,1,'A',2,'B',3,'C',4,'D',5,'E',6,'F'
                            ,7,'G',8,'H',9,'I',10,'J',11,'K',12,'L')
       FROM   gfc_qryupdrec2 l
       WHERE  l.oprid = r.oprid
       AND    l.qryname = r.qryname
       AND    l.recname = r.recname
       AND    l.selnum = r.selnum
       AND    l.oldrcdnum = r.rcdnum)
WHERE  (oprid, qryname) IN (
       SELECT oprid, qryname
       FROM   gfc_qryupd)
AND    r.recname != ' '
;

REM -----------------------------------------------------------------------
REM apply new order to field definitions
```

```
UPDATE psqryfield r
SET    fldrcdnum = (
       SELECT l.newrcdnum
       FROM   gfc_qryupdrec2 l
       WHERE  l.oprid = r.oprid
       AND    l.qryname = r.qryname
       AND    l.recname = r.recname
       AND    l.selnum = r.selnum
       AND    l.oldrcdnum = r.fldrcdnum)
WHERE  (oprid, qryname) IN (
       SELECT oprid, qryname
       FROM   gfc_qryupd)
AND    r.recname != ' '
;
```

Queries are version numbered objects. In order to force PeopleTools to refresh query definitions in the cache, the version number must be updated (see Listing 11-68).

Listing 11-68. *Incrementing PeopleSoft cache version numbers*

```
REM ------------------------------------------------------------------------
REM reset versions so that caches are updated
UPDATE pslock
SET    version=version+1
WHERE  objecttypename IN('QDM','SYS')
;

UPDATE psversion
SET    version=version+1
WHERE  objecttypename IN('QDM','SYS')
;

UPDATE psqrydefn
SET    version = (
       SELECT version
       FROM   pslock
       WHERE  objecttypename = 'QDM')
WHERE  (oprid, qryname) IN (
       SELECT oprid, qryname
       FROM   gfc_qryupd)
;
```

Only after you have reached the end of the process should all of these updates to the PeopleTools table be either committed or rolled back as a single transaction (see Listing 11-69). It is, of course, impossible to test the change without committing.

Listing 11-69. *Committing the changes*

```
REM ------------------------------------------------------------------------
REM if and only if you are satisfied with the results commit the updates
REM COMMIT;
REM DROP TABLE gfc_qryupd;
REM DROP TABLE gfc_qryupdrec;
REM DROP TABLE gfc_qryupdrec2;
```

The code in this section is supplied in `updqry.sql`.

Effective Date and Sequence Processing

This is an appropriate place to talk about a particular behavior of the Query designer that can be a cause of performance problems.

When you add a record to a query that contains the column EFFDT (effective date) or EFFSEQ (effective sequence), Query automatically adds effective date and sequence criteria to the query. To be fair, it does warn you that it is doing this, as shown in Figure 11-10.

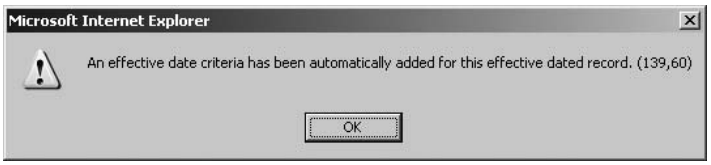


Figure 11-10. *Effective date warning*

In this example, I have used the table PS_EMPLOYEES, which has both EFFDT and EFFSEQ columns. The criterion that was automatically added can be seen in the Query Manager (see Figure 11-11).

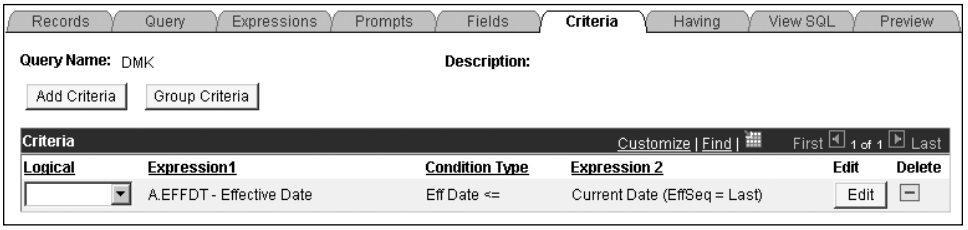


Figure 11-11. *Effective date criteria in the Query Manager*

The resulting SQL, shown in Listing 11-70, includes two subqueries.

Listing 11-70. *Query SQL with effective-date and effective-sequence subqueries*

```
SELECT A.EMPLID, A.EMPL_RCD, A.NAME
  FROM PS_EMPLOYEES A, PS_EMPLMT_SRCH_QRY A1
  WHERE A.EMPLID = A1.EMPLID
    AND A.EMPL_RCD = A1.EMPL_RCD
    AND A1.ROWSECCLASS = 'HCDPALL'
    AND ( A.EFFDT =
        (SELECT MAX(A_ED.EFFDT) FROM PS_EMPLOYEES A_ED
        WHERE A.EMPLID = A_ED.EMPLID
          AND A.EMPL_RCD = A_ED.EMPL_RCD
          AND A_ED.EFFDT <= SYSDATE)
    AND A.EFFSEQ =
        (SELECT MAX(A_ES.EFFSEQ) FROM PS_EMPLOYEES A_ES
        WHERE A.EMPLID = A_ES.EMPLID
          AND A.EMPL_RCD = A_ES.EMPL_RCD
          AND A.EFFDT = A_ES.EFFDT) )
```

However, the record is not effective-dated. There is only one row for each EMPLID/EMPL_RCD value. The key columns are shown in the Query designer (see Figure 11-12).

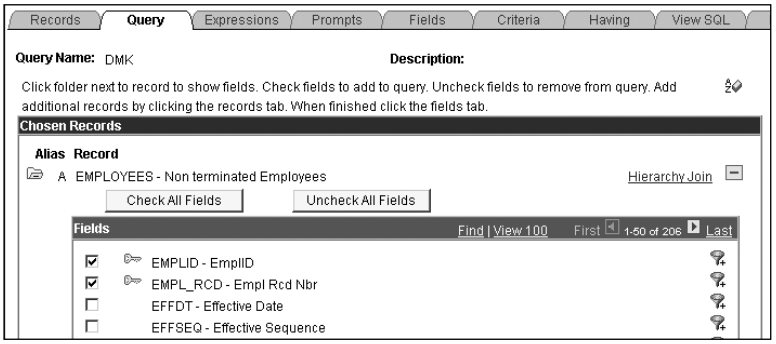


Figure 11-12. *Key fields are indicated in the Query designer*

So the subqueries cannot possibly change the result of the query,[12] but they do create additional work. There are two additional accesses of the unique index (because the subqueries are joined on the key columns) and two additional accesses of the table (because the EFFDT and EFFSEQ columns are not in the index, as they are not key fields). The additional accesses are likely to require physical as well as consistent reads, so the unnecessary effective date/sequence criteria has generated unnecessary physical I/O. This will degrade the performance of the query, and if there is extensive use of Query and Crystal Reports, it could also impact the performance of the whole system.

12. It is my opinion that this is a bug in the Query design tool and that the criteria should be automatically applied only to key fields.

■ **Tip** The most efficient way to do anything is not to do it in the first place. You should remove automatically added effective date/sequence criteria on non-key fields.

Application Engine

SQL statements are coded explicitly within the Application Engine, so all the usual methods can be applied directly. It is simply a matter of editing the step as necessary.

The Application Engine will also support platform-specific sections. PeopleSoft uses this feature to deliver different code on different database platforms, although the code is always functionally equivalent.

In the example in Figure 11-13, you can see that a different version of a step has been coded in an Application Engine program for each database platform.

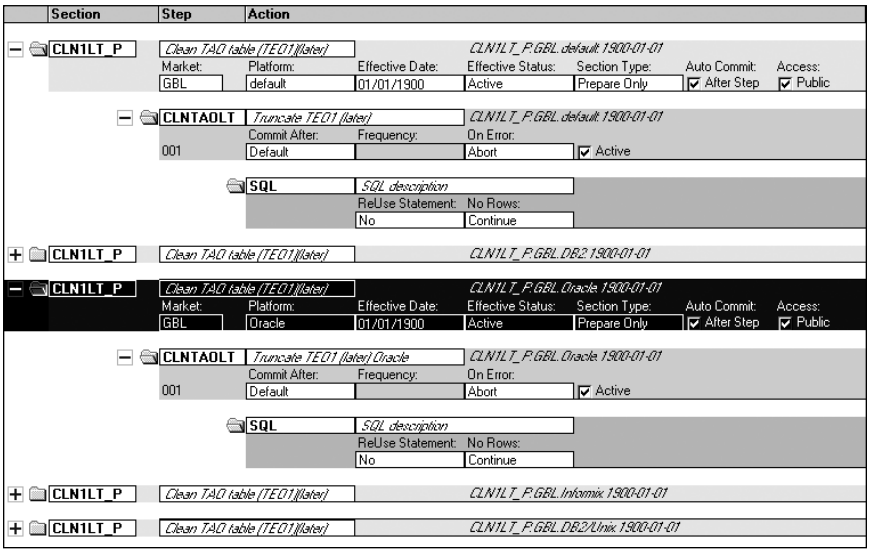


Figure 11-13. *Database-specific Application Engine sections*

The step does the same thing on all platforms. It removes all the data from a temporary working storage table. Table 11-2 shows the code in the Application Engine steps for each database platform. Oracle and SQL Server support the `TRUNCATE TABLE` command. The developer has chosen to use the `REUSE STORAGE` option on Oracle to retain the extents allocated to the table.[13] The `TRUNCATE` command does not exist on DB2 and Informix, so data simply has to be deleted from the temporary table.

13. If you introduce a Global Temporary Table into a process, make sure that it does not attempt to truncate the table with the `REUSE STORAGE` clause. The command does not raise any error, but it does not truncate the table either.

Table 11-2. *Application Engine* EO_CURRENCY.CLN1LT_P.CLNTAOLT *SQL*

Database	SQL
Default	%Execute() TRUNCATE TABLE PS_%Bind(TEO1_RECNAME,NOQUOTES)
DB2	DELETE FROM PS_%Bind(TEO1_RECNAME,NOQUOTES)
Oracle	%Execute() TRUNCATE TABLE PS_%Bind(TEO1_RECNAME,NOQUOTES) REUSE STORAGE;
Informix	DELETE FROM PS_%Bind(TEO1_RECNAME,NOQUOTES)

PeopleSoft develops its applications on Microsoft SQL Server. Where PeopleSoft does deliver database-specific sections, as in this example, the default section is SQL Server–specific.

If you choose to introduce an Oracle-specific change to a PeopleSoft-delivered Application Engine process, I recommend that you only make the change in the platform-specific section that matches your platform, if it already exists. Otherwise, do not create a platform-specific copy, but work on the default section. That way, when PeopleSoft delivers a patch or upgrade to the process, you can see the differences in the compare report, rather than have to manually compare the default and platform-specific steps.

Stored Statements in COBOL

The stored statements submitted by COBOL are loaded into the database in Data Mover scripts. You can make any change simply by editing the script and reloading it with Data Mover.

PeopleSoft delivers only a single set of stored statements, so they are all coded in a platform-generic fashion. If you are only running Oracle, then you are free to introduce Oracle-specific changes to performance. However, you will have to treat that as a customization, and compare new versions of the Data Mover scripts when they are delivered.

There are some restrictions you should observe:

- In the query in Listing 11-71, it might be tempting to change the joins in the subquery on PS_EARNINGS_BAL to use the bind variables. The bind variables correspond to specific coding in the programs so that COBOL variables are assigned to the bind variables. Each bind variable should appear only once in a stored statement.
- Columns should not be added to nor removed from the SELECT clause, nor should the order or type of the columns be changed. The COBOL programs create cursors for each of the stored statements to read the selected values into program variables. If the stored statement does not match the coding in the COBOL program, an error will occur.

Listing 11-71. *A stored statement*

```
STORE FGPCADJG_S_ERNYTD
SELECT
SUM(A.GRS_YTD)
FROM PS_EARNINGS_BAL A
WHERE A.EMPLID = :1
  AND A.COMPANY = :2
  AND A.BALANCE_ID = :3
```

```
  AND A.BALANCE_YEAR = :4
  AND A.BALANCE_PERIOD = (SELECT MAX(B.BALANCE_PERIOD)
                          FROM PS_EARNINGS_BAL B
                          WHERE B.EMPLID = A.EMPLID
                            AND B.COMPANY = A.COMPANY
                            AND B.BALANCE_ID = A.BALANCE_ID
                            AND B.BALANCE_YEAR = A.BALANCE_YEAR
                            AND B.BALANCE_PERIOD <= :5)
...
;
```

SQR

SQL statements are simply coded in text files that can be altered by any text editor. The SQL passed to the database is exactly as it is coded in the source. All of the standard query tuning methods can be implemented in SQR. However, there are some limitations.

Hints must be placed on lines on their own in BEGIN-SELECT clauses, as shown in Listing 11-72.

Listing 11-72. *Hints in SQR*

```
Begin-Select On-Error=SQL-Error
/*+ALL_ROWS*/
m.model_statement
m.parmcount
m.statement_type
...
```

Queries are made distinct by adding the DISTINCT keyword to the BEGIN-SELECT keyword. If you try to add the DISTINCT keyword to the main SELECT clause, the SQR will error. Hence, in the SQL submitted to the database, the hint will appear after the DISTINCT keyword and will not be recognized as a hint by the database.

In the example in Listing 11-73, I have added a comment immediately after the BEGIN-SELECT clause. You can see the resulting SQL in Listing 11-74. The comment appears in the SQL after the DISTINCT keyword. The workaround for hints, also shown in these listings, is to add an in-line view that selects a row from DUAL. This does not functionally alter the query, but you can now put the hint in the in-line view.

Listing 11-73. *Extract from* `sysrecrd.sqc`

```
begin-SELECT DISTINCT on-Error=SQL-Error
/*SYSRECORD-13*/
RECNAME          &Record13_RecName

 if (((#current-line + 1) = #sqr-max-lines) and $DetailErrFound = 'Y') or
($DetailErrFound = 'N')
       move 'Y' to $DetailErrFound
       do PrintSectionHeadings
   end-if
   let #rows = #rows +1
```

```
      print &Record13_RecName      (+1,#Start1)

FROM PSRECDEFN
, (SELECT /*+ALL_ROWS*/ 'x' FROM DUAL)
WHERE RECTYPE = 7 AND SQLTABLENAME <> ' '
ORDER BY RECNAME
end-SELECT
```

The SQL submitted to the database is shown in Listing 11-74.

Listing 11-74. *SQL generated by Listing 11-73*

```
SELECT DISTINCT /*SYSRECORD-13*/ RECNAME
FROM PSRECDEFN
, (SELECT /*+ALL_ROWS*/ 'x' FROM DUAL)
WHERE RECTYPE = 7 AND SQLTABLENAME <> ' '
ORDER BY RECNAME
```

Upgrade Considerations

Every one of the techniques described in this chapter involves making changes to the PeopleSoft application. These must be viewed as customizations. Changes to objects in the Application Designer can be seen in the compare reports. Changes to any flat files must be handled manually.

PeopleSoft patches and upgrades frequently rerelease existing programs. Care must be taken to make sure that while the new or fixed functionality is incorporated, the performance improvements are not lost. Therefore, it is essential that all changes be thoroughly documented.

Summary

PeopleSoft is a packaged application, and much of the SQL it generates is buried deep inside the application. To access that code requires some knowledge of PeopleSoft's proprietary design tools, PeopleTools, though it does not require that you have the proficiency of a developer. You are looking for SQL, and most of the time it still looks like SQL.

In this chapter, I discussed how to enable Oracle SQL trace. Having identified the SQL that needs to be tuned, I demonstrated some techniques that will help you find the source of that SQL in the PeopleSoft application, and then implement Oracle tuning techniques.

Resolving SQL performance problems *is* properly a part of the DBA's job. To be effective, the DBA must not simply identify the problem SQL, but must also determine its source, and how it can be fixed.

CHAPTER 12

Configuring the Application Server

This chapter deals with the mechanics of configuring the application server. It explains how the various files are combined and compiled in the Tuxedo configuration file. An understanding of this is necessary before moving on to sizing and configuring the application server for optimal performance, which is discussed in the next chapter. This chapter also discusses the Tuxedo Administration Console, a Java applet shipped by BEA with Tuxedo that permits remote administration and some monitoring of the application server.

Naturally, PeopleSoft also provides documentation about the application server. This chapter should be read in conjunction with that documentation.

Overview of Configuration Files

The application server is administered via the `psadmin` utility in `$PS_HOME/appserv`. This is a text mode utility that works in exactly the same way on Unix and Windows. The domain configuration process combines and generates a number of files, as shown in Table 12-1.

Table 12-1. *Tuxedo Configuration Files*

File Name	Description
`psappsrv.cfg`	Parameter values for both variables in the Tuxedo configuration file `psappsrv.ubx` and application servers.
`psappsrv.val`	Validation file for the interactive configuration dialog.
`psappsrv.ubx`	PeopleSoft Tuxedo template file to specify the layout of `psappsrv.ubx` and `psappsrv.env`.
`psappsrv.ubb`	Tuxedo native domain configuration source file. All of the variables have been resolved to literals.
`psappsrv.env`	Application server environment file that defines additional environmental variables for server processes.
`PSTUXCFG`	Compiled binary Tuxedo configuration file.

The application server domain configuration process is an option within the `psadmin` utility, as shown in Listing 12-1. The domain must be shut down in order to reconfigure it. As a safety precaution, you are asked whether you want to proceed.

Listing 12-1. `psadmin` *utility*

```
---------------------------------
PeopleSoft Domain Administration
---------------------------------
     Domain Name: HR88

  1) Boot this domain
  2) Domain shutdown menu
  3) Domain status menu
  4) Configure this domain
  5) TUXEDO command line (tmadmin)
  6) Edit configuration/log files menu
  7) Messaging Server Administration menu
  q) Quit

Command to execute (1-7, q) : 4

This option will shutdown the domain.
Do you want to continue? (y/n) [n] :
```

From PeopleTools 8.44, a new screen of information, called the Features and Settings report, is displayed before the configuration process is initiated, as shown in Listing 12-2. This has also been backported to PeopleTools 8.20. Selecting options 1 to 10 will change the value of the corresponding configuration variable for the current `psadmin` session but will not update `psappsrv.ubx`.

Listing 12-2. *PeopleTools 8.44 domain features and settings summary*

```
---------------------------------------------
Quick-configure menu -- domain: HR88
---------------------------------------------
      Features                     Settings
     ==========                   ==========
  1) Pub/Sub Servers   : No    13) DBNAME     :[HR88]
  2) Quick Server      : No    14) DBTYPE     :[ORACLE]
  3) Query Servers     : No    15) UserId     :[PSAPPS]
  4) Jolt              : Yes   16) UserPswd   :[PSAPPS]
  5) Jolt Relay        : No    17) DomainID   :[HR88]
  6) PC Debugger       : No    18) AddToPATH  :[d:\oracle\ora92\bin]
  7) Opt Engines       : No    19) ConnectID  :[people]
  8) Event Notification: No    20) ConnectPswd:[peop1e]
  9) MCF Servers       : No    21) ServerName :[]
 10) Perf Collator     : No    22) WSL Port   :[8800]
                               23) JSL Port   :[8810]
                               24) JRAD Port  :[9100]
```

```
      Actions
     =========
 11) Load config as shown
 12) Custom configuration
  h) Help for this menu
  q) Return to previous menu

HINT: Enter 13 to edit DBNAME, then 11 to load

Enter selection (1-24, h, or q): 12

Removing any existing configuration...
Generating new configuration...
Loading validation table...
```

Up to PeopleTools 8.43, the reconfiguration process continues from this point, asking whether you want to change any configuration values (as shown in Listing 12-3). From PeopleTools 8.44, option 12 will continue in the same manner, and option 11 will skip this section and proceed with configuration.

Listing 12-3. *Part of the interactive configuration dialog in PeopleTools 8.44*

```
Do you want to change any config values (y/n)? [n]:y
```

Figure 12-1 illustrates how the various configuration files are used and generated.

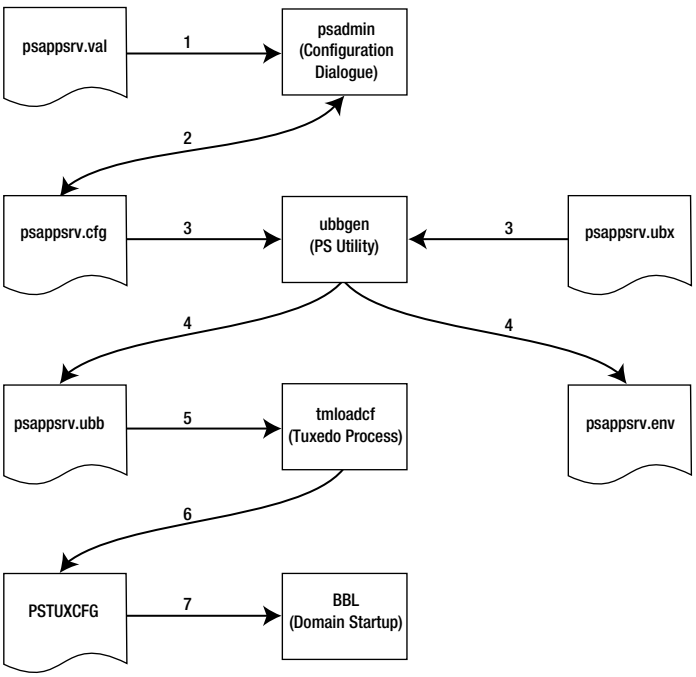


Figure 12-1. *Configuration processes and files*

The step details are as follows:

1–2. Although most people simply edit the psappsrv.cfg configuration file directly, there is an optional interactive configuration dialog in psadmin. The responses are validated with the instructions in the psappsrv.val file. The passwords in psappsrv.cfg can be encrypted within the interactive dialog.

3–4. The psappsrv.cfg and psappsrv.ubx files are read by ubbgen, which is a PeopleSoft process that generates the psappsrv.ubb and psappsrv.env files.

5–6. The Tuxedo tmloadcf utility compiles psappsrv.ubb into the binary configuration file PSTUXCFG.

7. When the domain is started, the BBL process is established. This process reads the PSTUXCFG file and starts the rest of the domain.

The ubbgen and tmloadcf processes are run back-to-back, without pause, by psadmin.

■ **Caution** You should never edit psappsrv.ubb. It is an intermediate file generated by the configuration process. However, it is useful to look at it to see what values were actually used to configure the Tuxedo domain.

In the following sections, I describe each of the configuration files, explaining how they are used and the purpose of some of the parameters. The examples are taken from PeopleTools 8.44, but the essential structure has not changed significantly since PeopleTools 7.0. PeopleSoft has added more servers as it has introduced more functionality.

psappsrv.ubx

This PeopleSoft template file is used to generate the psappsrv.ubb and psappsrv.env files that must conform to Tuxedo formats. It contains variables whose values are specified in the psappsrv.cfg file. The output files contain the literal values of those variables.

RELATIONSHIP BETWEEN THE TUXEDO TEMPLATE AND THE CONFIGURATION FILE

Most domain configuration changes will be made in the configuration file, psappsrv.cfg. However, there are legitimate reasons to make changes to the template file, psappsrv.ubx. Where different domains have different settings, a variable should be set in psappsrv.cfg and referenced in psappsrv.ubx. Thus you can arrange to have a single psappsrv.ubx, common to all domains at the same level of PeopleTools.

For example, you might want to enable the Tuxedo service trace (as discussed in Chapter 9). This requires adding the -r option to the server command line in psappsrv.ubx. I prefer to create a new variable in the trace section of psappsrv.cfg and reference it in psappsrv.ubx. This option can easily be enabled or disabled on different domains by changing a setting in the configuration file.

The new variables also appear in the interactive configuration dialog, as shown in Listing 12-4.

Listing 12-4. *Configuration variables in the interactive dialog*

```
Values for config section - Trace
    TraceSql=3
    TraceSqlMask=12319
    TracePC=0
    TracePCMask=4095
    TracePPR=0
    TracePPRMask=32767
    TraceAE=1152
    TraceOpt=0
    TraceOptMask=4095
    TracePPM=0
    DumpMemoryImageAtCrash=NONE
    DumpManagerObjectsAtCrash=Y
    LogErrorReport=N
    MailErrorReport=
    TuxedoServiceTrace=-r

Do you want to change any values (y/n)? [n]:
```

The new variable is set in the configuration file, as shown in Listing 12-5.

Listing 12-5. *Additional variable in* `psappsrv.cfg`

```
;-------------------------------------------------------------------------
; when set to -r enables Tuxedo service trace, else set to a single space
TuxedoServiceTrace=-r
```

It duly appears in `psappsrv.ubb` (see Listing 12-6), including the variable reference at the bottom.

Listing 12-6. *New variable in* `psappsrv.ubb`

```
CLOPT="-r -e D:\ps\hr88\appserv\HR88\LOGS\APPQ.stderr  -s@..\psappsrv.lst
-s@..\psqcksrv.lst -sICQuery -sSqlQuery:SqlRequest -- -C psappsrv.cfg -D HR88 -S
PSAPPSRV"
...
#*********************************************************************
#  ubbgen substitution values:
#
...
# [16]:                          {$Trace\TuxedoServiceTrace}:  -r
```

Features, Settings, and Ports Sections

The features, settings, and port settings sections, shown in Listing 12-7, are new in PeopleTools 8.44. They are used to generate the new features and settings report (shown in Figure 12-2 later in this chapter).

Listing 12-7. *Features, settings, and port settings sections in* `psappsrv.ubx`

```
*FEATURES
{"-label-"},{"-define-"},{"-on-"},{"-servers-"}
{Pub/Sub Servers},{PUBSUB},{No},{PSBRK(DSP/HND), PSPUB(DSP/HND), PSSUB(DSP/HND)}
{Quick Server},{QUICKSRV},{No},{PSQCKSRV}
{Query Servers},{QUERYSRV},{No},{PSQRYSRV}
{Jolt},{JOLT},{Yes},{JSL, JREPSVR}
{Jolt Relay},{JRAD},{No},{JRAD}
{PC Debugger},{DBGSRV},{No},{PSDBGSRV}
{Opt Engines},{OPTENG},{No},{PSOPTENG}
{Event Notification},{RENSRV},{Yes},{PSRENSRV}
{MCF Servers},{MCF},{No},{PSUQSRV, PSMCFLOG}
{Perf Collator},{PPM},{No},{PSPPMSRV}
*END
*SETTINGS
{"-label-"},{-formal name-}{"-value-"}
{DBNAME},{Startup/DBName},{ }
{WINDOWS}
{DBTYPE},{Startup/DBType},{MICROSFT}
{WINDOWS}
{UNIX}
{DBTYPE},{Startup/DBType},{}
{UNIX}
{UserId},{Startup/UserId},{QEDMO}
{UserPswd},{Startup/UserPswd},{QEDMO}
{DomainID},{Domain Settings/Domain ID},{PT81}
{WINDOWS}
{AddToPATH},{Domain Settings/Add to PATH},{}
{WINDOWS}
{UNIX}
{AddToPATH},{Domain Settings/Add to PATH},{.}
{UNIX}
{ConnectID},{Startup/ConnectId},{people}
{ConnectPswd},{Startup/ConnectPswd},{peop1e}
{ServerName},{Startup/ServerName},{}
*END
*PORT_SETTINGS
{WSL Port},{Workstation Listener/Port},{7000}
{JSL Port},{JOLT Listener/Port},{9000}
{JRAD Port},{JOLT Relay Adapter/Listener Port},{9100}
*END
*END
```

PS_DEFINES Section

The PS_DEFINES section is the only place where you will find any explanation of the ubbgen process and the format of psappsrv.ubx. Essentially, ubbgen substitutes variables delimited by curly brackets ({ or }) with literal values from the psappsrv.cfg file. There are four types of variables: environment, configuration, special, and prompted. I discuss each type in detail in the sections that follow.

Environment Variables

Environment variables are read by ubbgen with the genenv() function. However, the psadmin program also sets some environment variables. For example, $TUXDIR, the directory where Tuxedo was installed, must be set in the environment before executing psadmin, but $PS_SERVDIR, the directory where the domain configuration files are located, is set by psadmin. These variables are then used to set the TUXDIR and APPDIR parameters in the MACHINES section of the Tuxedo configuration:

```
TUXDIR="{$TUXDIR}"                    # Paths cannot end in '\'
APPDIR="{$PS_SERVDIR}"                # include the database path
```

Configuration Variables

Configuration variables are defined by the Tuxedo template file, psappsrv.ubx. ubbgen looks up the value of the variable in the configuration file psappsrv.cfg and substitutes the variable with its value as it writes the Tuxedo configuration file psappsrv.ubb. In the following example, Min Instances will be read from the PSAPPSRV section of psappsrv.cfg:

```
PSAPPSRV        SRVGRP=APPSRV
                SRVID=1
                MIN={$PSAPPSRV\Min Instances}
```

Special Variables

A number of *special variables* are generated by ubbgen. These have changed from release to release. The variables generated by ubbgen in PeopleTools 8.44 are set out in Table 12-2.

The values of MAXACCESSERS, MAXSERVERS, MAXSERVICES, and MAXWSCLIENTS are calculated from the number of servers specified in the configuration file psappsrv.cfg, and they are used to set the sizes of table structures in the Bulletin Board to the maximum values that could be used by the domain. This minimizes the use of IPC resources.

Table 12-2. ubbgen *Special Variables*

Variable	Description
CFGFILE	Name of the PeopleSoft configuration file. This is always psappsrv.cfg.
ENVFILE	Name of the server environment file. This is always psappsrv.env.
FS	Directory separator character. / on Unix; \ on Windows. Thus PeopleSoft can deliver the same configuration file for all platforms.
GID	Unix group ID that will run the Tuxedo domain. This is always 0 on Windows.
HI_WSL_PORT	Maximum WSL port number. For use when a firewall exists between the clients and the WSL.

(Continues)

Table 12-2. ubbgen *Special Variables (Continued)*

Variable	Description
IPCKEY	Segment ID for the Bulletin Board. It must be unique across the machine.
LOGDIR	Location of files, generated as $PS_HOME/appserv/<domain name>/LOGS.
LO_WSL_PORT	Minimum WSL port number. For use when a firewall exists between the clients and the WSL.
MACH	Name of the machine. Equivalent to result of uname -n on Unix systems.
MAXACCESSERS	Used to specify the total number of clients of the Bulletin Board, specified in the MACHINES section of the Tuxedo configuration file (see Listing 12-11). It is calculated as follows: = Number of WSH * (Number of clients per WSH + 1) + Number of JSL * (Number of clients per JSH + 2), if Jolt configured + Maximum number of PSAPPSRV processes + Maximum number of PSQCKSRV processes, if configured + Maximum number of PSQRYSRV processes, if configured + Maximum number of PSBRKHND_dflt, PSSUBHND_dflt, and PSPUBHND_dflt servers, if publish and subscribe configured + 1, if debug server enabled + Maximum number of OPTENG servers, if configured + 3, if MultiChannel Framework Servers (MCF) enabled + Maximum number of PSPPMSRV servers, if performance collector enabled + 11
MAXSERVERS	Total number of server processes required by Tuxedo to size the Bulletin Board, used in the RESOURCES section of the Tuxedo configuration file (see Listing 12-10). It is calculated as follows: = Maximum number of WSH processes + Maximum number of PSAPPSRV processes + Maximum number of PSQCKSRV processes, if enabled + Maximum number of PSQRYSRV processes, if enabled + 2 * Maximum number of JSH processes, if Jolt configured + Maximum number of PSBRKHND_dflt, PSSUBHND_dflt, and PSPUBHND_dflt servers, if publish and subscribe configured + 1, if debug server enabled + Maximum number of PSOPTENG servers + 1, if event notification enabled + 2, if MultiChannel Framework Servers (MCF) enabled + Maximum number of PSPPMSRV servers, if performance collector enabled + 11
MAXSERVICES	Calculated total number of services advertised across all server processes, used in the RESOURCES section of the Tuxedo configuration file (see Listing 12-10). It is calculated as follows: = 2 * Maximum number of WSH processes + 74 * Maximum number of PSAPPSRV processes + 8 * Maximum number of PSQCKSRV processes, if enabled + 2 * Maximum number of PSQRYSRV processes, if enabled + Number of JSH processes * Number of clients per JSH, if Jolt enabled + Maximum number of PSBRKHND_dflt, PSSUBHND_dflt, and PSPUBHND_dflt servers, if publish and subscribe configured + $2n\_ + 5n$, where n is the number of optimization servers that are enabled + 1, if event notification enabled + 2, if MultiChannel Framework Server (MCF) enabled + 31

Variable	Description
MAXWSCLIENTS	The sum of the maximum number of clients that the WSH and JSH processes can support, specified in the MACHINES section of the Tuxedo configuration file (see Listing 12-11). It is calculated as follows: = Maximum number of WSH * Number of clients per WSH + Maximum number of JSH processes * Number of clients per JSH
UID	Unix user ID that will run the Tuxedo domain. This is always 0 on Windows.
UNIX	True if running on a Unix platform; otherwise, false.
VERITY_OS	Part of path to Verity product used to search PeopleBooks.
VERITY_PLATFORM	Part of path to Verity product used to search PeopleBooks.
WINDOWS	True if running on a Windows platform; otherwise, false.

These formulas are very likely to change from release to release as additional servers and services are introduced. The formulas in this section were determined for ubbgen supplied with PeopleTools 8.44.09.

If you configure multiple queues for any application server process (see Chapter 13), you will need to manually calculate these values, because ubbgen will not take the additional queues into account.

Prompted Variables

All other variables are called *prompted*. They specify a number of yes/no questions that are asked during configuration (see Listing 12-8). These variables, and the Unix and Windows special variables, are used like #ifdef and #endif directives in C or SQR. The default responses to the questions are enclosed in square brackets.

Listing 12-8. *Prompted variables defined in* psappsrv.ubx

```
{PUBSUB} Do you want the Publish/Subscribe servers configured (y/n)? [n]:
{QUICKSRV} Move quick PSAPPSRV services into a second server (PSQCKSRV) (y/n)? [n]:
{QUERYSRV} Move long-running queries into a second server (PSQRYSRV) (y/n)? [n]:
{JOLT} Do you want JOLT configured (y/n)? [y]:
{JRAD} Do you want JRAD configured (y/n)? [n]:
{DBGSRV} Do you want to enable PeopleCode Debugging (PSDBGSRV) (y/n)? [n]:
{OPTENG} Do you want the Optimization engines configured (y/n)? [n]:
{RENSRV} Do you want Event Notification configured (PSRENSRV) (y/n)? [n]:
{MCF} Do you want MCF servers configured (y/n)? [n]:
{PPM} Do you want Performance Collators configured (PSPPMSRV) (y/n)? [n]:

*END
```

For example, if the QUERYSRV special variable is true, then the PSQRYSRV server is configured by including the section between the {QUERYSRV} references, as shown in Listing 12-9.

Listing 12-9. *Query server definition inside a prompted variable*

```
{QUERYSRV}
#
# PeopleSoft Query Application Server
#
PSQRYSRV        SRVGRP=APPSRV
                SRVID=70
                MIN={$PSQRYSRV\Min Instances}
                MAX={$PSQRYSRV\Max Instances}
                RQADDR="QRYQ"
                REPLYQ=Y
                CLOPT="{$PSQRYSRV\Spawn Server} -sICQuery -sSqlQuery:SqlRequest
-- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSQRYSRV"
{QUERYSRV}
```

■ **Tip** Your responses to the prompted variables will be determined by what features you are using. For instance, you will only need publish and subscribe servers if you are using that functionality in PeopleSoft. You should set the default values in psappsrv.ubx so that you simply accept the default values when configuring domains.

Main Tuxedo Section

Everything in `psappsrv.ubx` after the marked end of PS_DEFINES section until the PS_ENVFILE section is copied by `ubbgen` to `psappsrv.ubb`, after the variable substitutions have been made. `psappsrv.ubb` is a Tuxedo format file, as described in the Tuxedo documentation (see the reference section `ubbconfig(5)`), so the template in `psappsrv.ubx` must produce that format.

A Tuxedo configuration file consists of a number of sections: RESOURCES, MACHINES, GROUPS, SERVERS, and SERVICES.

Tuxedo RESOURCES Section

The RESOURCES section contains variables that describe the whole Tuxedo domain. The template for this section is shown in Listing 12-10.

Listing 12-10. *Template for the Tuxedo RESOURCES section*

```
*RESOURCES
IPCKEY          {IPCKEY}        # ( 32768 < IPCKEY < 262143 )
MASTER          "{MACH}"
DOMAINID        {$Domain Settings\Domain ID}_{IPCKEY}
MODEL           SHM
LDBAL           N
#
```

```
MAXMACHINES     256             # min, default=256
MAXGROUPS       100             # min, default=100
{MAXSERVERS}
{MAXSERVICES}
#
...
```

Tuxedo load balancing can be used when there is a choice of queues on which to place service requests (see Chapter 13).

Tuxedo MACHINES Section

A single Tuxedo domain can be split across more than one physical machine to permit load balancing. The Tuxedo machine on each physical server looks similar to an independent Tuxedo domain, but it has additional listener and bridge processes to connect it to the other machines in the domain. Thus, Tuxedo services can be transferred to a different, less heavily loaded Tuxedo machine.

The MACHINES section, the template for which is shown in Listing 12-11, specifies variables that could be set to different values on different physical machines, such as paths or environmental variables. Some of these parameters can be specified in both the RESOURCES and MACHINES sections. The specific value in the MACHINES section overrides the generic value in the RESOURCES section.

Although it is theoretically possible to configure a PeopleSoft application server domain to use the multimachine architecture, PeopleSoft neither uses nor supports this. In practice, you would never want to use multiple machines, because instead you would have multiple, independent application server domains on various physical machines.[1]

Listing 12-11. *Template for the Tuxedo MACHINES section*

```
*MACHINES
"{MACH}" LMID="{MACH}"                          # Machine name must be uppercase
        TUXDIR="{$TUXDIR}"                      # Paths cannot end in '\'
        APPDIR="{$PS_SERVDIR}"                  # include the database path
        TUXCONFIG="{$TUXCONFIG}"
        ULOGPFX="{LOGDIR}{FS}TUXLOG"
        ENVFILE="{$PS_SERVDIR}{FS}{ENVFILE}"
        UID={UID}                               # Has to be 0 at this time.
        GID={GID}                               # Has to be 0 at this time.
{WINDOWS}
        TYPE="i386NT"
{WINDOWS}
        NETLOAD=0                               # We are not using multiple machines.
        {MAXWSCLIENTS}
        {MAXACCESSERS}
```

1. I have seen the multi-machine architecture configured only once, in a production PeopleTools 7.5 system. Ultimately, it was removed due to instability. The configuration was found to be very sensitive to a problem with the network links between the physical application server nodes. It was found that using round-robin connection and failover between multiple single-machine application servers in the client connection configuration was equally effective and more reliable.

Tuxedo GROUPS Section

Application servers in a domain are logically divided into groups. The GROUPS section, shown in Listing 12-12, has a default clause that specifies the machine name. Tuxedo groups permit the same service to be configured differently on different servers in different groups.

With successive releases, PeopleSoft has created separate Tuxedo groups for the new server processes, but since PeopleTools 7.57 each service is advertised on only one queue. In the delivered application server configuration, PeopleSoft has not yet used the groups for any practical purpose. At some point in the future PeopleSoft could, for instance, apply different Tuxedo parameters to different groups of servers.

Listing 12-12. *Template for the Tuxedo GROUPS section*

```
*GROUPS

#
# Tuxedo Groups
# For application group numbers for new machines (LMIDs)
# use group numbers 101-199;  201-299;  etc.
#
DEFAULT:        LMID="{MACH}"

BASE            GRPNO=1

WATCH           GRPNO=10

MONITOR         GRPNO=50

PPMGRP          GRPNO=91

{RENSRV}
RENGRP          GRPNO=92
{RENSRV}

{MCF}
MCFGRP          GRPNO=93
{MCF}

{OPTENG}
OPTGRP          GRPNO=96
{OPTENG}

{DBGSRV}
DBGSRV          GRPNO=97
{DBGSRV}

{PUBSUB}
PUBSUB          GRPNO=98
{PUBSUB}
```

```
APPSRV          GRPNO=99

{JOLT}
#
# JOLT Groups
#
JREPGRP         LMID="{MACH}"
                GRPNO=94
JSLGRP          LMID="{MACH}"
                GRPNO=95
{JOLT}
```

Some groups (RENGRP, MCFGRP, OPTGRP, PUBSUB, and JOLT) are defined only if the corresponding prompt variable is true.

Tuxedo SERVERS Section

The list of server processes has continued to grow with each release of PeopleTools. Each of the server processes in a domain is specified in the SERVERS section of `psappsrv.ubx`.

The default clause, shown in Listing 12-13, specifies default parameters that apply to all servers unless overridden on the server definition. The Tuxedo documentation describes CLOPT, the command-line options, separately in the reference section in `servopts(5)`.

Listing 12-13. *Server defaults definition*

```
*SERVERS

DEFAULT:        CLOPT="-A"      # Advertise all services.
                REPLYQ=N        # Reply queue not needed for our simple setup.
                MAXGEN=6        # Max number of restarts in the grace period.
                GRACE=60        # Grace period in seconds.
                RESTART={$Domain Settings\Restartable}
                SYSTEM_ACCESS=FASTPATH
```

A server can be started six (MAXGEN) times or, if you prefer, restarted five times (MXGEN – 1) times in the 60-second grace period (GRACE), before it is marked as being in error.

Tuxedo Event Broker

The Tuxedo Event Broker, TMSYSEVT, is the only server process delivered by BEA. It is not configured by default in PeopleSoft application servers, but it is commented out as shown in Listing 12-14. It is useful only if you want to use the Tuxedo Administration Console to administer the domain. It is documented in the Tuxedo `tmsysevt(5)` reference. Configuration of the console is discussed at the end of this chapter.

Listing 12-14. *The Tuxedo Event Broker definition is commented out.*

```
#
# Tuxedo System Event Server
#
#TMSYSEVT        SRVGRP=BASE
#                SRVID=200
```

Watch Server (PSWATCHSRV)

PSWATCHSRV (see Listing 12-15) was introduced in PeopleTools 8.4. It is unusual in that it does not connect to the database. It does not advertise or handle services. Its job is to kill server processes that are stuck and cannot be shut down.

■ **Note** Any parameters to the left of the double hyphen in the command-line option are Tuxedo parameters that can be found in the Tuxedo `servopts(5)` reference. Anything to the right is passed to the `tmsvrinit()` function defined in the server by PeopleSoft. So, anything to the right of the double hyphen is a PeopleSoft-defined parameter. PeopleSoft does not document these parameters.

IPCKEY, which is the ID of the Bulletin Board shared memory segment, is passed to PSWATCHSRV. Therefore, I infer that the server is making an attachment to the Bulletin Board so that it can search for locked servers.

There is only ever one instance of PSWATCHSRV. The minimum and maximum number of instances of the server is hard-coded as one.

Listing 12-15. *PSWATCHSRV server definition*

```
#
# PeopleSoft domain watchdog PSWATCHSRV
#
PSWATCHSRV      SRVGRP=WATCH
                SRVID=1
                MIN=1
                MAX=1
                RQADDR="WATCH"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -ID {IPCKEY} -C {CFGFILE} -D {$Domain Settings\Domain ID}
-S PSWATCHSRV"
```

Main Application Server (PSAPPSRV)

The PSAPPSRV server (see Listing 12-16) does most of the work for online activity in the PeopleSoft application server.

Listing 12-16. *PSAPPSRV server definition*

```
#
# PeopleSoft Application Server
#
PSAPPSRV        SRVGRP=APPSRV
                SRVID=1
                MIN={$PSAPPSRV\Min Instances}
                MAX={$PSAPPSRV\Max Instances}
                RQADDR="APPQ"
                REPLYQ=Y
```

The command-line option for PSAPPSRV, shown in Listing 12-17, varies depending upon whether separate PSQCKSRV or PSQRYSRV processes are enabled. If they are enabled, then the services that they advertise are no longer advertised by PSAPPSRV.

The services to be advertised are specified by the `-s` parameter. There are three formats for this parameter, as shown in Table 12-3. This parameter can be specified several times, as shown in Listing 12-17.

Table 12-3. *Advertise Service Parameter Formats*

Format	Description
`-s <filename>`	Advertise the list of services in the named files. PeopleSoft supplies two files, `psappsrv.lst` and `psqcksrv.lst`, which are located in `$PS_HOME/appserv`. These files are simply a list of services. All the domains located in a particular $PS_HOME share the same files.
`-s <service name>`	Advertise a named service.
`-s <service name:function name>`	Normally, the service and function names are the same. This format allows a single function to handle different services. PeopleSoft uses this to distinguish between long-running queries (`SqlQuery`) issued by the query tools and other, quicker queries (`SqlRequest`). The long-running queries can then be moved to a different server so that they do not block the main application serving server processes.

The substitution variables can be nested two deep, as shown in Listing 12-17, but no further. This permits four different command lines for the PSAPPSRV server depending upon whether PSQCKSRV and PSQRYSRV are configured. If they are available, then certain services are advertised on those servers instead of PSAPPSRV.

Listing 12-17. *PSAPPSRV command-line definition showing nested substitution variables*

```
{QUERYSRV}
{QUICKSRV}
                CLOPT="{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSAPPSRV"
{QUICKSRV}
{!QUICKSRV}
                CLOPT="{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst
-s@..{FS}psqcksrv.lst -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSAPPSRV"
{!QUICKSRV}
{QUERYSRV}
{!QUERYSRV}
{QUICKSRV}
                CLOPT="{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst -sICQuery
-sSqlQuery:SqlRequest -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSAPPSRV"
{QUICKSRV}
{!QUICKSRV}
```

```
                CLOPT="{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst
-s@..{FS}psqcksrv.lst -sICQuery -sSqlQuery:SqlRequest -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSAPPSRV"
{!QUICKSRV}
{!QUERYSRV}
```

Quick Server (PSQCKSRV)

PSQCKSRV (see Listing 12-18) was introduced in PeopleTools 7 to separately handle the very short duration services, such as simple SQL queries, so they could get through the application server quickly and did not have to wait in the queue behind the larger and slower services. The concept is rather like a "10 items or less" checkout line at a supermarket. However, these services are only used by the Windows three-tier client and the Application Designer. The PIA does not require this server.

Listing 12-18. *PSQCKSRV server definition*

```
{QUICKSRV}
#
# PeopleSoft Quick Application Server
#
PSQCKSRV        SRVGRP=APPSRV
                SRVID=50
                MIN={$PSQCKSRV\Min Instances}
                MAX={$PSQCKSRV\Max Instances}
                RQADDR="QCKQ"
                REPLYQ=Y
                CLOPT="{$PSQCKSRV\Spawn Server} -s@..{FS}psqcksrv.lst -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSQCKSRV"
{QUICKSRV}
```

Query Server (PSQRYSRV)

PSQRYSRV (see Listing 12-19) handles the long-running queries issued by the Query and nVision tools in the PIA and in three-tier mode. If Crystal Reports is run in three-tier mode, the PeopleSoft ODBC driver also submits `SqlQuery` service requests.

Listing 12-19. *PSQRYSRV server definition*

```
{QUERYSRV}
#
# PeopleSoft Query Application Server
#
PSQRYSRV        SRVGRP=APPSRV
                SRVID=70
                MIN={$PSQRYSRV\Min Instances}
                MAX={$PSQRYSRV\Max Instances}
                RQADDR="QRYQ"
```

```
                REPLYQ=Y
                CLOPT="{$PSQRYSRV\Spawn Server} -sICQuery -sSqlQuery:SqlRequest –
-C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSQRYSRV"
{QUERYSRV}
```

PSSAMSRV

The PSSAMSRV server (see Listing 12-20) is used only by three-tier clients, and mostly when submitting requests to the Process Scheduler. However, it cannot be disabled.

Listing 12-20. *PSSAMSRV server definition*

```
#
# PeopleSoft SQL Access Application Server
#
PSSAMSRV        SRVGRP=APPSRV
                SRVID=100
                MIN={$PSSAMSRV\Min Instances}
                MAX={$PSSAMSRV\Max Instances}
                RQADDR="SAMQ"
                REPLYQ=Y
                CONV=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSSAMSRV"
```

Performance Collator Server (PSPPMSRV)

The performance collator server, PSPPMSRV, is new in PeopleTools 8.44 (see Listing 12-21). It is used by the PeopleSoft Performance Monitor (which is discussed in detail in Chapter 10) to collect metrics and insert them into the database. These servers are configured on the system collecting the performance metrics, not the system being measured.

Listing 12-21. *PSPPMSRV server definition*

```
{PPM}
# Performance Collator. No services, just managed by Tuxedo.
PSPPMSRV        SRVGRP=PPMGRP
                SRVID=100
                MIN={$PSPPMSRV\Min Instances}
                MAX={$PSPPMSRV\Max Instances}
                RQADDR="PPMQ2"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSPPMSRV"
{PPM}
```

Debug Server (PSDBGSRV)

The debug server process, PSDBGSRV (see Listing 12-22), permits the developer to step through PeopleCode in the Application Designer while running a PIA session. The Application Designer makes a socket connection to this server. Only one debug server can be configured in an application server.

Listing 12-22. *PSDBGSRV server definition*

```
{DBGSRV}
#
# PeopleCode Debugger PSDBGSRV
#
PSDBGSRV        SRVGRP=DBGSRV
                SRVID=1
                MIN=1
                MAX=1
                RQADDR="DBGQ"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSDBGSRV"

{DBGSRV}
```

Optimization Framework Servers (PSOPTENGn)

The OPTGRP group (see Listing 12-23) contains the processes associated with the PeopleSoft Optimization Framework (POF)[2] introduced in PeopleTools 8.4. It provides a basis for developing applications that use "optimization-based, decision-making capability in the PeopleTools environment."

Listing 12-23. *Optimization server definitions*

```
{OPTENG}
# Optimization Engines, one process per queue. One PIID in a queue.
# Each queue is explicitly listed with a unique SRVID, RQADDR, and command
# line server name. Server names are "PSOPTENGn", where 1 <=n <= 99.
PSOPTENG        SRVGRP=OPTGRP
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="OPTSQ1"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSOPTENG1"

PSOPTENG        SRVGRP=OPTGRP
                SRVID=102
                MIN=1
                MAX=1
                RQADDR="OPTSQ2"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSOPTENG2"
{OPTENG}
```

2. For more information, see the *POF Guide* in PeopleBooks.

Real-Time Notification Server (PSRENSRV)

The PSRENSRV process (see Listing 12-24) was introduced in PeopleTools 8.4. It is a modified web server used to send real-time event notifications, such as report notifications, to PIA users. It is also used by the MultiChannel Framework.

Listing 12-24. *PSRENSRV server definition*

```
{RENSRV}
# Event Notification server.
PSRENSRV        SRVGRP=RENGRP
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="RENQ1"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSRENSRV"
{RENSRV}
```

MultiChannel Framework Servers

The MultiChannel Framework (MCF) was introduced in PeopleTools in 8.4 "to support multiple interaction channels for call center agents or other PeopleSoft users who must respond to incoming requests and notifications on these channels." Listing 12-25 contains a definition of MCF servers.

Listing 12-25. *MCF servers definition*

```
{MCF}
# MCF Universal Queue server. These are stateful and unique; hence, each needs
# a unique ID on the command line "PSUQSRVn", where 1 <= n <= 9.
PSUQSRV         SRVGRP=MCFGRP
                SRVID=110
                MIN=1
                MAX=1
                RQADDR="UQSRV"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSUQSRV1"
# MCF Logging server. These are stateful and unique; hence, each needs
# a unique ID on the command line "PSMCFLOGn", where 1 <= n <= 9.
PSMCFLOG        SRVGRP=MCFGRP
                SRVID=120
                MIN=1
                MAX=1
                RQADDR="MCFLG"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSMCFLOG1"
{MCF}
```

Application Messaging Servers

There are up to six different server processes that can be configured to support application messaging (also referred to as the *Integration Broker*), as shown in Listing 12-26. It is possible to configure more than one instance of each of the handler servers, but there can be only a single instance of each of the three dispatcher servers.

Listing 12-26. *Application messaging servers definition*

```
{PUBSUB}
#############################################################################
#
# Publish/Subscribe Servers
#
#   THIS SECTION SHOULD NEVER BE EDITED MANUALLY, PSADMIN REQUIRES THIS EXACT FORMAT.
#                       -----
#############################################################################

# DEFAULT Publication broker handler
PSBRKHND        SRVGRP=PUBSUB
                SRVID=101
                MIN={$PSBRKHND_dflt\Min Instances}
                MAX={$PSBRKHND_dflt\Max Instances}
                RQADDR="BRKHQ_dflt"
                REPLYQ=Y
                CLOPT="{$PSBRKHND_dflt\Spawn Server} -s PSBRKHND_dflt:BrkProcess -
-C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSBRKHND_dflt"

# DEFAULT Publication broker server (dispatcher)
PSBRKDSP        SRVGRP=PUBSUB
                SRVID=100
                MIN=1
                MAX=1
                RQADDR="BRKDQ_dflt"
                REPLYQ=Y
                CLOPT="-s PSBRKDSP_dflt:Dispatch -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSBRKDSP_dflt"

# DEFAULT publication contract handler
PSPUBHND        SRVGRP=PUBSUB
                SRVID=201
                MIN={$PSPUBHND_dflt\Min Instances}
                MAX={$PSPUBHND_dflt\Max Instances}
                RQADDR="PUBHQ_dflt"
                REPLYQ=Y
                CLOPT="{$PSPUBHND_dflt\Spawn Server} -s PSPUBHND_dflt:PubConProcess -
-C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSPUBHND_dflt"

# DEFAULT Publication contractor server (dispatcher)
```

```
PSPUBDSP        SRVGRP=PUBSUB
                SRVID=200
                MIN=1
                MAX=1
                RQADDR="PUBDQ_dflt"
                REPLYQ=Y
                CLOPT="-s PSPUBDSP_dflt:Dispatch -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSPUBDSP_dflt"

# DEFAULT subscription contract handler
PSSUBHND        SRVGRP=PUBSUB
                SRVID=301
                MIN={$PSSUBHND_dflt\Min Instances}
                MAX={$PSSUBHND_dflt\Max Instances}
                RQADDR="SUBHQ_dflt"
                REPLYQ=Y
                CLOPT="{$PSSUBHND_dflt\Spawn Server} -s PSSUBHND_dflt:SubConProcess –
-C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSSUBHND_dflt"

# DEFAULT Subscription contractor server (dispatcher)
PSSUBDSP        SRVGRP=PUBSUB
                SRVID=300
                MIN=1
                MAX=1
                RQADDR="SUBDQ_dflt"
                REPLYQ=Y
                CLOPT="-s PSSUBDSP_dflt:Dispatch -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSSUBDSP_dflt"

#_@_APSRV   WARNING: DO NOT MODIFY THIS LINE. Marker for append point used by PSADMIN


##############################################################################
#   END Publish/Subscribe Servers section
##############################################################################
{PUBSUB}
```

In a development environment, the overhead of the application messaging servers can be significant. There is an alternative development template that can be selected when the application server domain is created (see Listing 12-49), in which there are only two messaging servers, as shown in Listing 12-27.

Listing 12-27. *Definition of application messaging servers for a development environment*

```
{PUBSUB}
# Message Broker
PSMSGDSP        SRVGRP=PUBSUB
                SRVID=100
```

```
                MIN=1
                MAX=1
                CLOPT="-sPSBRKDSP_dflt:Dispatch -sPSPUBDSP_dflt:Dispatch
-sPSSUBDSP_dflt:Dispatch -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S PSMSGDSP"

# Message Broker Handler
PSMSGHND        SRVGRP=PUBSUB
                SRVID=101
                MIN={$PSMSGHND\Min Instances}
                MAX={$PSMSGHND\Max Instances}
                RQADDR="MBHQ"
                REPLYQ=Y
                CLOPT="{$PSMSGHND\Spawn Server} -sPSBRKHND_dflt:BrkProcess
-sPSPUBHND_dflt:PubConProcess -sPSSUBHND_dflt:SubConProcess -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSMSGHND"
{PUBSUB}
```

Performance Monitor Server (PSMONITORSRV)

The PSMONITORSRV server (see Listing 12-28) was introduced in PeopleTools 8.44. It has three functions:

- Measuring host resource usage and Tuxedo performance metrics for the domain being monitored by the Performance Monitor
- Canceling ad hoc queries
- Resolving master/slave failover for the Integration Broker

This server is always configured, and there is only ever one instance of PSMONITORSRV in each domain.

Listing 12-28. *PSMONITORSRV server definition*

```
#
# PeopleSoft domain monitor
#
PSMONITORSRV    SRVGRP=MONITOR
                SRVID=1
                MIN=1
                MAX=1
                RQADDR="MONITOR"
                REPLYQ=Y
                RESTART=Y
                CLOPT="-A -- -ID {IPCKEY} -C {CFGFILE} -D {$Domain Settings\Domain ID}
-S PSMONITORSRV"
```

Workstation Listener and Handlers (WSL/WSH)

The Workstation Listener (WSL) listens for incoming connections from Windows three-tier clients. There are separate WSL command lines on Unix and Windows, as shown in Listing 12-29. On Unix, a device file is specified with the `-d` parameter.

Listing 12-29. *WSL server specification*

```
#
# Workstation Listener
#  -I xx   Max time (seconds) for a client connect
#  -T xx   Max time (minutes) for a client to stay idle.
#  -m xx   Min number of workstation handlers
#  -M xx   Max number of workstation handlers
#  -x xxx  Multiplexing, the max number of clients per handler
#
#
WSL             SRVGRP=BASE
                SRVID=20
{WINDOWS}
                CLOPT="-A -- -n {$Workstation Listener\Address}:{$Workstation
Listener\Port} -z {$Workstation Listener\Encryption} -Z {$Workstation
Listener\Encryption} -I {$Workstation Listener\Init Timeout} {WSL Client Cleanup

Timeout} -m {$Workstation Listener\Min Handlers} -M {$Workstation Listener\Max
Handlers} -x {$Workstation Listener\Max Clients per Handler} -c {$Workstation
Listener\Tuxedo Compression Threshold} -p {LO_WSL_PORT} -P {HI_WSL_PORT}"
{WINDOWS}
{UNIX}
                CLOPT="-A -- -n {$Workstation Listener\Address}:{$Workstation
Listener\Port} -z {$Workstation Listener\Encryption} -Z {$Workstation
Listener\Encryption} -d {$PS_TUXDEV} -I {$Workstation Listener\Init Timeout}
{WSL Client Cleanup Timeout} -m {$Workstation Listener\Min Handlers} -M
{$Workstation Listener\Max Handlers} -x {$Workstation Listener\Max Clients per
Handler} -c {$Workstation Listener\Tuxedo Compression Threshold} -p
{LO_WSL_PORT} -P {HI_WSL_PORT}"
{UNIX}
```

Jolt Servers

The Jolt Listener (JSL) listens for incoming connections from the Java servlet (see Listing 12-30). The Jolt Repository server (JREPSRV) translates Java classes to Tuxedo service requests (discussed in Chapter 2).

The `-W` parameter on JREPSRV permits the `jrepository` database to be updated. There is no reason for a PeopleSoft customer to do this. The relationships between Java classes and Tuxedo services are closely bound to the coding of PeopleTools. Leaving the `jrepository` database writable is a security risk, and the `-W` parameter should be removed.

Listing 12-30. *Jolt Listener and Jolt Repository server definition*

```
{JOLT}
#
# JOLT Listener and Rep Server
#
```

```
JSL             SRVGRP=JSLGRP
                SRVID=200
                CLOPT="-A -- {TUXDEV} -n {$JOLT Listener\Address}:{$JOLT
Listener\Port} -m {$JOLT Listener\Min Handlers} -M {$JOLT Listener\Max Handlers}
-I {$JOLT Listener\Init Timeout} -j {$JOLT Listener\Client Connection Mode} -x
{$JOLT Listener\Max Clients per Handler} {Jolt Encryption} {Jolt Client Cleanup
Timeout} -c {$JOLT Listener\Jolt Compression Threshold} -w JSH"

JREPSVR         SRVGRP=JREPGRP
                SRVID=250
                CLOPT="-A -- -W -P {$PS_SERVDIR}{FS}jrepository"
{JOLT}
```

Jolt Relay Adapter (JRAD) Server

The Jolt Relay Adapter (JRAD), shown in Listing 12-31, was introduced in PeopleTools 7.5 to permit the Java client applet to be served from a web server on a different node from the application server. Applets can only make socket connections back to the web server that served them. The relay adapter is used to pass the requests from the web server node to the application node. Thus a web server could be placed in a DMZ, and a port could be opened on the firewall through which requests could be passed to the application server.

The PIA does not require JRAD. The servlet can connect to any node.

Listing 12-31. *Jolt Internet Relay server*

```
{JRAD}
#
# JOLT Internet Relay (Back End)
#
JRAD            SRVGRP=JSLGRP
                SRVID=2501
                CLOPT="-A -- -l {$JOLT Relay Adapter\Listener Address}:{$JOLT Relay
Adapter\Listener Port} -c {$JOLT Listener\Address}:{$JOLT Listener\Port}"
{JRAD}
```

Tuxedo SERVICES Section

The SERVICES section of a Tuxedo file, shown in Listing 12-32, does not define which services are available; rather, it is a method of setting attributes on those services. The list of services that can be advertised by particular servers is hard-coded into the server when it is compiled, although the services actually advertised are controlled by the server command-line options, as seen earlier on PSAPPSRV.[3] Server processes can advertise services that are not described in the SERVICES section. For example, the service `RenRequest` is advertised on PSAPPSRV, but it does not appear in `psappsrv.ubb` unless PSRENSRV is configured.

3. A server can also dynamically advertise servers, as is the case on startup of the Application Engine server (PSAESRV) in the Process Scheduler in PeopleTools 8.4 (see Chapter 14).

Listing 12-32. *Tuxedo SERVICES section*

```
*SERVICES
...
ICPanel         SRVGRP=APPSRV
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"
...
ICQuery         SRVGRP=APPSRV
                LOAD=50 PRIO=50
{QUERYSRV}
                SVCTIMEOUT={$PSQRYSRV\Service Timeout}
{QUERYSRV}
{!QUERYSRV}
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
{!QUERYSRV}
                BUFTYPE="ALL"
...
{RENSRV}
# This timeout determines how long PSRENSRV will wait to boot.
RenRequest      SRVGRP=APPSRV
                LOAD=50 PRIO=50
                SVCTIMEOUT=30
                BUFTYPE="ALL"
# Service on PSRENSRV.
RenDummySvc     SRVGRP=RENGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"
{RENSRV}
...
```

A load and priority is defined for each of the services. These values affect load balancing, spawning of servers, and dequeuing of requests, all of which are discussed in the next chapter. PeopleSoft gives all services the same load and priority. These parameters come into play only if there are multiple queues that can handle the same service and load balancing is enabled in the resources section.

A service timeout variable is defined in `psappsrv.cfg` for each server type. When a separate PSQRYSRV server is defined, the `ICQuery` service is no longer advertised on the PSAPPSRV, but on the PSQRYSRV, and the service timeout on the service is taken from the PSQRYSRV section instead.

PS_ENVFILE Section

The PS_ENVFILE section, shown in Listing 12-33, is used by PeopleSoft to generate the server process environment file. The file created by `ubbgen` is a Tuxedo format file, but this section is

a PeopleSoft creation. It is read by each server process as it starts, and it sets environment variables for that server process. The substitutions continue to be made in the same way. You can see that there are different path settings for Windows and additional library paths for Unix.

Listing 12-33. *Server environmental settings definition*

```
# ------------------------------------------------------------------------
*PS_ENVFILE
TM_RESTARTSRVTIMEOUT=30
TM_RESTARTSRVTIMEOUTKILL=Y
{WINDOWS}
PATH={$PS_HOME}\bin\server\winx86;{$PS_HOME}\bin\server\winx86\interfacedrivers;
{$Domain Settings\Add to PATH};{$PS_HOME}\jre\bin\client;
{$PS_HOME}\verity\{VERITY_OS}\{VERITY_PLATFORM}\bin
INFORMIXSERVER={$Startup\ServerName}
#Set IPC_EXIT_PROCESS=1 to use ExitProcess to terminate server process.
#Set IPC_TERMINATE_PROCESS=1 to use TerminateProcess to terminate server process.
#If both are set, TerminateProcess will be used to terminate server process.
#IPC_EXIT_PROCESS=1
FLDTBLDIR32=$TUXDIR\udataobj
FIELDTBLS32=tpadm
IPC_TERMINATE_PROCESS=1
{WINDOWS}
{UNIX}
LD_LIBRARY_PATH={$LD_LIBRARY_PATH}:{$PS_HOME}/verity/{VERITY_OS}/{VERITY_PLATFORM}/bin
LIBPATH={$LIBPATH}:{$PS_HOME}/verity/{VERITY_OS}/{VERITY_PLATFORM}/bin
SHLIB_PATH={$SHLIB_PATH}:{$PS_HOME}/verity/{VERITY_OS}/{VERITY_PLATFORM}/bin
COBPATH={$PS_HOME}/cblbin
PATH={$PS_HOME}/bin:{$Domain Settings\Add to PATH}:
   {$PS_HOME}/verity/{VERITY_OS}/{VERITY_PLATFORM}/bin
FLDTBLDIR32=$TUXDIR\udataobj
FIELDTBLS32=tpadm
INFORMIXSERVER={$Startup\ServerName}
{UNIX}
```

psappsrv.cfg

This file specifies most of the configuration of the application server domain. It is used in two ways:

- It is combined with `psappsrv.ubx` by `ubbgen` to produce `psappsrv.ubb` and `psappsrv.env`. Variables that appear in `psappsrv.ubx` are used as part of the Tuxedo configuration, such as the minimum and maximum number of instances of each server process.
- Variables that do not appear in `psappsrv.ubx` are read directly by the PeopleSoft application server processes, for example `EnableDBMonitoring`.

The configuration file is broken into sections. The variables are referenced in `psappsrv.ubx` by section and variable name. The various sections are documented in PeopleBooks, so I will only comment on some parameters in the sections that follow.

Startup

The Startup section, shown in Listing 12-34, specifies how the PeopleSoft application server processes connect to the database. `Servername` is not used on Oracle.

■ **Tip** It is a good idea to create a PeopleSoft Operator ID that has only the privilege to start the application server and nothing else. The password then has no other use than to start the application server. The passwords can be encrypted in the interactive configuration dialog.

Listing 12-34. *Startup section of an application server configuration file*

```
 [Startup]
;============================================================================
; Database Signon settings
;============================================================================
DBName=HR88
DBType=ORACLE
UserId=PSAPPS
UserPswd=PSAPPS
ConnectId=people
ConnectPswd=peop1e
ServerName=
```

Database Options

The Database Options section, shown in Listing 12-35, controls how the application servers connect to the database.

Listing 12-35. *Database Options section of an application server configuration file*

```
 [Database Options]
;============================================================================
; Database-specific configuration options
;============================================================================

SybasePacketSize=
; Please see Chapter "Tuning and Administration", in
; Oracle Installation and Administration Guide for details
UseLocalOracleDB=0
;ORACLE_SID=
; Dynamic change allowed for EnableDBMonitoring
EnableDBMonitoring=1
```

If UseLocalOracleDB is set to 1, the connect string used by the application servers will not include the TNS service name (which is the same as the DBName). The server processes will attempt to make a direct shared-memory connection to the database. This will work only if the application server and database are on the same node, and the Oracle Server ID environment variable, ORACLE_SID, is set to the name of the database instance.

The same effect can be obtained by defining the TNS_SERVICE with an IPC key, as shown in Listing 12-36.

Listing 12-36. *Extract from* tnsnames.ora

```
HR88 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(Key = ora92))
    )
    (CONNECT_DATA =
      (SID = HR88)
      (ORACLE_HOME = d:\oracle\ora92)
    )
  )
```

I prefer this arrangement because it does not rely upon the ORACLE_SID variable. If you have application servers on the database server and on another node, then the application server configuration can remain the same on both.

EnableDBMonitoring is described in Chapter 9.

Security

In PeopleTools 7.*x*, a database user was created for every PeopleSoft operator. The 2-tier login checks that the operator password matches the password to the schema as well as the password stored in operator definition table (PSOPRDEFN). Setting the parameter Validate Signon with Database to 1 causes the application server to do the same when an operator signs in.

In PeopleTools 8, this parameter still enables the same functionality, but the database schemas are no longer created or maintained by PeopleSoft for each user. They have been replaced with ConnectID, described in Chapter 3. If you enable this parameter, user logins will fail unless you manually create the corresponding database accounts. The Validate Signon parameter (see Listing 12-37) should never be enabled in PeopleTools 8.

Listing 12-37. *Part of the Security section of a configuration file*

```
[Security]
;============================================================================
; Security settings
;============================================================================
; Use the database for additional signon authentication.
; The default value is 0 (DB signon validation OFF).
Validate Signon with Database=0
```

Workstation and Jolt Listeners

The two listener processes are very similar, so I will deal with them together. Their configurations are shown in Listing 12-38.

Listing 12-38. *Workstation and Jolt Listener settings*

```
[Workstation Listener]
;============================================================================
; Settings for Workstation Listener
;============================================================================
;Address Note: Can be either Machine Name or IP address.
;Address Note: %PS_MACH% will be replaced with THIS machine's name
Address=%PS_MACH%
Port=8800
Encryption=0
Min Handlers=2
Max Handlers=3
Max Clients per Handler=10
Client Cleanup Timeout=10
Init Timeout=5
Tuxedo Compression Threshold=5000

 [JOLT Listener]
;============================================================================
; Settings for JOLT Listener
;============================================================================
;Address Note: Can be either Machine Name or IP address.
;Address Note: %PS_MACH% will be replaced with THIS machine's name
Address=%PS_MACH%
Port=8810
Encryption=0
Min Handlers=2
Max Handlers=7
Max Clients per Handler=10
Client Cleanup Timeout=10
Init Timeout=5
Client Connection Mode=ANY
Jolt Compression Threshold= 2147483647
```

The delivered PeopleSoft default configuration files have 40 clients per handler in both the WSL and JSL. The Tuxedo default parameter for the multiplexing factor (which corresponds to what PeopleSoft calls "clients per handler") is 10. By reducing the number of clients per handler, more WSH and JSH processes will be required, and more return queues to the handler processes will be created. This will reduce contention on the return queue.

The total number of clients that can connect to the application server via the WSL is the maximum number of handlers multiplied by the multiplexing factor. If you reduce the number of clients per handler, you should increase the maximum number of handlers by the same factor so that the listener will support the same number of concurrent connections.

The timeout parameter is expressed in minutes (all other temporal Tuxedo parameters are expressed in seconds). The PeopleSoft default is 60 minutes before an inactive thread is disconnected. I would suggest reducing this to as little as 8 to 10 minutes.[4] If the user returns after the application server session has been timed out, then the servlet or Windows client will silently re-authenticate to the application server and continue working. The user may experience a momentary pause, but will not receive any error message. If the timeout is unrealistically large, then the system will have to be configured to support more concurrent sessions than are actually needed.

Any message larger than the compression threshold (expressed in bytes) that is sent across the network between the client (either Windows or servlet) and the handler (either WSH or JSH) will be compressed. In a good WAN environment, compression is unlikely to be beneficial.

Three-tier clients connected across a WAN may benefit from message compression. However, it is advisable to check with the network administrators whether compression is enabled at a hardware (router) level. It is then at least a waste of CPU resources, if not counterproductive, to have double compression.

The servlet and the application server should always be located physically close to each other, ideally on the same low latency network. Compression is not appropriate in this case. There are two options:

- Set a very large compression threshold so that no messages are large enough to be compressed. The maximum permitted value is 2,147,483,647 bytes.
- Compression is only enabled if the compression threshold is specified, so remove the `-c` parameter from the JSL section of `psappsrv.ubx`.

Domain Settings

This section contains some general settings for the application server, as shown in Listing 12-39.

Listing 12-39. *Domain settings*

```
[Domain Settings]
;===============================================================================
; General settings for this Application Server.
;===============================================================================
;-------------------------------------------------------------------------------
; TUXEDO Domain for this Application Server (8 characters or less).
;
Domain ID=HR88_1
;-------------------------------------------------------------------------------
; Additional directories for the Application Server's PATH environment
; variable. This should include the location of the database DLLs.
;
```

4. Research at one customer site showed that if a user doesn't return to his PeopleSoft session within about 10 minutes, then he is unlikely to come back for a much longer period.

```
Add to PATH=d:\oracle\ora92\bin
;------------------------------------------------------------------------
; Location of TUXEDO and PeopleSoft Application Server log files for this
; Application Server.
;
;Log Directory=%PS_SERVDIR%\LOGS
```

In general, the domain ID should be set to the name of the database. However, if you have multiple application servers on the same PeopleSoft database, it is advisable to give each one a different domain ID. When you enable `DBMonitoring`, the domain ID will appear in the column V$SESSION.CLIENT_INFO, and you will easily know to which application server the session relates.

If, particularly on a Windows server, you have multiple Oracle homes, you may wish to ensure that a particular application server uses SQL*Net from a particular Oracle home by explicitly setting the path in the application server configuration.

If you wish to move the application server logs to a different directory, uncomment this parameter and set it accordingly.

Trace

Various trace types can be enabled for the entire server in the configuration file, as shown in Listing 12-40. Trace should be enabled only in a limited test environment. There is a significant overhead to just trace SQL activity. PeopleCode trace can be even heavier.

Masks can be used to restrict the traces that can be enabled, particularly by the user via the PIA.

Listing 12-40. *Trace settings*

```
[Trace]
;============================================================================
; Server Trace settings
;============================================================================

;----------------------------------------------------------------------------
; SQL Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - SQL statements
; 2         - SQL statement variables
; 4         - SQL connect, disconnect, commit and rollback
; 8         - Row Fetch (indicates that it occurred, not data)
; 16        - All other API calls except ssb
; 32        - Set Select Buffers (identifies the attributes of columns
;             to be selected).
; 64        - Database API specific calls
; 128       - COBOL statement timings
; 256       - Sybase Bind information
```

```
; 512       - Sybase Fetch information
; 1024      - SQL Informational Trace
; 4096      - Manager information
; 8192      - Mapcore information
; Dynamic change allowed for TraceSql and TraceSqlMask
TraceSql=3
TraceSqlMask=12319

;------------------------------------------------------------------------
; PeopleCode Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - Trace Evaluator instructions  (not recommended)
; 2         - List Evaluator program        (not recommended)
; 4         - Show assignments to variables
; 8         - Show fetched values
; 16        - Show stack
; 64        - Trace start of programs
; 128       - Trace external function calls
; 256       - Trace internal function calls
; 512       - Show parameter values
; 1024      - Show function return value
; 2048      - Trace each statement in program  (recommended)
; Dynamic change allowed for TracePC and TracePCMask
TracePC=0
TracePCMask=4095
```

Cache Settings

The cache settings parameters disappear completely in the PeopleTools 8.20 and 8.43 configuration files, but they have reappeared as comments in 8.44, as shown in Listing 12-41.

Listing 12-41. *Application server cache settings*

```
 [Cache Settings]
;============================================================================
; Settings for managed object caching;
; Default EnableServerCaching=2, ServerCacheMode=0
; Default CacheBaseDir=%PS_SERVDIR% if defined else %PS_HOME/<domain name>/cache
; You can change these values by uncommenting and setting to the desired value
;============================================================================
;----------------------------------------------------------------------
; EnableServerCaching -
;  0        Server file caching disabled
;  1        Server file caching limited to most used classes
;  2        Server file caching for all types
;EnableServerCaching=2
;----------------------------------------------------------------------
```

```
; CacheBaseDir = the base file cache directory
;CacheBaseDir=%PS_SERVDIR%
;------------------------------------------------------------------------
; ServerCacheMode
;  0      One file cache directory per app server process
;  1      One file cache directory per domain (shared file cache, needs to be preloaded)
;ServerCacheMode=0
;------------------------------------------------------------------------
; Deprecated cache settings : MaxInMemoryObjects
;------------------------------------------------------------------------
```

The default value for server caching in PeopleTools 8.1 configuration files was 1, so only some PeopleTools objects were cached in the application server. All versions of PeopleTools 8.*x* usually benefit from caching all PeopleTools object types.

Remote Call

Remote call is a method of synchronously initiating batch COBOL programs on the application server from the PIA, rather than scheduling processes. It is mainly used for functionality such as online voucher edit and post processing in the Financials product. These batch processes can take typically take around 10 to 30 seconds to execute, even in a well-tuned system. When the application server process initiates the COBOL process, it is in the middle of executing a service routine that must wait for the COBOL process to complete. Meanwhile, it cannot handle any other service. Therefore, there is a risk that if there are many remote calls, and their performance degrades, that all the PSAPPSRV processes could be blocked and the application server will appear to hang until a remote call finishes and a service completes.

Up to PeopleTools 7.*x*, when using the Windows three-tier client there is a dedicated Tuxedo service, `RemoteCall`, that handles this. It is possible to move this service to a separate set of servers on a different queue. However, from PeopleTools 8 this is no longer possible because in the PIA the remote call is made by the generic `ICPanel` service, so you cannot distinguish between ordinary component operations and remote calls.

The `RCCBL Redirect` parameter, shown in Listing 12-42, controls whether the COBOL process generates a trace file on the application server.

Listing 12-42. *Remote call trace setting*

```
;============================================================================
; Settings for RemoteCall
;============================================================================

;----------------------------------------------------------------------------
; RemoteCall child process output redirection
;
; If this parameter is non-zero, the child process output is saved to
; <Domain Settings\Log Directory>\<program>_<oprid>.out, and any error
; output is saved to <program>_<oprid>.err.
; By default, the output is not saved
;
RCCBL Redirect=0
```

PeopleSoft Server Processes

There is a section in psappsrv.cfg for most of the various types of PeopleSoft application server processes. The section for PSAPPSRV is shown in Listing 12-43.

Listing 12-43. *Configurable parameters for the PSAPPSRV server*

```
 [PSAPPSRV]
;===========================================================================
; Settings for PSAPPSRV
;===========================================================================

;---------------------------------------------------------------------------
; UBBGEN settings
Min Instances=1
Max Instances=3
Service Timeout=300

;---------------------------------------------------------------------------
; Number of services after which PSAPPSRV will automatically restart.
; If the recycle count is set to zero, PSAPPSRV will never be recycled.
; The default value is 5000.
; Dynamic change allowed for Recycle Count
Recycle Count=5000

; Max Fetch Size -- max result set size in KB for a SELECT query
; Default is 5000KB. Use 0 for no limit.
Max Fetch Size=5000
```

The server definitions contain a number of parameters:

- Some specify the minimum and maximum number of processes, although there are only single instances of some servers.
- Service timeouts are the maximum amount of time that a service request can spend either on the queue to the application server or active on the server process before it is timed out by Tuxedo. The timeout parameters appear in the SERVICES section of psappsrv.ubx.
- Some server processes recycle after handling the number of server processes specified. This is essentially a tactic to hide memory leaks and relinquish memory allocated to the server process.
- The maximum fetch size controls the amount of data that can be fetched by the SqlRequest or SqlQuery services executed by PSAPPSRV, PSQCKSRV, or PSQRYSRV. When a query is submitted from the PIA, the entire result set is returned to the servlet in a single Tuxedo message, although only part of it may be displayed.

psappsrv.val

If you use the interactive configuration dialog in psadmin, your responses can be validated according to the specification in the validation file, psappsrv.val, shown in Listing 12-44.

Listing 12-44. *Extract from* `psappsrv.val`

```
 [Startup]
DBName={string,8}
DBType={string,8}(DB2ODBC,DB2UNIX,INFORMIX,MICROSFT,ORACLE,SYBASE)
...
[Workstation Listener]
Port={int}(1025-65536)
...
```

The code in Listing 12-44 checks that:

- `DBName` is a string of up to eight characters.
- `DBType` is a string of up to eight characters and must be one of the values in the list.
- `Port` must be a number between 1,025 and 65,536.

psappsrv.ubb

This file, the beginning of which is shown in Listing 12-45, is one of the outputs from `ubbgen`. All the variables in `psappsrv.ubx` have been resolved to literal values. This file is then compiled with the Tuxedo `tmloadcf` utility.

Listing 12-45. *Beginning of* `psappsrv.ubb`

```
########################################################################
#
# This is a skeletal TUXEDO configuration file - "psappsrv.ubb" designed
# to be used for PeopleTools 7.5 app server and the Remote Call mechanism.
# To configure additional resources, machines, servers, services, etc.
# please refer to "ubbconfig" in section 5 of the TUXEDO System Reference
# Manual.
#
########################################################################

*RESOURCES
IPCKEY          55698        # ( 32768 < IPCKEY < 262143 )
MASTER          "GO-FASTER-3"
DOMAINID        HR88_55698
MODEL           SHM
LDBAL           N
...
```

At the end of `psappsrv.ubb` is a comment section that reports the values of all the `ubbgen` variables referenced in `psappsrv.ubx` and the values that were substituted when it was generated, as shown in Listing 12-46. Any variables that you add, such as `Trace\TuxedoServiceTrace`, will also be reported.

Some variables in `psappsrv.cfg` are not referenced in `psappsrv.ubx` but are read directly by the server processes. They do not appear in this report.

Listing 12-46. *Configuration variables and values substituted in* psappsrv.ubb

```
#*********************************************************************
#  ubbgen substitution values:
#
# [ 0]:                                          {IPCKEY}:  58660
# [ 1]:                                            {MACH}:  GO-FASTER-3
# [ 2]:                          {$Domain Settings\Domain ID}:  HR88
# [ 3]:                                     {MAXSERVERS}:  MAXSERVERS       31

# [ 4]:                                   {MAXSERVICES}:  MAXSERVICES      336

# [ 5]:                                      {$TUXDIR}:  D:\ps\bea\Tuxedo8.1
# [ 6]:                                   {$PS_SERVDIR}:  D:\ps\hr88\appserv\HR88
# [ 7]:                                {$TUXCONFIG}:  D:\ps\hr88\appserv\HR88\PSTUXCFG
# [ 8]:                                    {LOGDIR}:  D:\ps\hr88\appserv\HR88\LOGS
# [ 9]:                                         {FS}:  \
# [10]:                                    {ENVFILE}:  psappsrv.env
# [11]:                                        {UID}:  0
# [12]:                                        {GID}:  0
# [13]:                                  {MAXWSCLIENTS}:  MAXWSCLIENTS=100

# [14]:                                  {MAXACCESSERS}:  MAXACCESSERS=131

# [15]:                       {$Domain Settings\Restartable}:  Y
# [16]:                          {$Trace\TuxedoServiceTrace}:  -r
# [17]:                                       {CFGFILE}:  psappsrv.cfg
# [18]:                            {$PSAPPSRV\Min Instances}:  1
# [19]:                            {$PSAPPSRV\Max Instances}:  3
# [20]:                             {$PSAPPSRV\Spawn Server}:  -p 1,600:1,1
# [21]:                            {$PSSAMSRV\Min Instances}:  1
# [22]:                            {$PSSAMSRV\Max Instances}:  2
# [23]:                       {$Workstation Listener\Address}:  //GO-FASTER-3
# [24]:                          {$Workstation Listener\Port}:  8800
# [25]:                   {$Workstation Listener\Encryption}:  0
# [26]:                  {$Workstation Listener\Init Timeout}:  5
# [27]:                          {WSL Client Cleanup Timeout}:  -T 10
# [28]:                  {$Workstation Listener\Min Handlers}:  2
# [29]:                  {$Workstation Listener\Max Handlers}:  3
# [30]:      {$Workstation Listener\Max Clients per Handler}:  10
# [31]: {$Workstation Listener\Tuxedo Compression Threshold}:  5000
# [32]:                                     {LO_WSL_PORT}:  8801
# [33]:                                     {HI_WSL_PORT}:  8803
# [34]:                                         {TUXDEV}:
# [35]:                               {$JOLT Listener\Address}:  //GO-FASTER-3
# [36]:                                  {$JOLT Listener\Port}:  8810
# [37]:                           {$JOLT Listener\Min Handlers}:  2
# [38]:                           {$JOLT Listener\Max Handlers}:  7
```

```
# [39]:                        {$JOLT Listener\Init Timeout}:  5
# [40]:              {$JOLT Listener\Client Connection Mode}:  ANY
# [41]:             {$JOLT Listener\Max Clients per Handler}:  10
# [42]:                                    {Jolt Encryption}:
# [43]:                        {Jolt Client Cleanup Timeout}:  -T 10
# [44]:          {$JOLT Listener\Jolt Compression Threshold}:  1000000
# [45]:                          {$PSAPPSRV\Service Timeout}:  300
# [46]:                          {$PSSAMSRV\Service Timeout}:  300
# [47]:                          {$PSQCKSRV\Service Timeout}:  300
# [48]:                                          {$PS_HOME}:  D:\ps\hr88
# [49]:                       {$Domain Settings\Add to PATH}:  d:\oracle\ora92\bin
# [50]:                                       {VERITY_OS}:  winx86
# [51]:                                   {VERITY_PLATFORM}:  _nti40
# [52]:                                {$Startup\ServerName}:
#*********************************************************************
```

The values of the special variables are also reported, as shown in Listing 12-47.

Listing 12-47. *Special variables and values reported in* `psappsrv.ubb`

```
#*********************************************************************
#  ubbgen control values:
#
# [ 0]:              {UNIX}:  FALSE
# [ 1]:             {!UNIX}:  TRUE
# [ 2]:           {WINDOWS}:  TRUE
# [ 3]:          {!WINDOWS}:  FALSE
# [ 4]:            {PUBSUB}:  FALSE
# [ 5]:           {!PUBSUB}:  TRUE
# [ 6]:          {QUICKSRV}:  FALSE
# [ 7]:         {!QUICKSRV}:  TRUE
# [ 8]:          {QUERYSRV}:  FALSE
# [ 9]:         {!QUERYSRV}:  TRUE
# [10]:              {JOLT}:  TRUE
# [11]:             {!JOLT}:  FALSE
# [12]:              {JRAD}:  FALSE
# [13]:             {!JRAD}:  TRUE
# [14]:            {DBGSRV}:  FALSE
# [15]:           {!DBGSRV}:  TRUE
# [16]:            {OPTENG}:  FALSE
# [17]:           {!OPTENG}:  TRUE
# [18]:            {RENSRV}:  FALSE
# [19]:           {!RENSRV}:  TRUE
# [20]:               {MCF}:  FALSE
# [21]:              {!MCF}:  TRUE
# [22]:              {PPM}:  FALSE
# [23]:             {!PPM}:  TRUE
#*********************************************************************
```

psappsrv.env

The Tuxedo server environment file, shown in Listing 12-48, is generated by `ubbgen` and is read by each application server process when it starts. The environmental variables specified are set for the application server session.

Listing 12-48. `psappsrv.env`*: Tuxedo server environment file*

```
TM_RESTARTSRVTIMEOUT=30
TM_RESTARTSRVTIMEOUTKILL=Y
PATH=D:\ps\hr88\bin\server\winx86;D:\ps\hr88\bin\server\winx86\interfacedrivers;
d:\oracle\ora92\bin;D:\ps\hr88\jre\bin\client;D:\ps\hr88\verity\winx86\_nti40\bin
INFORMIXSERVER=
#Set IPC_EXIT_PROCESS=1 to use ExitProcess to terminate server process.
#Set IPC_TERMINATE_PROCESS=1 to use TerminateProcess to terminate server process.
#If both are set, TerminateProcess will be used to terminate server process.
#IPC_EXIT_PROCESS=1
FLDTBLDIR32=$TUXDIR\udataobj
FIELDTBLS32=tpadm
IPC_TERMINATE_PROCESS=1
```

Not all the values in the environment file need to be fully resolved to literals. It is legitimate to dynamically reference other environmental variables, such as `$TUXDIR` in this example.

Configuration Template Files

A number of template files are delivered in $PS_HOME/appserv as a part of the PeopleTools installation. When a domain is created, you are asked to choose a template (see Listing 12-49), and the appropriate template files are copied to the domain directory.

Listing 12-49. *Configuration template menu*

```
Please enter name of domain to create :HR88NEW

Configuration templates:

  1) developer
  2) large
  3) medium
  4) small


Select config template number:
```

There are four template files: `developer.cfx`, `large.cfx`, `medium.cfx`, and `small.cfx`. One of these is copied to `psappsrv.cfg` in the domain directory when the domain is created depending on the choice made in the preceding menu.

The small, medium, and large configuration templates are intended for use with production application server domains of different sizes. They differ only in the number of servers configured. The developer template has only two application message servers instead of the usual six, as discussed earlier (see Listing 12-27).

There are also two UBX template files. If you choose a developer template, `developer.ubx` is copied to the domain directory; otherwise, `psappsrv.ubx` is copied.

New versions of the template files are delivered by PeopleTools upgrades, but existing domains are not affected. Any changes to the template need to be migrated into existing domains. I usually create new domains, compare the new and old configuration files, and resolve any differences manually with a file comparison tool.

Tuxedo Administration Console

The Tuxedo Administration Console, shown in Figure 12-2, is a Java applet that runs in a web browser. It provides a useful graphical overview of the structure of a Tuxedo domain, and it permits remote administration of BEA Tuxedo applications. It also provides some basic remote monitoring of Tuxedo domains.

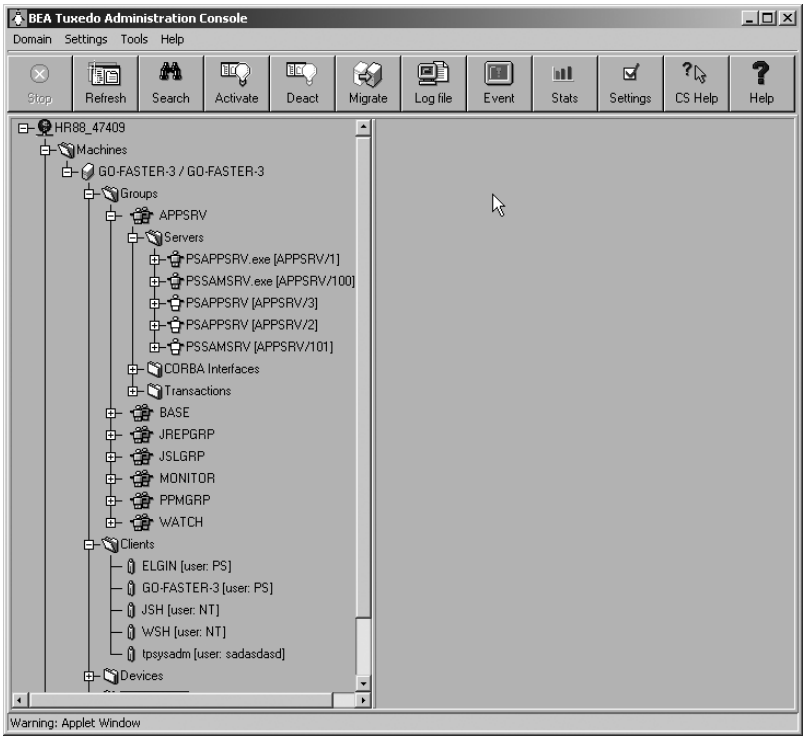


Figure 12-2. *Tuxedo Administration Console*

I won't go into the functionality of the Tuxedo Administration Console in detail here because it is covered thoroughly in the Tuxedo documentation, which is delivered with the product in `%TUXDIR%\help\guiguide` (where `%TUXDIR%` is the location where Tuxedo was installed), and it is also available on the BEA website (http://e-docs.bea.com/tuxedo/tux81/index.htm).

The principal advantage of the Tuxedo Administration Console is that it permits remote administration of the Tuxedo domain via a web browser. It is possible to stop and restart either an entire domain or individual application server processes.

The console also provides a workaround for a known incompatibility between Tuxedo and Windows 2000 Terminal Services that prevents application server processes from starting when `psadmin` is run from a Terminal Services window.[5]

In addition, the Tuxedo log file can be viewed from the console, which can be useful when a Windows-based application server is behind a firewall and shared files cannot accessed.

The Tuxedo documentation covers the generic installation process in detail, but in the following sections I will explain how to configure the console specifically for PeopleSoft, and PeopleSoft for the console.

Configuring the BEA Administration Console for PeopleSoft

In order to use the Tuxedo Administration Console, you need to run a web server to serve up the Java applet and a listener process to handle communications from the applet once it is running on the client browser, as I explain in the following sections.

Tuxedo Console Listener (wlisten)

The Tuxedo Console Listener process receives incoming connections from console applets and starts a console gateway process (`wgated`). The listener is started with the command shown in Listing 12-50, and options are read from the parameter file, `webgui.ini`.

Listing 12-50. *Batch script to start the Tuxedo Console Listener*

```
net start "BEA ProcMGR V8.1"
set TUXDIR=D:\ps\bea\Tuxedo8.1
%TUXDIR%\bin\wlisten -i %TUXDIR%\udataobj\webgui\webgui.ini
```

The initialization file specifies the address and port to which `wlisten` listens for incoming connections, which by default is configured as 4003. At the bottom of the file is a list of the known Tuxedo domains. Domains can also be specified in the console, which will then update `webgui.ini`. In this case (see Listing 12-51), HR88 is an application server, and PSNT is a Process Scheduler.

Listing 12-51. `%TUXDIR%\udataobj\webgui\webgui.ini`*: Tuxedo Console Listener configuration file*

```
# Web GUI initialization file.
# Created 18-Aug-04 18:44:28 by BEA software installation program.
#
```

5. See PeopleSoft Support Solution ID: 702869, "E-AS: LIBTUX_CAT: 681 ERROR Failure to create message queue, reported on Windows 2000 when starting Tuxedo Application Server."

```
TUXDIR=d:\ps\bea\Tuxedo8.1
INIFILE=d:\ps\bea\Tuxedo8.1/udataobj/webgui/webgui.ini
NADDR=//GO-FASTER-3:4003
DEVICE=/dev/tcp
CODEBASE=/java
DOCBASE=http://go-faster-3:80/doc
FOREGROUND=N
SNAPDIR=d:\ps\bea\Tuxedo8.1/udataobj/webgui/java/snapshot
SNAPBASE=/java/snapshot
#
# In order to configure one or more domains as part of the Web GUI pull-down
# menu, add lines to this file of the form DOMAIN=domainname;tuxconfig
DOMAIN=HR88;D:\ps\hr88\appserv\HR88\PSTUXCFG
DOMAIN=PSNT;D:\ps\hr88\appserv\prcs\HR88\PSTUXCFG
```

The listener process will run as a background process unless FOREGROUND is set to Y. The default is N.

Tuxedo Web Server (tuxwsvr)

BEA provides a simple web server (see Listing 12-52) for use with the Tuxedo Administration Console, although you could use any HTTP server.

Listing 12-52. *Command to start the BEA web server*

```
%TUXDIR%\bin\tuxwsvr.exe -l //go-faster-3:80 -F -i %TUXDIR%\udataobj\tuxwsvr.ini
```

By default, the web server will run as a background process unless the `-F` parameter is specified. This option is useful during testing and until the console is functioning correctly.

The web server initialization file, shown in Listing 12-53, maps the virtual served directories to directories in the physical file system.

Listing 12-53. `%TUXDIR%\udataobj\tuxwsvr.ini`*: Tuxedo web server configuration file*

```
# tuxwsvr initialization file.
# Created 18-Aug-04 18:44:28 by BEA software installation program.
#
CGI    /cgi-bin   d:\ps\bea\Tuxedo8.1/udataobj/webgui/cgi-bin
HTML   /java      d:\ps\bea\Tuxedo8.1/udataobj/webgui/java
HTML   /doc       d:\ps\bea\Tuxedo8.1/help
HTML   /          d:\ps\bea\Tuxedo8.1/udataobj/webgui
```

Starting the Console

After you start the listener and web server processes, you can access the console from `webguitop.html` in the root directory of the web server. To continue the example, the URL for this page is http://go-faster-3:80/webguitop.html. This page presents a button that when clicked runs the CGI gateway to the console, `tuxadm` (http://go-faster-3/cgi-bin/tuxadm.exe?TUXDIR=d:\ps\bea\Tuxedo8.1). You are then presented with the login dialog shown in Figure 12-3.

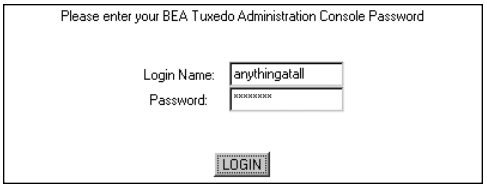


Figure 12-3. *BEA Tuxedo Administration Console applet screen*

The login name is totally ignored by the console—any input or none at all will be accepted. Only the password is validated against the password that you entered when you installed Tuxedo, which is held, unencrypted, in `%TUXDIR%\udataobj\tlisten.pw`, as shown in Listing 12-54.

Listing 12-54. `tlisten.pw`*: Tuxedo password file*

```
password
```

If the login is successful, you are taken to the screen shown in Figure 12-2.

Configuring PeopleSoft for the BEA Administration Console

PeopleSoft application server processes rely on two environmental variables, `PS_SERVDIR` and `PS_SERVER_CFG`. These variables specify the location of PeopleTools configuration and cache files for a domain, and they are usually set by the `psadmin` utility. However, if you start either a whole domain or just a single server process via the BEA console, `psadmin` will not be used.

The solution is to set the environmental variables in the server environment file, `psappsrv.env`. As described earlier in this chapter, this file is generated by `ubbgen` from the Tuxedo template file, `psappsrv.ubx`. The variables can be specified in the template file as shown in Listing 12-55.

Listing 12-55. *Extract from* `psappsrv.ubx`*: Server environment variables for the Tuxedo Administration Console*

```
#rem added in order to start domain with Tuxedo Console
PS_SERVDIR={$PS_SERVDIR}
PS_SERVER_CFG={$PS_SERVDIR}{FS}{CFGFILE}
```

The `PS_SERVDIR` environmental variable is set by `psadmin` when the domain is configured, so the literal value of the variable is put into `psappsrv.env` (see Listing 12-56).

Listing 12-56. *Extract from* `psappsrv.env`*: Server environmental variables for the Tuxedo Administration Console*

```
#rem added in order to start domain with Tuxedo Console
PS_SERVDIR=D:\ps\hr88\appserv\HR88
PS_SERVER_CFG=D:\ps\hr88\appserv\HR88\psappsrv.cfg
```

This advice also applies to the Process Scheduler when run under Tuxedo.

Summary

Chapter 2 explained some of the basic principles of a Tuxedo application server, and this chapter explained how the configuration files are related. It also demonstrated how to configure the Tuxedo Administration Console.

You should now know how to make configuration changes to application servers should the need arise. In the next chapter, you will look at how to size an application server and how some of the parameter settings can impact performance.

CHAPTER 13

Tuning the Application Server

Chapters 2 and 12 discussed the underlying architecture of and how to configure the application server. They provide a basis for this chapter, in which I discuss some of the issues that can affect application server performance, including sizing the domain to have an appropriate number of application server processes, setting operating system kernel parameters, and load balancing.

In general, there is little you can do in the application server to improve the performance of a PeopleSoft system, but there are some things that you should consider in order to prevent the application server from becoming a performance bottleneck. Sometimes, application server problems are a symptom of other underlying issues, including database performance problems.

PeopleSoft's official recommendations are set out in its Red Paper titled "Online Performance Configuration Guidelines," which can be obtained from the Customer Connection website.

Sizing

The sizing of a PeopleSoft application server can significantly affect the performance of the system as a whole. The use of too many application servers or processes can have just as significant and detrimental an effect as the use of too few.

If you configure too few servers, they may not be able to keep up with the demands made by the users, and you will see requests queuing in the application server. Configure too many server processes and you could either place an excessive load on the database server, or consume all the CPU on the application server, or cause contention in the application server. All of these could, again, give rise to queuing in the application server.

If the application server is co-resident with the database, and the application server consumes all the CPU, you can expect to see significant degradation of database performance.

The separate query servers put very little load on the application server node, but pass additional, relatively heavy, queries through to the database. Permitting too many concurrent report queries will degrade database performance.

You can think of the application server as being like a tap on a pipe. You can open or close the tap to regulate the flow through the pipe. Similarly, you can configure more or fewer application server processes to control the load on the application and database servers.

Spawning

Before looking at how many server processes to configure, it is important to understand the concept of *server spawning* in Tuxedo. A server process is defined in the Tuxedo configuration file (see Listing 13-1) as having a minimum and maximum number of instances. When the domain is started, Tuxedo will start the minimum number of processes and will spawn additional instances on demand, up to the maximum number permitted.

Listing 13-1. *Definition of PSAPPSRV in* `psappsrv.ubb`

```
PSAPPSRV        SRVGRP=APPSRV
                SRVID=1
                MIN=1
                MAX=3
                RQADDR="APPQ"
                REPLYQ=Y
                CLOPT="-p 1,600:1,1 -s@..\psappsrv.lst -s@..\psqcksrv.lst -sICQuery
-sSqlQuery:SqlRequest -- -C psappsrv.cfg -D HR88 -S PSAPPSRV"
```

Demand is defined in terms of the number of requests on the queue that leads to the search, or their cumulative load. The situation is like a line in a post office or bank, where one line leads to many windows (behind each of which sits an agent). If there is sufficient demand, additional windows are opened, and if there are idle agents, a window might be closed.

Spawning in Tuxedo is controlled by the `-p` parameter, which is of the form

```
-p[L][low_water][,[terminate_time]][:[high_water][,create_time]]
```

The Tuxedo documentation (`servopts(5)`) states that if the number of services on the queue[1] exceeds the `high_water` level for at least `create_time` seconds, a new server is spawned. If the load drops below `low_water` for at least `terminate_time` seconds, a server is deactivated. If the `L` parameter is specified, the load of the services on the queue is compared to the `low_water` and `high_water` thresholds.

In the PeopleSoft application server, the spawning parameter is set to `-p 1,600:1,1` for all servers. This means that if a queue of one or more requests exists continuously for 1 second, then another server will be spawned (as long as the number of servers is less than the maximum number). Conversely, if there is no queuing for 600 seconds, and there are more than the minimum number of servers, one will be shut down.

However, the spawning algorithm is dependent on the duration services.[2] The logic that Tuxedo uses to check whether or not a new server should be spawned is in the Tuxedo libraries that are linked into the server process executables. Each server process autonomously determines whether to spawn another server, and not the WSH of JSH processes that enqueue the service request, nor the BBL that spawns the restart server, which actually starts the new process.

After processing a message from the queue, a server will execute the code that does the server-spawning management. The first time a server process detects that the high-water mark

1. Tuxedo 8.1 documentation talks about "requests per server" on the queue.
2. See the Tuxedo Support Solution S-01484: CR018998, "Server spawning algorithm is dependent on long-running service time."

is exceeded, it will save the current time. It then processes the next message on the queue. If, when that message has completed, the high-water mark is still exceeded, and the difference between the current time and the saved time exceeds the create time, then a new server is spawned.

Hence the time taken to detect that the high-water mark has been exceeded may be the time taken for a server process to execute two services. This can lead to some scenarios in which Tuxedo will be slow to spawn additional servers:

- It would be a mistake to configure a domain with a minimum of only one PSAPPSRV server and to rely on spawning to initiate sufficient server processes. If the only PSAPPSRV process was occupied with a long-running service, and a queue of requests built up during that time, no server would spawn until that service completed. It is not unusual for some operations to take a long time; for example, an online voucher edit and post process on a Financials system might easily take 60 seconds for the remote call to the COBOL program to complete. So it might take a server process more than 1 minute to decide to spawn an extra server.
- The PSQRYSRV server executes ad hoc queries that often run for long periods of time. Therefore the servers will test for queuing less frequently. If all the servers on a queue are occupied, and a queue of further requests builds up, no additional server process could be spawned until a query service completes.
- Many PeopleSoft sites use a number of small servers to host application servers, rather than a single large server. These smaller servers will rarely have more than four CPUs. Therefore, as explained in the discussion in the following sections, the application server domains will usually run four to eight PSAPPSRV processes. Smaller domains can be slower to spawn because there are fewer servers. Therefore, the minimum number of servers configured should be adequate to cope with typical loads, and less reliance should be placed on Tuxedo spawning.

Too Few Server Processes?

If you have too few server processes to handle the incoming requests, then you will see requests build up on the queues in the application server. The following sections cover a couple of methods that can be used to determine a reasonable minimum number of server processes for a system.

Concurrency

It is possible to determine how many service requests execute concurrently. The Tuxedo service trace will report the start and end time of each service to an accuracy of 1/100 of a second on Unix systems and 1/1,000 of a second on Windows. Hence you can determine how many services were being processed when each service started. I usually do this by loading the Tuxedo service trace into a database table (as described in Chapter 9 in the section "Tuxedo Service Trace (-r Option)") and using PL/SQL to calculate the concurrency. Then I can present that information graphically. If you have several application servers, you can combine the service traces to determine the concurrency across all application servers.

Listing 13-2 shows the PL/SQL code that will calculate the concurrency of each request.

Listing 13-2. *PL/SQL to calculate Tuxedo service request concurrency*

```
DECLARE
   l_count INTEGER := 0;
   CURSOR   c_txrpt IS
   SELECT   *
   FROM     txrpt t
   WHERE    concurrent = 0
   ORDER BY queue, stime desc, etime desc, stimestamp desc;
   p_txrpt c_txrpt%ROWTYPE;
BEGIN
   OPEN c_txrpt;
   LOOP
      FETCH c_txrpt into p_txrpt;
      EXIT WHEN c_txrpt%NOTFOUND;

      UPDATE txrpt a
      SET    concurrent = (
             SELECT COUNT(*)
             FROM   txrpt b
             WHERE  b.stime <= a.stime
             AND    b.etime >= a.stime
             AND    b.stimestamp <= a.stimestamp
             AND    b.stimestamp >= a.stimestamp
                    - ceil((a.etime-a.stime)/1000+1)/86400
             AND    b.queue = a.queue
             )
      WHERE service = p_txrpt.service
      AND   pid = p_txrpt.pid
      AND   stimestamp = p_txrpt.stimestamp
      AND   stime = p_txrpt.stime
      AND   etime = p_txrpt.etime
      ;

      IF l_count <= 1000 THEN
         l_count := l_count + 1;
      ELSE
         l_count := 0;
         COMMIT;
      END IF;
   END LOOP;
   COMMIT;
   CLOSE c_txrpt;
END;
/
```

This SQL is not particularly efficient. It requires an index on the table to satisfy the UPDATE statement.

```
CREATE index txrpt2 ON txrpt(queue,stime,etime,stimestamp);
```

If you are working with a large volume of trace data, then range-partitioning the table on `stime` can be advantageous.

Calculating concurrency shows how many server processes you actually need. If you achieve improvements in performance, the number of server processes that you need may decrease, thus saving resources on both the application and database servers. The graph shown in Figure 13-1 depicts the concurrency of PSAPPSRV requests on a system on two different days. Between these two dates a number of improvements were made to database performance, and so as the service time fell, the number of services that executed concurrently also fell.

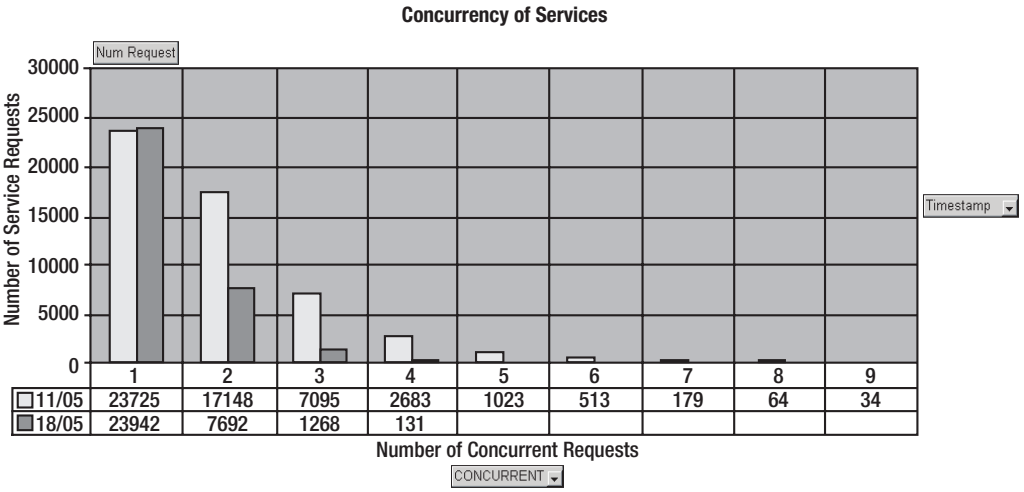


Figure 13-1. *Concurrency of Tuxedo services*

On May 11 it would have been appropriate to configure a minimum of five PSAPPSRV processes. This would have ensured that there was a free server process for 98.5% of requests. By May 18, three PSAPPSRV processes would have been enough for 99.6% of requests, and the system never needed more than four processes.

Application Server Monitoring

Another effective and entirely pragmatic way to assess the required minimum number of server processes is to monitor the number of application server processes and see whether Tuxedo spawns additional server processes, and if so, how many processes and how frequently they should spawn.

It is possible to use `tmadmin`, the Tuxedo command-line administrative utility, within Unix scripts to poll the application server and collect metrics (as described in Chapter 9). The graph in Figure 13-2 was produced from data collected from the `tuxmon.sh` script.[3]

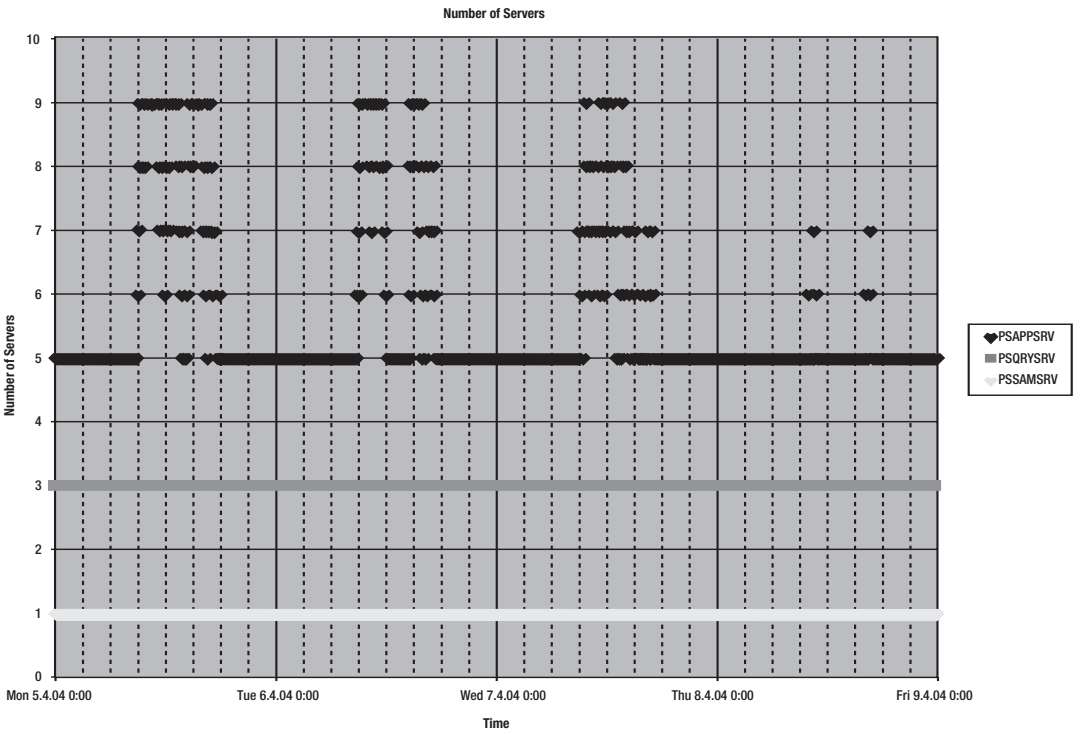


Figure 13-2. *Number of application server processes*

This system is configured with a minimum of five application server processes and a maximum of nine. On most days, and for much of the working day, Tuxedo spawned additional servers and sometimes all available servers. It might be worthwhile to increase the minimum number of server processes to at least seven. The `tuxmon.sh` script also collects queuing information (not shown), and it does not show any queuing on the system, so nine server processes is probably enough.

Too Many Server Processes?

There are a number of situations in which you may decide that you have too many server processes configured on a PeopleSoft system. The sections that follow describe some of the symptoms you should look out for.

3. The `tuxmon.sh` script can be downloaded from www.go-faster.co.uk. It collects Tuxedo performance metrics and sends them back by e-mail for analysis.

Database Overload

Having too many application server processes can put more load on the database server. I have seen some systems, with a very large amount of online activity, in which all the application server processes were occupied handling existing requests, so new requests were starting to build up on the queues in the application server. In one case, the customer responded by repeatedly adding more physical application servers, but did not obtain any improvement in performance. The reason for this was that, in fact, the application server processes were mostly waiting for responses from the database. In other words, the database had become the bottleneck.

PeopleSoft's ad hoc Query utility can be a particularly significant cause of database overhead. It is a facility that is sold to prospective customers as a major functional benefit. Users can easily build queries to extract data from the system themselves, rather than have a predefined report developed for them. However, they can also easily generate queries that perform very poorly. There are a couple of common scenarios in which this can happen:

- A query scans large tables and generates a large quantity of physical reads. This leads to contention on the disk subsystem, and the performance of the database as a whole degrades.
- A query uses nested loop joins with an index range scan. The table is accessed by ROWID. There is virtually no physical I/O, but a huge amount of logical I/O, which in turn consumes a huge amount of CPU and can result in significant latch contention on the cache buffer chains.

The situation can be aggravated by the user resubmitting the query, should it time out. The Oracle shadow process can continue to process the first query, and now you have two (or more, for especially insistent users) sessions consuming resources.

In most cases, I would suggest that the correct response to either scenario is to reduce the number of application servers to a level where the database can cope with the load, and accept that requests will build up on the application server queues. In particular, you should restrict the number of query server processes and other batch reporting activity to preserve online activity. Then, you can address the causes of poor database performance, thus increasing throughput through the existing server processes and perhaps permitting more process to be configured.

Running Out of CPU

When executing PeopleCode, the PSAPPSRV processes are heavy consumers of CPU. The PeopleSoft Ping case studies (see Chapter 10) show that the application server service time is closely related to CPU speed. If there is no free CPU available to an application server, then processes will have to wait to run on a processor, and this will degrade PeopleCode performance.

In terms of the overall performance of a system, it is better for the number of application server processes to be restricted such that the CPU on the application server is almost, but not quite, fully utilized. Thus the PSAPPSRV processes will not contend for CPU, and the application server service time will not increase due to CPU contention. If the number of application server processes is increased, the same amount of CPU may have to be shared between more concurrently executing processes.

■ **Note** Put simply, once the CPU is fully utilized, it is better for application server service requests to wait on a Tuxedo IPC queue than to contend with other service requests for CPU and end up on the operating system run queue.

When there is a CPU shortage, increasing the number of application server processes can cause increased contention on the shared memory queues. When a request is either enqueued or dequeued by an application server process, it needs to acquire an exclusive lock on the queue. This can cause additional contention (see the next section, "Multiple Queues").

The following steps are a method for estimating the number of application server processes that you can use before the CPU will be fully utilized:

1. On a Unix server, collect CPU statistics with the `sar` (system activity reporter) or `sadc` (system activity data collector) command, and generate a simple tabular report. On Windows, use the Performance Monitor to collect CPU statistics that can be exported to a comma-separated file. I would suggest collecting the information on a 1-minute cycle.
2. Use the Tuxedo service trace to report the time and duration of each Tuxedo service.
3. Load both sets of statistics into a database (using SQL*Loader, as described in Chapter 9). From this, you can determine how much service time was expended in each 1-minute sample for which you have CPU statistics.
4. Produce a graph of CPU utilization against Tuxedo service time (see Figure 13-3).

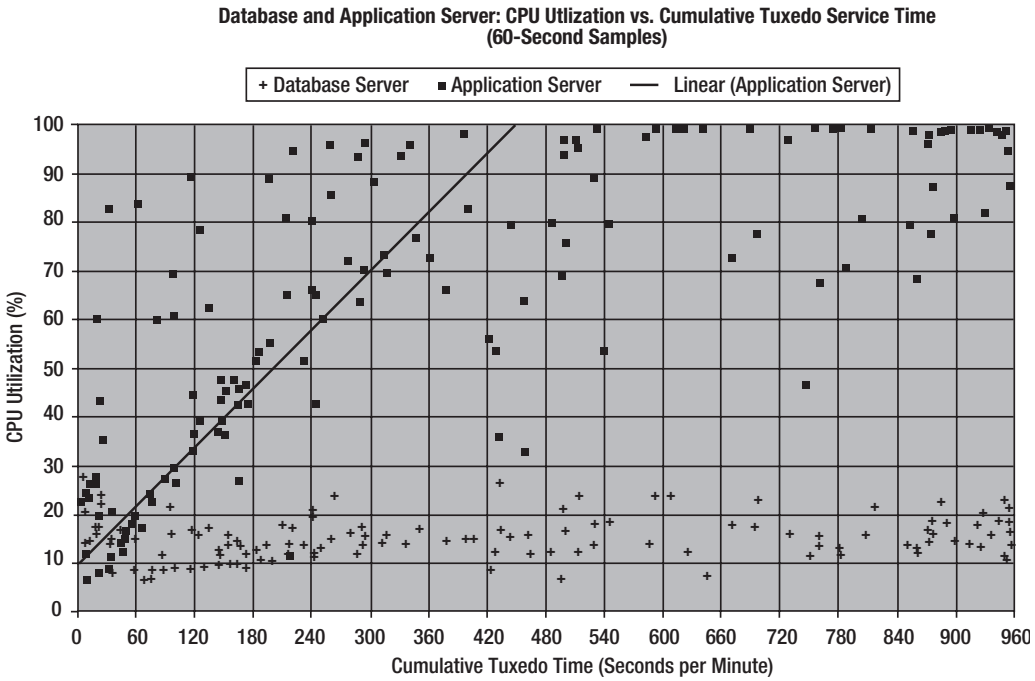


Figure 13-3. *CPU utilization vs. Tuxedo service time*

The graph shows the results from a system where the database and application server were on separate physical servers. CPU for both servers has been plotted against Tuxedo service time. The database CPU utilization rarely exceeds 20%, while the application server is fully CPU utilized at about 450 seconds of Tuxedo service time per minute. The application server clearly becomes the bottleneck first.

The solid line on the graph is a linear approximation to the application server CPU utilization up to the point at which the CPU on the application server is fully utilized. As noted, it reaches 100% CPU utilization at around 450 seconds of Tuxedo service time per minute.

There are also some data points that indicate up to 960 seconds per minute of service time. These are consistent with long-running SQL queries, where the database is active and the application server is waiting for a response and consuming less CPU.

Four hundred and fifty seconds (which is 7.5 minutes) of Tuxedo service time per minute suggests that 7.5 permanently concurrently actively PSAPPSRV processes would fully consume all the CPU on the application server. Therefore, the absolute maximum number of PSAPPSRV processes that should be used for this physical application server node is seven. That estimate makes no allowances for any other activity on that server, in which case the maximum number of application server processes may need to be further reduced.

Multiple Queues

Application server processes on a single queue will contend because it is necessary to briefly take out a lock on the queue in order to enqueue or dequeue a request. The contention will increase as the number of servers increases, and by the time there are ten servers, the contention can become significant. Therefore, BEA recommends that you do not have more than ten server processes per queue. This is likely to be necessary only on large, highly active online systems (usually CRM or self-service HR), and where the application servers have sufficient CPUs to support more than ten PSAPPSRV processes.

One option to avoid overloading a queue would be to simply configure a second application server on the same node. The servlet would randomly connect to one server or the other, and the load would roughly balance between the servers. The downside, of course, is that there would be two servers to maintain, administer, and monitor. There would also be two sets of Tuxedo processes to run.

An alternative would be to configure a second queue for the same type of server process. If, for example, you needed to run up to sixteen application server processes, you could have two queues serving PSAPPSRV processes, each with up to eight processes. To create the additional queue, it is only necessary to add a second PSAPPSRV section in `psappsrv.ubx`, as shown in Listing 13-3. I have removed the "prompted" variables and the alternative CLOPT lines for clarity, but they should also be duplicated.

Listing 13-3. *Two APPQs queues configured in* `psappsrv.ubx`

```
#
# PeopleSoft Application Server
#
PSAPPSRV        SRVGRP=APPSRV
                SRVID=1
                MIN={$PSAPPSRV\Min Instances}
                MAX={$PSAPPSRV\Max Instances}
                RQADDR="APPQ1"
```

```
                REPLYQ=Y
                CLOPT="{$Trace\TuxedoServiceTrace} -e {LOGDIR}{FS}APPQ.stderr
{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst -s@..{FS}psqcksrv.lst -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSAPPSRV"
#
# PeopleSoft Application Server
#
PSAPPSRV        SRVGRP=APPSRV
                SRVID=11
                MIN={$PSAPPSRV\Min Instances}
                MAX={$PSAPPSRV\Max Instances}
                RQADDR="APPQ2"
                REPLYQ=Y
                CLOPT="{$Trace\TuxedoServiceTrace} -e {LOGDIR}{FS}APPQ.stderr
{$PSAPPSRV\Spawn Server} -s@..{FS}psappsrv.lst -s@..{FS}psqcksrv.lst -- -C {CFGFILE}
-D {$Domain Settings\Domain ID} -S PSAPPSRV"
```

There are two changes in the second PSAPPSRV server definition:

- The server ID must be unique for each server process within a group, so I have added 10 to the server ID, because there should never be more than 10 server processes on a queue.
- The queue has been given a unique name of APPQ2. This will be visible in the `tmadmin` interface.

All configuration variables have been used for both servers. This ensures that the two queues have the same configuration and handle approximately the same load. The maximum number of instances of PSAPPSRV, in `psappsrv.cfg`, now means the number of server processes per queue. So, to have 16 PSAPPSRV processes, `Max Instances` must be set to 8.

The additional queues and servers require additional resources to be allocated in the Tuxedo configuration. The sizes of the Bulletin Board internal tables are determined by the Tuxedo parameters `MAXACCESSERS`, `MAXSERVERS`, and `MAXSERVICES`. The values for these parameters are calculated by `ubbgen` (see Chapter 12), but `ubbgen` will not take the second queue into account. The values must be calculated manually, and I recommend replacing the existing special variables in `psappsrv.ubx` with new configuration variables, as shown in Listing 13-4, which are then defined in `psappsrv.cfg`.

Listing 13-4. *Replacing special variables in* `psappsrv.ubx`

```
*RESOURCES
...
#{MAXSERVERS}
#{MAXSERVICES}
```

```
MAXSERVERS      {$Domain Settings\MaxServers}
MAXSERVICES     {$Domain Settings\MaxServices}
...
# ------------------------------------------------------------------

*MACHINES
"{MACH}" LMID="{MACH}"                          # Machine name must be uppercase
...
#        {MAXACCESSERS}
        MAXACCESSERS={$Domain Settings\MaxAccessers}
```

The literal values can then be seen in `psappsrv.ubb` (see Listing 13-5).

Listing 13-5. *Corresponding extract from* `psappsrv.ubb`

```
 [Domain Settings]
...
;---------------------------------------------------------------------------
; Manually defined Tuxedo domain sizing parameters
; required because multiple APPQs defined
MaxServers=37
MaxServices=563
MaxAccessers=137
```

Kernel Configuration

Whether you are running Tuxedo on Unix or Windows, you need to consider the kernel parameters that control the amount of IPC resources that are available to the system.

Tuxedo makes extensive use of IPC queues. IPC resources are global to the entire machine and are shared by all Tuxedo domains. It is necessary to make sure there are sufficient resources to support all the queues that are created by all the application server and Process Scheduler domains (as described in Chapter 2 in the section "IPC Resources"). Otherwise, you will get run-time errors when Tuxedo can't create additional queues when starting additional servers.

On Unix systems, you may have to take other applications into account when determining kernel parameters. On Windows, the kernel is simulated by the Tuxedo IPC helper service, so it is only necessary to consider the Tuxedo domains.

On Windows, BEA provides a Control Panel utility to configure Tuxedo. One of the tabs defines IPC resources, as shown in Figure 13-4.

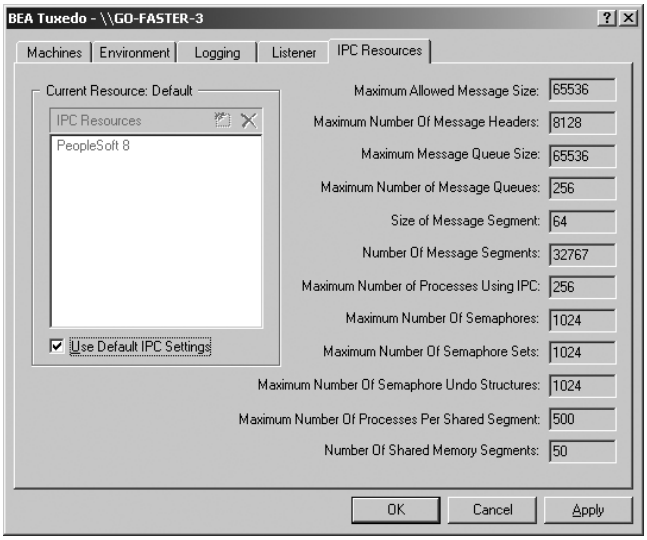


Figure 13-4. *Tuxedo IPC configuration on Windows*

Most of these parameters correspond directly to Unix kernel parameters. The mapping is described in Table 13-1.

Table 13-1. *IPC Resource Name Mappings Between Windows and Unix*[4]

Windows Name	Traditional Unix Name
Maximum Allowed Message Size	MSGMAX
Maximum Number Of Message Headers	No matching name
Maximum Message Queue Size	MSGMNB
Maximum Number of Message Queues	MSGMNI
Size of Message Segment	MSGSSZ
Number Of Message Segments	MSGSEG
Maximum Number of Processes Using IPC	NPROC
Maximum Number Of Semaphores	SEMMNS
Maximum Number Of Semaphore Sets	SEMMNI
Maximum Number Of Semaphore Undo Structures	SEMMNU
Maximum Number Of Processes Per Shared Segment	No matching name
Number Of Shared Memory Segments	SHMMNI

4. From the Tuxedo 8.1 documentation.

■ **Note** The Tuxedo 8.1 documentation makes recommendations for setting these parameters in the sections "IPC Resource Configuration on a UNIX System" and "Configuring BEA Tuxedo ATMI for Windows 2000."

IPC Queue Sizing

The following warning appears in all versions of the Tuxedo manual, and it applies to all operating systems:

> *If the limit specified by any of [the IPC] parameters is exceeded, then a blocking condition occurs. There is one exception to this rule: MSGMAX. Messages that exceed 75 percent of MSGMNB, or that are larger than MSGMAX, are placed in a UNIX file. A very small message containing the filename is then sent to the recipient. Because this mode of operation results in a severe reduction in performance, [BEA] strongly recommend that you avoid it.*[5]

When this occurs, especially in a highly active online system, the time taken to enqueue or dequeue requests increases because a physical disk access is required, leading to increased contention on the queues.

It is possible to see evidence of this phenomenon on some Unix systems. The temporary files are created in directories whose names begin with `tx`, located in the temporary directory, but sometimes Tuxedo does not delete all of these directories.

So the next question is, just how big are Tuxedo messages in PeopleSoft? The Tuxedo log can be enhanced to show every Tuxedo function call by either setting an environmental variable, `TMTRACE=on`, or by issuing the `tmadmin` command `changetrace on`. This additional information is intended to assist debugging by the Tuxedo developer, but it does show the sizes of messages sent to and from the application server processes.

Table 13-2 shows the functions that are of interest in a PeopleSoft system. There are other functions, and they are explained in more detail in the Tuxedo documentation.

Table 13-2. *Tuxedo Service Functions*

Function	Description
`tpalloc`	This function allocates memory to a buffer returning a pointer to it. The last parameter is the number of bytes requested.
`tprealloc`	This function changes the size of an existing buffer, usually to extend it. The pointer to the memory and the new buffer size are passed as parameters.
`tpfree`	This function releases the memory from use as a buffer. The pointer allocated by `tpalloc` is passed in as a parameter.

(Continues)

5. See Tuxedo Documentation ➤ Installing the Tuxedo System ➤ IPC Resource Configuration on a Unix System (http//e-docs.bea.com/tuxedo/tux81/install/insappd.htm).

Table 13-2. *Tuxedo Service Functions (Continued)*

Function	Description
tpservice	This function is the call for the service routine. The first parameter of the structure is the name of the service, and the fourth parameter is the size of the message passed to the server.
tpreturn	This function returns the result from a service routine. The fourth parameter is the size of the message returned from the service.

The size of the messages passed within the application server varies with the PeopleTools release, the complexity of the PIA components (or panel groups), and the amount of application data involved. Listing 13-6 shows an example of the additional trace information in a Tuxedo log. It shows a single ICPanel service call. The incoming message is 10,817 bytes in length. The service returns a message of 49,514 bytes.

Listing 13-6. *Sample of a Tuxedo log with trace information*

```
225340.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:  { tpservice({"ICPanel", 0x0,
0x27a20e0, 10817, 0, -1, {1082411157, 0, 29}})
225340.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    { tpalloc("CARRAY", "", 8192)
225340.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    } tpalloc = 0x35e0c48
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    { tprealloc(0x35e0c48, 65536)
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    } tprealloc = 0x4a0d020
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    { tprealloc(0x4a0d020, 131072)
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    } tprealloc = 0x4af40e0
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    { tpreturn(2, 0,
0x4af40e0, 49514, 0x0)
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:    } tpreturn [long jump]
225451.GO-FASTER-3!PSAPPSRV.3884.1060.0: TRACE:at:  } tpservice
```

It is possible to analyze this trace and determine the number and sizes of the messages.

■ **Caution** Enabling this trace can have a noticeable impact on performance. It should not usually be used for long periods. It generates a lot of trace information very quickly. I generally find that a sample from a few minutes of activity is sufficient to demonstrate the message sizes.

There are two approaches to analyzing the trace file. The Unix shell and awk script in Listing 13-7 can be used to extract the distribution of message sizes.

Listing 13-7. msglen.sh*: Script to analyze Tuxedo message sizes*

```
TUXLOG=$PS_HOME/appserv/DEVP8/LOGS
for i in $TUXLOG/TUXLOG.[0-1][0-9][0-3][0-9][0-9][0-9]
  do
    FILE=`basename $i`
    echo "$FILE \c"
    egrep -h "tpservice\(|tpreturn\(" $i | \
awk '{
        elements=split($4,var1,"(");
        if(var1[1]=="tpservice") {
            dir="c->s"
            elements=split($4,var1,"\"");
            service=var1[2]
        }
        if(var1[1]=="tpreturn") {
            dir="s->c"
        }
        elements=split($7,var1,",")
        size=var1[1]
        print dir, service, size
}' | \
    sort -k3n,3 | \
    awk '{
        lineno++
        totsize+=$3
        print lineno, $1, $2, $3, totsize
    }' | \
    sort -rn | \
    awk 'BEGIN{
       printf("Message direction,Service name,Message size (bytes),")
       printf("Proportion of messages not larger than this message,")
       printf("Proportion of traffic in messages not larger than this message\n")
    }{
        if(lines==0) {
            lines=$1
            totsize=$5
            sizeleft=totsize
        }
        if($4>0) {
            printf("%s,%s,%d,%f,%f\n", $2, $3, $4, $1/lines, $5/totsize)
        }
    }' | \
    tee $FILE.msglen |\
    wc -l
done
```

Alternatively, the tuxlog file can be loaded into a database table, created by Listing 13-8, with SQL*Loader.

Listing 13-8. tuxlog_pre.sql

```
CREATE TABLE tuxlog
(timestamp  DATE         NOT NULL
,nodename   VARCHAR2(20) NOT NULL
,prcsname   VARCHAR2(20) NOT NULL
,prcsid     VARCHAR2(20) NOT NULL
,funcname   VARCHAR2(20) NOT NULL
     CONSTRAINT funcname CHECK (funcname IN('tpservice','tpreturn'))
,service    VARCHAR2(20) NOT NULL
,msg_size   NUMBER(8)
);
```

The FUNCNAME constraint on the table limits the rows loaded to just the two Tuxedo functions that report message sizes. By careful control of termination characters, it is possible to load the Tuxedo log directly into the database with SQL*Loader with the control file in Listing 13-9, without the need to reformat it.

Listing 13-9. tuxlog.ldr

```
LOAD DATA
INFILE 'TUXLOG.041904'
REPLACE
INTO TABLE tuxlog
FIELDS TERMINATED BY WHITESPACE
TRAILING NULLCOLS
(timestamp TERMINATED BY '.' "TO_DATE(:timestamp,'HH24MISS')"
,nodename  TERMINATED BY '!'
,prcsname  TERMINATED BY '.'
,prcsid    TERMINATED BY ':'
,dummy1    FILLER
,dummy2    FILLER
,funcname  TERMINATED BY '(' --function name
,service   TERMINATED BY ',' --"TRANSLATE('{','')"
,dummy3    FILLER TERMINATED BY ','
,dummy4    FILLER TERMINATED BY ','
,msg_size  TERMINATED BY ','
)
```

The data can then be extracted from the database into an Excel spreadsheet to produce a graph, as shown in Figure 13-5.

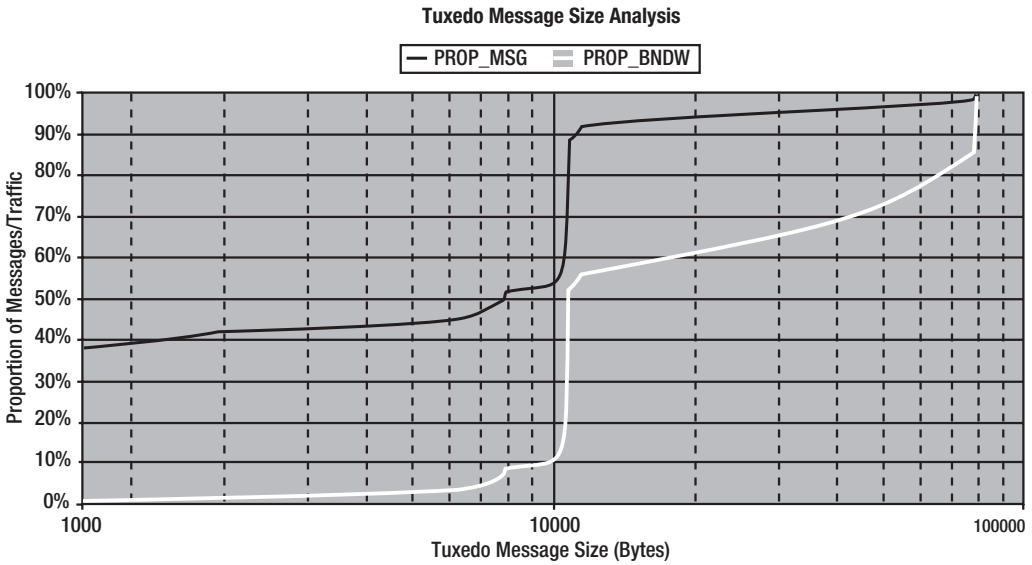


Figure 13-5. *Tuxedo message size analysis (*`tuxlog.xls`*)*

In the case of a few tests run on a demo HCM8.8/PeopleTools 8.44 database, most of the messages are under 11KB, so these would definitely fit in the default-sized 64KB queues. Just 4% of messages are over 48KB, and these account for 28% of the total message volume. You can't tell from this measurement that this is a performance problem. All you can say is that a potential problem exists.

This information can be used to justify a change to the Unix kernel parameters that control IPC queue sizing.

Recommended IPC Parameter Changes

Messages of up to 100KB are typical in PeopleTools 8.1, and messages of 70KB are common in PeopleTools 8.4. Where possible, I would suggest increasing the maximum message size to at least 256KB. The maximum queue size should be at least as large the maximum messages size and possibly higher if the system exhibits queuing.

On Windows, it is possible to save tuned sets of IPC parameters (see Figure 13-6).

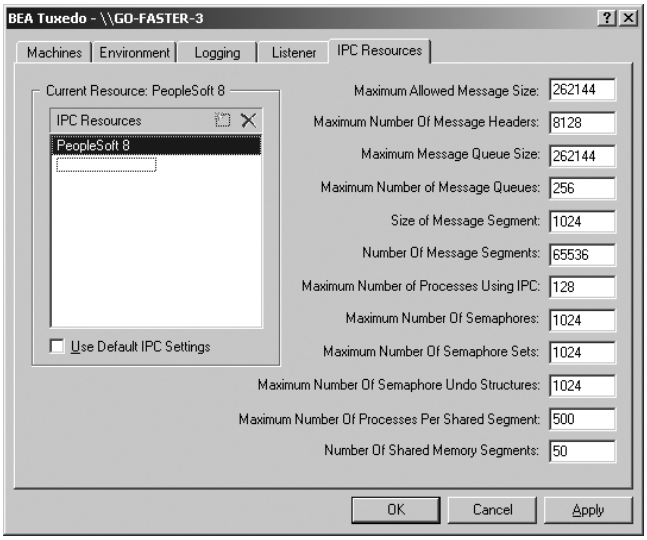


Figure 13-6. *IPC settings tuned for PeopleSoft 8*

The increase in the maximum message size increases the total amount of shared memory that could be used. Either additional message segments must be provided or the size of the message segments should be increased. I have done both in order to satisfy the following formula:

```
Maximum Number of Message Queues * Maximum Message Queue Size
     = Number Of Message Segments * Size of Message Segment
     = Total Memory overhead of Message Queues
```

■ **Note** Not all of the kernel parameters can be set on all of the supported flavors of Unix, and sometimes they cannot be set as desired. For example, on HP-UX 11, the maximum message size that can be set is 64KB, although a larger value would be desirable.

Other Tuxedo Options

In the sections that follow I describe a number of other Tuxedo features and options that PeopleSoft does not use in its delivered application server configurations. However, they can be easily incorporated and, if used appropriately, they can improve or at least preserve system performance.

Operating System Priority

On Unix systems, the operating system scheduling priority of a process can be lowered with the `nice` command. Only the superuser (root) can increase the priority. When a server has no free CPU, processes with a lower priority get less time on the CPU. Where there is free CPU, the scheduling priority does not affect the amount of CPU that the process can utilize.

When Tuxedo is run on Unix, the priority of server processes can be adjusted using the `-n` server option, as shown in Listing 13-10. The parameters to this option are simply passed through to the `nice(2)` function. Hence, this option does not work on Windows.

Listing 13-10. *Extract from* `psappsrv.ubb`

```
PSAPPSRV        SRVGRP=APPSRV
                SRVID=1
                MIN=1
                MAX=3
                RQADDR="APPQ"
                REPLYQ=Y
                CLOPT="-n 5 -p 1,600:1,1 -s@..\psappsrv.lst -s@..\psqcksrv.lst --
-C psappsrv.cfg -D HR88 -S PSAPPSRV"
```

There are a number of potential uses for the `-n` server option:

- The operating system priority of a process is inherited from its parent. The priority of the Process Scheduler running under Tuxedo could be lowered so that batch processes also run with a lower priority. In PeopleTools 8.4, the priority of the Application Engine server processes can be lowered directly. This is discussed in more detail in Chapter 14.
- If the application server is co-resident with the database server, then the application server could be run at a lower priority to prevent it from starving the database of CPU.
- A company running HR and Payroll has two websites and two application servers connected to the same database: one for self-service HR and the other for the "back-office" HR department. The self-service application server is run at a lower priority.[6]

Load Balancing

Load balancing in a Tuxedo domain is only relevant if the same service is advertised on more than one queue. This is not the case in a PeopleTools 8 application server domain,[7] unless multiple queues have been configured for the same application server, as described earlier in this chapter. Only larger systems, where more than ten PSAPPSRV processes are required, will need to consider the load balancing option.

In this section, I want to look at what Tuxedo does if there is a choice as to where a service request can be enqueued (see Figure 13-7).

6. The example is in production and has been put to the test. In the early days of the system, an e-mail was sent to all 30,000 employees telling them that they could see their pay slips online in the self-service HR application. Most of them promptly tried this. This far exceeded the design specification of the system. The self-service application server was completely overwhelmed, and performance was severely affected, but the back-office HR function was able to continue almost unaffected.
7. Until PeopleTools 7.57, the services advertised on the PSQCKSRV were also advertised on PSAPPSRV. Tuxedo could enqueue requests for the quick services on either queue.

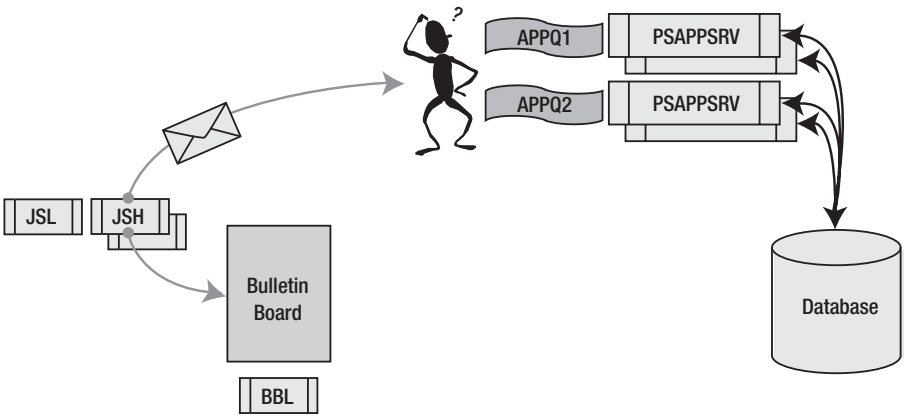


Figure 13-7. *Where to enqueue?*

If load balancing is not enabled, the request is placed on the first available queue. This can result in the queues receiving a significantly unequal load.

Load balancing is enabled in the RESOURCES section of the Tuxedo configuration file, as shown in Listing 13-11. In the delivered PeopleSoft configuration it is disabled.

Listing 13-11. *Tuxedo RESOURCES section from* `psappsrv.ubx` *with load balancing enabled*

```
*RESOURCES
IPCKEY          {IPCKEY}        # ( 32768 < IPCKEY < 262143 )
MASTER          "{MACH}"
DOMAINID        {$Domain Settings\Domain ID}_{IPCKEY}
MODEL           SHM
LDBAL           Y
```

If load balancing is enabled, then Tuxedo will place the request on a queue where there is an idle server process. If there is no idle server, then for each queue upon which the request could be enqueued, the sum of the load of each of the service requests of the queue is calculated. The request is enqueued on the queue with the lower total load.

A useful analogy is a row of supermarket checkouts. Each checkout represents a queue offering a checkout service. The shoppers with their baskets represent the messages requesting the service. When you have finished shopping, how do you choose which checkout line to join? What most people do is look at the lines and make a judgment as to which line will take the shortest amount of time to get through. To put it a different way, which queue has the least load queued on it. That judgment is more sophisticated than simply counting the number of baskets; it involves an estimate of how many items are in the baskets.

This is exactly what Tuxedo does. It uses the definition of the service in the SERVICES section of the Tuxedo configuration file to obtain the load value for the service, and so sums the loads of all the service requests in the queue.

Unfortunately, all the services in a PeopleSoft domain are delivered with a load of 50 (see Figure 13-8). That is like saying that all baskets in the supermarket take the same amount of time to get through the checkout. This is demonstrably not the case in both supermarkets and Tuxedo application servers.

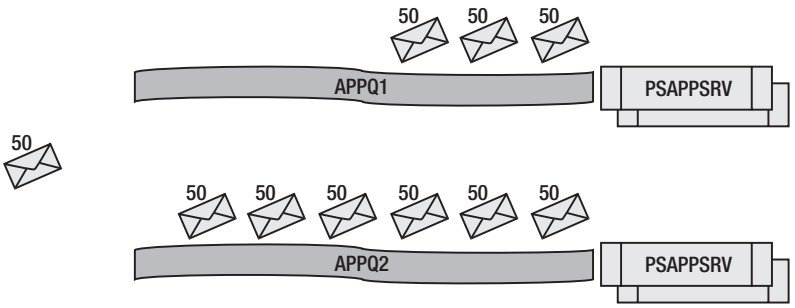


Figure 13-8. *All PeopleSoft services are delivered with the same loads.*

If you choose your supermarket checkout line simply on the basis of the number of baskets in the line, then you could get in a line behind three baskets filled to overflowing, in preference to six almost empty baskets in another line.

If load balancing is to be used, then realistic loads must be applied to the services.

■ **Note** If load balancing is enabled, you may find that the `printqueue` command in `tmadmin` incorrectly reports requests on the queue when in fact there are none.[8] This inaccuracy may have an effect on load balancing. It occurs because the shared memory–based statistics accuracy in Tuxedo defaults to "approximate." Setting it to "exact" (with the `sstats ex` command in `tmadmin`) will prevent the inaccuracy, but at the cost of a small performance hit.

Calculating Service Loads

If the Tuxedo service trace (the `-r` option described in Chapter 9) is enabled, you will get a log of every service that passes through an application server and the time it took to execute. The trace file can be processed directly with the Tuxedo `txrpt` utility to produce a summary report of performance (see Listing 13-12) that includes an average service time.

8. See the Tuxedo Support Solution S-03851: CR017477, "Shared memory–based statistics accuracy may influence the load balancing."

Listing 13-12. `txrpt` *report*

```
      SERVICE SUMMARY REPORT


SVCNAME         7a-8a       ...    14p-15p         TOTALS
                Num/Avg     ...    Num/Avg         Num/Avg
--------------- --------    ...    --------        -------
ICPanel         471/0.64    ...    490/0.36        15875/0.53
ICScript        207/0.27    ...    94/0.29         4644/0.25
PortalRegistry  120/0.09    ...    27/0.09         1758/0.08
GetCertificate  23/0.24     ...    13/0.19         625/0.18
HomepageT       30/0.27     ...    7/0.24          464/0.25
PsmChkRptAuth   3/0.01      ...    0/0.00          211/0.01
FileAttach      0/0.00      ...    0/0.00          7/0.05
ICWorkList      1/0.55      ...    0/0.00          4/0.48
--------------- -------     ...    -------         -------
TOTALS          855/0.45    ...    631/0.34        23588/0.42
```

If the load on the queue is to be a measure of how long it is likely to take to get onto the server, then the load for a service should be proportional to the average execution time. So set the service load in the SERVICES section of `psappsrv.ubx` to be the duration of the service, as shown in Listing 13-13.

Listing 13-13. *Extract from* `psappsrv.ubx`

```
PortalRegistry  SRVGRP=APPSRV
                LOAD=8 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"

ICPanel         SRVGRP=APPSRV
                LOAD=53 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"

ICScript        SRVGRP=APPSRV
                LOAD=25 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"

HomepageT       SRVGRP=APPSRV
                LOAD=25 PRIO=50
                SVCTIMEOUT={$PSAPPSRV\Service Timeout}
                BUFTYPE="ALL"
```

Figure 13-9 shows a contrived but entirely plausible scenario. The load of three `ICPanel` requests on APPQ1 is 159, but the load of six services on APPQ2 is only 154. The new request will be enqueued on APPQ2, then, because although there are more requests, their total load is lower, and the request should reach the server process more quickly.

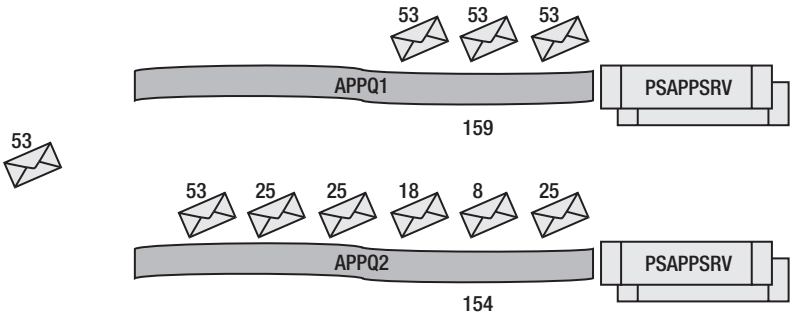


Figure 13-9. *Load balancing with realistic loads*

The load of a service can also be changed dynamically in the `tmadmin` utility with the `changeload` or `chl` command, as shown in Listing 13-14.

Listing 13-14. *Dynamically changing the load of a service in* `tmadmin`

```
> help scp
serviceparms (scp) -g groupname -i srvid -s service
-------------------------------------------------
Print the parameters associated with the specified service.

> scp -g APPSRV -i 1 -s ICPanel
    Service Name: ICPanel
   Function Name: ICPanel
            Load: 50
        Priority: 50
         Address: 0xa

> help chl
changeload (chl) [-m machine] {-q qaddress [-g groupname] [-i srvid]
        [-s service] | -g groupname -i srvid -s service |
        -I interface [-g groupname]} newload
-------------------------------------------------------------------------------
Change the load associated with the specified service or interface.

> chl -g APPSRV -i 1 -s ICPanel 53
2 entries changed.

> scp -g APPSRV -i 1 -s ICPanel
    Service Name: ICPanel
   Function Name: ICPanel
            Load: 53
        Priority: 50
         Address: 0xa
```

```
> scp -g APPSRV -i 2 -s ICPanel
    Service Name: ICPanel
   Function Name: ICPanel
            Load: 53
        Priority: 50
         Address: 0xa
```

So performance improvements can be attempted by adjusting service loads without downtime, and they can be reversed if they are not successful. If they are successful, they can be added to the domain configuration. The load can also be changed dynamically via the Administration Console.

Even though you have to specify a server in the `chp` command on which to change the load of a service, it applies to all the servers on that queue. The queue is identified in terms of the server ID.

If the service is advertised on several queues in the same group, the `scp` command will change only the load of the service for the one queue where the specified server ID resides. Note that the Tuxedo configuration permits only the service load to be specified for all the queues and servers in a group.

Service Priority

The priority of a service controls how it is dequeued by the server processes. Higher priority requests are dequeued first (see Figure 13-10). However, a lower priority message does not remain forever enqueued, because every tenth message is also retrieved from the front of the queue.

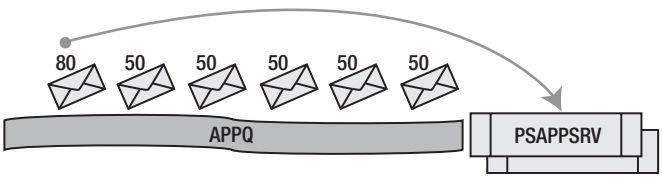


Figure 13-10. *Service dequeuing priority*

As noted previously, PeopleSoft delivers all services with a priority of 50, so it is always the request at the front of the queue that is dequeued. Adjusting service priorities is unlikely to be beneficial to configure in PeopleSoft systems, and I have included it mainly to distinguish load and priority.

A possible use might be to give `ICScript` services a higher priority so that navigation responds more quickly. A user might make have to make a number of selections to navigate the menu hierarchy before entering a panel. If a system is exhibiting queuing in the application server, menu accesses will also degrade. Menu navigation performance would be improved at the expense of component response.

The priority of a service can also be changed dynamically in the `tmadmin` utility with the `changepriority` or `chp` command, as shown in Listing 13-15, or via the Administration Console.

Listing 13-15. *Dynamically changing the priority of a service in* `tmadmin`

```
> scp -g APPSRV -i 1 -s ICScript
    Service Name: ICScript
   Function Name: ICScript
            Load: 50
        Priority: 50
         Address: 0xc

> help chp
changepriority (chp) [-m machine] {-q qaddress [-g groupname] [-i srvid]
        [-s service] | -g groupname -i srvid -s service |
        -I interface [-g groupname]} newpri
------------------------------------------------------------------------------
Change the dequeuing priority associated with the specified service
 or interface.

> chp -g APPSRV -i 1 -s ICScript 80
2 entries changed.

> scp -g APPSRV -i 2 -s ICScript
    Service Name: ICScript
   Function Name: ICScript
            Load: 50
        Priority: 80
         Address: 0xc
```

The comments made in the previous section about the scope of the `chl` command and service load also apply to the `chp` command and service priority.

Other Tips

Finally, I would like to mention a few configuration options that could make administration of the PeopleSoft application server a little easier, particularly in an environment with a requirement of high availability.

Unix User Accounts

It is common for a single physical Unix server to run several application server domains, especially in development environments. It is advisable to have each Tuxedo domain running under a different Unix user account. The IPC resources created by the domain will be owned by the Unix user running the domain, and the owner can be seen in the output of the `ipcs` command. Hence it is easy to associate IPC resources with a particular domain.

Occasionally, a Tuxedo domain can lock up and you will need to kill the IPC resources associated with that domain. PeopleSoft even provides a script, `killipc.sh`, to do that. It finds and deletes all the IPC resources owned by a particular Unix user and group.

The situation is complicated if several domains are running as the same Unix user. Although it is possible to work out which process belongs to which domain from the process's

command line that can be seen from the `ps` command, it is only possible to identify IPC resources from the process ID of the last process to access them. However, some of the IPC queues (described in Chapter 2) may not have been used by any process. It is then impossible to determine the domain to which they belong. You then have no alternative but to shut down all the Tuxedo domains running under that Unix account and kill any outstanding IPC resources. This loss of service would be bad enough in a development system, but in a production environment it could be much more serious.

Multiple Domains on Small Production Systems

If you operate with only a single application server domain in production, it is still useful to configure the servlet to reference a second server so that you can make configuration changes to the application server without requiring system downtime.

A second application server domain can be started, and then the first can be shut down and reconfigured. The users will be automatically reconnected to the surviving application server without any error message. There will be no loss of service, although the users might experience a temporary loss of performance.

Cycling the Application Server Without Shutting It Down

Occasionally (usually after an application upgrade) it may be necessary to recycle the PSAPPSRV processes in order to clear the in-memory cache of PeopleTools objects, so that the application change that was upgraded is recognized and loaded. Instead of recycling the whole application server, you can use the `stop` and `boot` commands of the `tmadmin` interface to stop and restart individual server processes. So long as you start with more than a single PSAPPSRV process, there will be no loss of service for the users, although again they may experience some loss of performance.

It is possible to automate this process with a Unix shell script, as shown in Listing 13-16.

Listing 13-16. *Extract from* `tuxcycle.sh` *showing recycling PSAPPSRV processes*

```
...
tuxcmd() {
(
$TUXDIR/bin/tmadmin -r <<! 2>/dev/null
psr
!
) | grep "PSAPPSRV" | grep "APPQ" | sort  |\
awk '
{
    printf("stop -g %s -i %s\n",$3,$4);
    printf("boot -g %s -i %s\n",$3,$4);
}'
}
...
$TUXDIR/bin/tmadmin <<! 2>&1
`tuxcmd`
!
```

The procedure `tuxcmd()` lists the PSAPPSRV processes and formats the commands with `awk`, which are then piped into another `tmadmin` session that executes them.

Reconfiguring Tuxedo with the Administration Console

You can make changes to various settings, such as service loads and priorities, in both the Tuxedo Administration Console and the `tmadmin` interface. Other settings can only be changed in the Administration Console, such as server command-line options. You can see in Figure 13-11 that I have changed the name of the Tuxedo standard error file to `APPQ_HR88.stderr2` in the executable options.

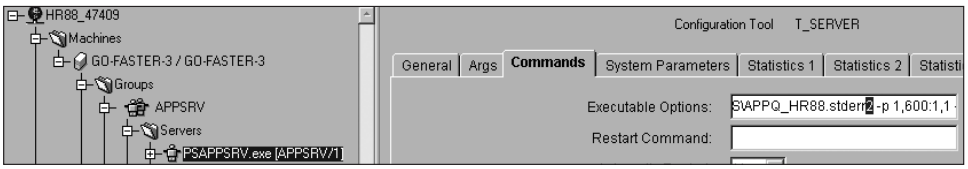


Figure 13-11. *Setting a server command line in the Tuxedo Administration Console*

■ **Caution** There is a significant difference between `tmadmin` and the Tuxedo Administration Console when making dynamic configuration changes. Changes made via `tmadmin` will persist only until the domain is shut down, and they will be lost when the domain is restarted. The Tuxedo Administration Console not only changes the current settings for the server, but also updates the compiled Tuxedo configuration file, `PSTUXCFG`—but not any other file! Therefore, the setting will persist until the domain is reconfigured in `psadmin`, at which point the change will be lost because the configuration file is compiled from `psappsrv.ubb`.

The `PSTUXCFG` file can be decompiled with the Tuxedo `tmunloadcf` utility, as shown in Listing 13-17.

Listing 13-17. `tmunloadcf` *decompiles the Tuxedo configuration file* `PSTUXCFG`.

```
rem _tmunloadcf.bat
set PS_SERVDIR=D:\ps\hr88\appserv\HR88
set TUXCONFIG=%PS_SERVDIR%\PSTUXCFG
set TUXDIR=D:\ps\bea\Tuxedo8.1
set PATH=%TUXDIR%/bin;%PATH%
%TUXDIR%\bin\tmunloadcf > tmunloadcf.txt
```

The output from `tmunloadcf` is similar to `psappsrv.ubb`, but it includes all default values and excludes any comments. The new definition for the command-line options for PSAPPSRV that were set in the console (as shown in Figure 13-11) can be seen in Listing 13-18.

Listing 13-18. *Decompiled Tuxedo configuration file*

```
"PSAPPSRV"    SRVGRP="APPSRV"    SRVID=1
   CLOPT="-r -e D:\ps\hr88\appserv\HR88\LOGS\APPQ_HR88.stderr2 -p 1,600:1,1
-s@..\psappsrv.lst -s@..\psqcksrv.lst -sICQuery -sSqlQuery:SqlRequest -- -C
psappsrv.cfg
-D HR88 -S PSAPPSRV"
   RQADDR="APPQ"
   RQPERM=0660    REPLYQ=Y    RPPERM=0660    MIN=1 MAX=3    CONV=N
   SYSTEM_ACCESS=FASTPATH
   MAXGEN=6    GRACE=60    RESTART=Y
   MINDISPATCHTHREADS=0    MAXDISPATCHTHREADS=1    THREADSTACKSIZE=0
   SICACHEENTRIESMAX="500"
```

If you want to permanently retain any configuration changes that you have made with the console, they will have to be manually transferred into `psappsrv.ubx` and `psappsrv.cfg`.

■ **Tip** It is helpful to keep a decompiled version of the Tuxedo configuration before making any changes in the console. If, having made changes, you decompile the configuration file again, you can compare the two files to identify any Tuxedo changes.

Summary

This chapter completes the final piece of the application server jigsaw. You should now be able to monitor the application server, and size and adjust it so that it is not causing a performance bottleneck. In which case, you should be left with SQL performance issues that should be easier to diagnose.

CHAPTER 14

The Process Scheduler

The Process Scheduler, also sometimes referred to as the Batch Scheduler or Batch Server, is an agent that initiates batch and report processes on a server. Although this element of PeopleSoft infrastructure is only of limited direct interest to the DBA, there are some aspects of the Process Scheduler of which you should be aware.

In this chapter, I describe the architecture and configuration of the Process Scheduler, the data that is set up when a process is scheduled to be run by an operator in the PIA. I also discuss some configuration options and administrative tasks that can prevent the overhead of the Process Schedulers and batch programs from affecting the performance of the whole system.

When a process is scheduled to run, a number of request records are inserted into various tables in the database, including PSPRCSRQST. Several schedulers can be configured on different physical servers for a single PeopleSoft database. The schedulers regularly poll the database, looking for work. The polling frequency can be configured in the scheduler definition.

A process may be scheduled to run on a specific scheduler or any scheduler that is available. From PeopleTools 8.4, process requests can be load balanced across schedulers. A process may be scheduled to run at a particular time in the future. Several processes can be run, either sequentially or in parallel, as a part of a job.

Some process types, such as nVision and Crystal Reports, can run only on Windows. Unix-based systems will nearly always require a Process Scheduler on a Windows server. Other process types, including Application Engine, SQR, and COBOL, can run on any platform.

The workload created by a Process Scheduler can be restricted. Each scheduler can be configured to concurrently run no more than a maximum number of a particular type of process (including not running that type at all) and no more than an overall maximum number of processes of any type.

As with the application server, the configuration of the Process Scheduler can be used to restrict the workload on the database.

Process Scheduler Architecture

The Process Scheduler remained virtually unchanged between PeopleTools versions 3 and 7.5. However, since the introduction of the PIA, the Process Scheduler has evolved considerably. In the next sections I want to follow that evolution.

PeopleTools 7.x

Up to version 7, the Process Scheduler was a single stand-alone process called PTPUPRCS, written in COBOL. From PeopleTools 7, it was started from within the `psadmin` utility that was also used to start the application server. Prior to that, it was simply started from a batch script.

When the Process Scheduler ran on a Windows server, the report output and log files, when not printed directly, were written to a shared drive. It was common practice to create a Unix print queue so that SQR reports could be run on a Unix server and could print to a network printer.

A compiled Process Scheduler executable was delivered by PeopleSoft for Windows, but on Unix platforms the source code had to be compiled.

■ **Note** PeopleSoft COBOL connects to the database via a C subprogram for which only the object code is supplied. Thus, it is not possible to work out from the source code how the passwords are encrypted.

PeopleTools 8.1x

For PeopleTools 8, the Process Scheduler was rewritten in C, and PeopleSoft shipped a compiled executable, now called PSPRCSRV, for each supported operating system.

With the introduction of the PIA, PeopleSoft users would not be on just the office LAN or WAN. Batch and report processes could no longer be run on the users' PCs, so PeopleSoft had to provide a mechanism to deliver report output to users. It introduced a second process, PSDSTSRV, the Report Distribution Agent. PSDSTSRV also polls the tables PSPRCSRQST and PS_CDM_LIST, which hold the scheduled requests, looking for processes that have completed and for which there are files to deliver.

Upon completion of a process initiated by the Process Scheduler, the Report Distribution Agent transfers the output files to the Report Repository, which is the directories on the web server from whence the files can be served up by the reporting servlet to the PIA client browser. These two processes always exist as a pair; every Process Scheduler needs a Report Distribution Agent. Rather confusingly, the two processes are also collectively referred to as the Process Scheduler.

The Process Scheduler and Report Distribution Agent can, optionally, be configured to run as server processes within a Tuxedo domain, hence the change in the name of the executable. This domain has no listeners, and no Tuxedo services are defined; thus no service calls are passed between the server processes. Tuxedo is merely used as a mechanism to start the processes, but it also has the merit of automatically restarting the processes in the event that they should terminate abnormally.

PeopleTools 8.4x

In PeopleTools 8.4, Tuxedo is no longer optional. The Process Scheduler has become a full-fledged Tuxedo domain, as depicted in Figure 14-1.

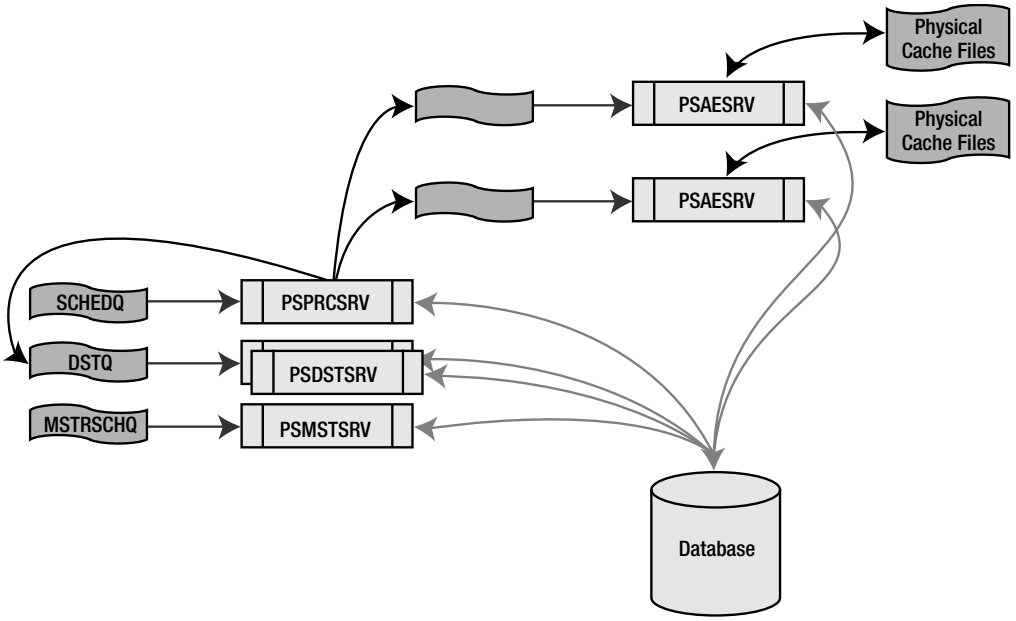


Figure 14-1. *Process Scheduler Tuxedo domain*

However, the Process Scheduler domain has some significant differences from the application server domain that was described in Chapters 2, 12, and 13. First of all, the Process Scheduler domain has no external clients and so has no listener processes. Instead, server processes in the Process Scheduler domain respond to Tuxedo service requests issued by the Process Scheduler server process, PSPRCSRV.

■ **Tip** PeopleSoft does not provide an option in `psadmin` for the Tuxedo `tmadmin` utility. However, you can easily write a simple script to set the necessary environmental variables and then call `tmadmin`:

```
set PS_HOME=D:\ps\hr88
set PS_SERVDIR=%PS_HOME%\appserv\prcs\HR88
set PS_SERVER_CFG=%PS_SERVDIR%\psprcs.cfg
set TUXCONFIG=%PS_SERVDIR%\PSTUXCFG
set TUXDIR=D:\ps\bea\Tuxedo8.1
set PATH=%TUXDIR%/bin;%PATH%
%TUXDIR%\bin\tmadmin
```

In the application server, all the Tuxedo service calls are synchronous. However, this is not always the case with the Process Scheduler domain. When report output and log files have to be transferred to the Report Repository, the Process Scheduler generates a `PostReport` service

call, which is routed by Tuxedo via the DSTQ queue to the Report Distribution Agent, PSDSTSRV. In the Tuxedo trace in Listing 14-1, you can see that the Process Scheduler submits the service asynchronously, with the `tpacall()` function.[1]

Listing 14-1. *Tuxedo trace for the* `PostReport` *service call*

```
231846.GO-FASTER-3!PSPRCSRV.5832.4536.-2: TRACE:at:
   { tpacall("PostReport", 0x1d07250, 41, 0x7)
231846.GO-FASTER-3!PSDSTSRV.4160.5900.0: TRACE:at:
   { tpservice({"PostReport", 0x4, 0x1cc5028, 41, 0, -2147483648, {0, 0, 0}})
231846.GO-FASTER-3!PSPRCSRV.5832.4536.-2: TRACE:at:
   } tpacall = 0 [CLIENTID {0, 0, 0}]
231846.GO-FASTER-3!PSDSTSRV.4160.5900.0: TRACE:at:
   { tpalloc("FML32", "", 4000)
...
231853.GO-FASTER-3!PSDSTSRV.4160.5900.0: TRACE:at:    } tpreturn [long jump]
231853.GO-FASTER-3!PSDSTSRV.4160.5900.0: TRACE:at:  } tpservice
```

Once the `PostReport` message is successfully enqueued, the Process Scheduler does not wait for a reply. The `PostReport` service continues to process, in this case for 7 seconds, after `tpacall()` returned a response.

Up to PeopleTools 8.1, the Application Engine was a stand-alone executable initiated by the Process Scheduler. In PeopleTools 8.4, it has become a Tuxedo server process. The Tuxedo domain should be configured to run as many PSAESRV processes as the Process Scheduler can run concurrent Application Engine programs. Otherwise, you will get the warning message in the Process Scheduler log file shown in Listing 14-2.

Listing 14-2. *Extract from* `SCHDLR_0729.LOG`

```
Validating Application Engine Server Setup....
      Processes with Generic Process Type of Application Engine:
               Optimization Engine      :    2
               Application Engine       :    2
               Total:                        4
      Server Max Concurrent Task Allowed     2
      Application Engine Max Instances:      1
Warning: System has detected the Process Scheduler Server is not set with the
required minimum Application Engine Tuxedo Server needed to run the maximum
concurrency based on Process Scheduler Server Setting. It is recommended to have
PSAESRV instance in the Process Scheduler Config file set to 2. The current max
instance setting for the Application Engine is 1.
```

Nothing untoward will happen in this situation. The Process Scheduler will attempt to run only as many concurrent Application Engine programs as there are PSAESRV processes.

---

1. The `tpcall()` function sends a service request and waits for a reply message, but `tpacall()` does not wait for the reply message. These functions are defined in the BEA documentation "ATMI C Function Reference, Section 3c" (see http://e-docs.bea.com/tuxedo/tux81/rf3c/rf3c.htm).

■ **Note** If you manually shut down one of the PSAESRV processes, the service that it advertises will no longer be available. The Process Scheduler may still attempt to run a program on the deactivated process by requesting that service. The request will fail, and the following error message will be found in the scheduler's error log, `SCHDLR_<mmdd>.log`:

```
PSPRCSRV.1496 [09/24/04 00:52:47 PS@GO-FASTER-3](1) (NET.334): Tuxedo cannot
find the service RunAeAsync2.  Make sure the application server advertising this
service is booted.
PSPRCSRV.1496 [09/24/04 00:52:47 PS@GO-FASTER-3](2)        Process Scheduler
encountered an error sending the request to the AE Tuxedo Server
```

A separate queue is created for each PSAESRV process. Each PSAESRV server dynamically advertises three routines as services on startup, as shown in the Tuxedo trace in Listing 14-3. The server ID is appended to the routine name to form the service name.

Listing 14-3. *Tuxedo trace of the PSAESRV process startup*

```
PSAESRV.4768.4904.0: TRACE:at:  { tpsvrinit(25, "PSAESRV -C dom=PSNT_63788 -g
101 -i 2 -u GO-FASTER-3 -U D:\ps\hr88\appserv\prcs\HR88\LOGS\TUXLOG -m 0 -- -C
psprcs.cfg -CD HR88 -S PSAESRV")
PSAESRV.4768.4904.0: TRACE:at:    { tpadvertise("RunAeAsync2", 0x401080)
PSAESRV.4768.4904.0: TRACE:at:    } tpadvertise = 1 [tperrno TPENOENT]
PSAESRV.4768.4904.0: TRACE:at:    { tpadvertise("ChkAeStatus2", 0x4011b0)
PSAESRV.4768.4904.0: TRACE:at:    } tpadvertise = 1 [tperrno TPENOENT]
PSAESRV.4768.4904.0: TRACE:at:    { tpadvertise("CnclAeProg2", 0x4012e0)
PSAESRV.4768.4904.0: TRACE:at:    } tpadvertise = 1 [tperrno TPENOENT]
```

Therefore, each Application Engine service is advertised on only a single server process. By choosing the service name, the Process Scheduler effectively chooses which Application Engine server process will handle the request. PSPRCSRV submits a `RunAEAsync`*n* service request to a PSAESRV server to initiate an Application Engine program on the server.

The service is submitted synchronously via `tpcall()` so that the Process Scheduler will know when the service has reached a server process, but the service routine completes immediately on the server although the Application Engine program continues to run. From PeopleTools 8.44, the Process Scheduler periodically checks that the Application Engine program is running with the `CheckAEStatus`*n* service. This implies that PSAERV is a threaded process, where one thread runs the `CheckAEStatus`*n* service while another runs the Application Engine program.

The new Optimization Engine has been similarly implemented as a number of Tuxedo servers. Services are dynamically advertised by the servers as they start, so that the Process Scheduler controls exactly where the processes are run.

The configuration process of the Tuxedo domain for the Process Scheduler is very similar to that of the application server described in Chapter 12. While most of the application file names are called `psappsrv.*`, the Process Scheduler files are called `psprcsrv.*`, with the same extension. The exception is that the configuration file is `psprcs.cfg`. PeopleSoft's `ubbgen` utility is used to merge the configuration file and the Tuxedo template to generate a Tuxedo configuration file (`psprcsrv.ubb`).

A master Process Scheduler has been added in PeopleTools 8.4, which balances load between the various Process Schedulers on a system by moving queued requests to different Process Schedulers. This also permits different processes within the same job to run on different Process Schedulers and therefore on different operating systems. For example, you might have a job consisting of a COBOL process followed by a Crystal Reports process. If you only have a COBOL compiler on Unix, the COBOL process must be run on Unix, but the Crystal Reports process can run only on Windows.

Process Monitor

PeopleSoft provides a PIA component called the Process Monitor to enable users to view the Process Scheduler requests, and to retrieve reports and log files via the browser. This page, shown in Figure 14-2, is essentially a view of the table PSPRCSRQST.

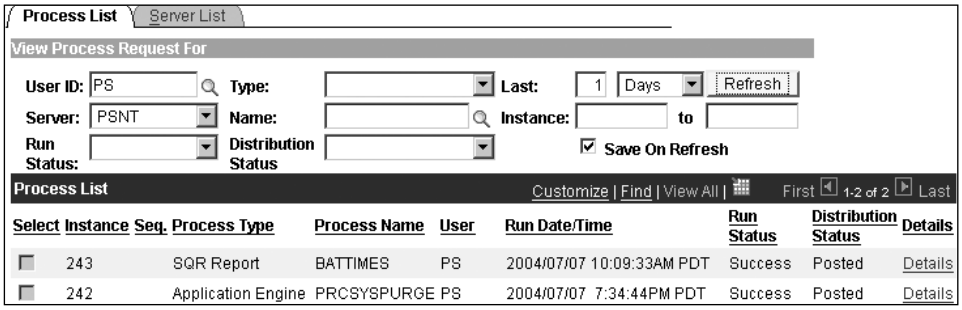


Figure 14-2. *Process Monitor*

DBA Issues

There are several aspects of the Process Scheduler that will be of concern to the DBA. I discuss a number of these issues in the sections that follow.

What Happens When a Process Is Scheduled?

In this section, I will walk through the SQL that is generated when a report process is scheduled via a component in the PIA. This will explain the information the Process Scheduler needs in order to run a batch process. The SQL can be extracted with the PeopleTools trace. The following SQL was generated when I scheduled the SQR report BATTIMES, and it was captured in a PeopleTools trace of the SQL. For clarity, I have replaced the bind variables with the literal values.

The PIA component BATTIMINGS, shown in Figure 14-3, is used to schedule the request.

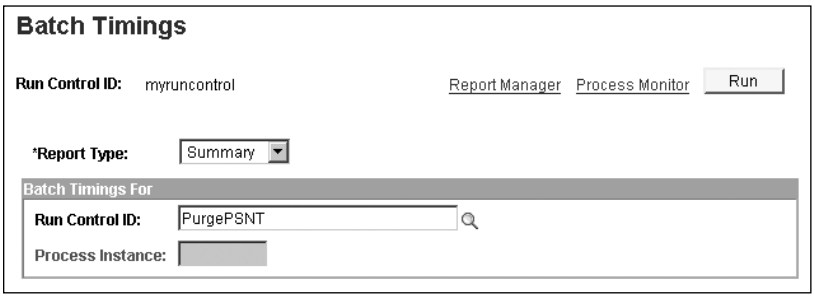


Figure 14-3. *Component to schedule a Batch Timings report*

The component sets up a *run control record* that is used to pass application-specific parameters to the scheduled process. Different processes can either share or have different run control records, depending on what parameters need to be passed to them. In Listing 14-4, the table PS_BATRUNCNTL is specific to the BATTIMES.SQR process and is used to pass parameters specific to that process.

Listing 14-4. *Setting up the process-specific run control*

```
INSERT INTO PS_BATRUNCNTL (OPRID,RUN_CNTL_ID,LANGUAGE_CD,LANGUAGE_OPTION,
BAT_REPORTTYPE,BAT_RUNCNTLID,PRCSINSTANCE)
VALUES('PS', 'myruncontrol', 'ENG', 'O', 'S', 'PurgePSNT', '0');
```

Next, the output destination options for the report are put into the run control detail record, as shown in Listing 14-5.

Listing 14-5. *Destination operations*

```
INSERT INTO PS_PRCSRUNCNTLDTL (OPRID, RUNCNTLID, PRCSTYPE, PRCSNAME, OUTDESTTYPE,
OUTDESTFORMAT, OUTDEST, PSRF_FOLDER_NAME,  PRCSFILENAME, EMAIL_WEB_RPT,
EMAIL_LOG_FLAG)
VALUES ('PS',1,'SQR Report','BATTIMES',1,1,' ',' ',' ',0,0)
;
```

PeopleSoft has separate sequence numbers for scheduled processes and report output. Sequence numbers are incremented and then selected, as shown in Listing 14-6.

■ **Note** PeopleSoft does not use Oracle sequences.

Listing 14-6. *Obtaining the next process request and content sequences numbers*

```
UPDATE PS_PRCSSEQUENCE
SET SEQUENCENO = SEQUENCENO + 1
WHERE PRCSSEQKEY = 0;
```

```
SELECT SEQUENCENO,MIN_SEQ_NBR,MAX_SEQ_NBR
FROM PS_PRCSSEQUENCE WHERE PRCSSEQKEY = 0;

UPDATE PS_PRCSSEQUENCE
SET SEQUENCENO = SEQUENCENO + 1
WHERE PRCSSEQKEY = 1;

SELECT SEQUENCENO,MIN_SEQ_NBR,MAX_SEQ_NBR
FROM PS_PRCSSEQUENCE WHERE PRCSSEQKEY = 1;
```

Distribution and content request records are inserted, as shown in Listing 14-7. By convention, tables with *PRCS* in their name are related to the process, and tables with *CDM* in their name are related to the distribution of process and report output and log files. PeopleSoft sometimes refers to report output as "content." I belive that the acronym CDM refers to *Content and Distribution Management*.

Listing 14-7. *Creating the distribution request and content records*

```
INSERT INTO  PS_PRCSRQSTDIST( PRCSINSTANCE,DISTID,DISTIDTYPE) VALUES (243, 'PS', 2);
INSERT INTO  PS_CDM_AUTH ( PRCSINSTANCE,CONTENTID,DISTID,DISTIDTYPE) VALUES
(243, 206, 'PS', 2);
INSERT INTO  PS_CDM_LIST ( PRCSINSTANCE, CONTENTID, PRCSNAME, PRCSTYPE,
CONTENT_DESCR, OUTDESTFORMAT, RQSTDTTM, ENDDTTM, EXPIRATION_DATE, DISTSTATUS,
DISTNODENAME, OUTPUTDIR, ADMIN_FILENAME, TRANSFERINSTANCE, PRCSOUTPUTDIR,
GENPRCSTYPE, LOGFILEONLY_FLAG, FILENAME, PSRF_FOLDER_NAME)
VALUES (243, 206, 'BATTIMES', 'SQR Report', 'Stored Batch Timings Report', 2,
SYSDATE, TO_DATE(SUBSTR(' ', 0, 19),'YYYY-MM-DD-HH24.MI.SS'),
TO_DATE(' ','YYYY-MM-DD'), 1, 'go-faster-3', ' ', 'index.html', 0, ' ', 1, 0,
' ', ' ');
```

The PSPRCSPARMS table holds the parameters for the process request. These mostly come from the process type definition. You can see in Listing 14-8 that environmental variables and the access ID and password (explained in Chapter 3) are left as variables, which are substituted by the Process Scheduler at run time.

Listing 14-8. *Inserting a run-time parameter record*

```
INSERT INTO PSPRCSPARMS ( PRCSINSTANCE,PARMLIST,WORKINGDIR,CMDLINE,OUTDEST,
ORIGPARMLIST,ORIGOUTDEST,PRCSOUTPUTDIR,PRCSPARMEXTFLAG,PRCSFILENAME ) VALUES
('243', '-CT ORACLE -CS %SERVER% -CD HR88 -CA %ACCESSID% -CAP %ACCESSPSWD% -RP
BATTIMES -I 243 -R 1 -CO PS -OT 6 -OP %LOG/OUTPUT DIRECTORY% -OF 2 -LG ENG',
'%DBBIN%', '%TOOLBINSRV%\PSSQR.EXE', '%%Log/Output Directory%%', '-CT ORACLE –CS
%SERVER% -CD HR88 -CA %ACCESSID% -CAP %ACCESSPSWD% -RP BATTIMES -I 243 -R 1 –CO
PS -OT 6 -OP %LOG/OUTPUT DIRECTORY% -OF 2 -LG ENG', '%%Log/Output Directory%%',
' ', 0, ' ');
```

The table PSPRCSQUE was added in PeopleTools 8.1 (see Listing 14-9). It holds information needed to schedule the request and to check that it is still running. When the process is initiated, the operating system process ID of the client process is stored in the column SESSIONIDNUM on this table.[2] This information is needed should the Process Scheduler need to kill the process.

Listing 14-9. *Creating a new process request record*

```
INSERT INTO PSPRCSQUE ( PRCSINSTANCE, JOBINSTANCE, PRCSJOBSEQ, PRCSJOBNAME,
PRCSTYPE, PRCSNAME, RUNLOCATION, OPSYS,SERVERNAMERQST, SERVERNAMERUN, RUNDTTM,
RECURDTTM, RECURNAME, OPRID, PRCSPRTY, SESSIONIDNUM, RUNSTATUS, RQSTDTTM,
LASTUPDDTTM, RUNCNTLID, PRCSRTNCD, CONTINUEJOB, USERNOTIFIED, INITIATEDNEXT,
OUTDESTTYPE, OUTDESTFORMAT, ORIGPRCSINSTANCE, GENPRCSTYPE, RESTARTENABLED,
TIMEZONE, MAINJOBNAME, MAINJOBSEQ, MAINJOBINSTANCE, PRCSITEMLEVEL,
EMAIL_WEB_RPT, EMAIL_LOG_FLAG, PSRF_FOLDER_NAME, SERVERASSIGN, SCHEDULENAME,
PRCSWINPOP, MCFREN_URL_ID, RECURORIGPRCSINST, RETRYCOUNT, P_PRCSINSTANCE,
PRCSCATEGORY, PRCSCURREXPIREDTTM, DISTSTATUS)
VALUES (243, 0, 0, ' ', 'SQRReport', 'BATTIMES', '2', 2, 'PSNT', ' ',
TO_DATE(SUBSTR('2004-07-07-10.09.33.000000', 0, 19),
'YYYY-MM-DD-HH24.MI.SS'),NULL,' ', 'PS',5, 0, 5, SYSDATE,  SYSDATE, 1, 0, 0, 0,
0, 6, 2, 243, 1, 0, ' ', ' ', 0, 0, 0, 0, 0, ' ', 'PSNT', ' ',0, ' ',
243,0,0,'Default',NULL,1);
```

Before PSPRCSQUE was added, all the information needed to schedule the process was held on PSPRCSRQST (see Listing 14-10). So for historical reasons, there is some duplication of information between these two tables.

Only PSPRCSRQST holds the start and end times for each process, which can be viewed via the Process Monitor component. Therefore, this table is of interest when trying to quantify batch performance.

Listing 14-10. *Creating a process request record*

```
INSERT INTO PSPRCSRQST ( PRCSINSTANCE, JOBINSTANCE, PRCSJOBSEQ, PRCSJOBNAME,
PRCSTYPE, PRCSNAME, RUNLOCATION, OPSYS, DBNAME, DBTYPE, SERVERNAMERQST,
SERVERNAMERUN, RUNDTTM, RECURNAME, OPRID, PRCSVERSION, RUNSTATUS, RQSTDTTM,
LASTUPDDTTM, BEGINDTTM, ENDDTTM, RUNCNTLID, PRCSRTNCD, CONTINUEJOB,
USERNOTIFIED, INITIATEDNEXT, OUTDESTTYPE, OUTDESTFORMAT, ORIGPRCSINSTANCE,
GENPRCSTYPE, RESTARTENABLED, TIMEZONE, MAINJOBNAME, MAINJOBSEQ, MAINJOBINSTANCE,
PRCSITEMLEVEL, PSRF_FOLDER_NAME, SCHEDULENAME, RETRYCOUNT, P_PRCSINSTANCE,
RECURORIGPRCSINST, DISTSTATUS, PRCSCATEGORY, PRCSCURREXPIREDTTM)
VALUES (243, 0, 0, '', 'SQR Report, 'BATTIMES', '2', 2, 'HR88', 2, 'PSNT', ' ',
TO_DATE(SUBSTR('2004-07-07-10.09.33.000000', 0, 19),'YYYY-MM-DD-HH24.MI.SS'),
' ','PS', 0, 5, SYSDATE, SYSDATE, NULL, NULL, 1, 0, 0, 0, 0, 6, 2, 243, 1, 0,
' ', ' ', 0, 0, 0,' ',' ', 0, 0, 243, 1, 'Default',NULL);
```

2. When an SQR is run, PSPRCSQUE.SESSIONIDNUM contains the process ID of the PSSQR wrapper program, which calls the SQR program, and not the SQR program itself. From PeopleTools 8.4, the server ID of the Application Engine server process is stored in PSPRCSQUE.SESSIONIDNUM, and not the PID of the server process.

The output destination parameters are updated, as shown in Listing 14-11. I have no idea why these values were not set when the record was inserted by the SQL statement shown in Listing 14-5.

Listing 14-11. *Updating output destination parameters*

```
UPDATE PS_PRCSRUNCNTLDTL
SET OUTDESTTYPE=6,
    OUTDESTFORMAT=2,
    OUTDEST=' ',
    PSRF_FOLDER_NAME=' ',
    PRCSFILENAME=' ',
    EMAIL_WEB_RPT=0,
    EMAIL_LOG_FLAG= 0
WHERE OPRID='PS' AND RUNCNTLID='myruncontrol'
AND PRCSTYPE = 'SQR Report' AND PRCSNAME = 'BATTIMES'
```

The run control distribution record is replaced, as shown in Listing 14-12.

Listing 14-12. *Deleting and reinserting the run control distribution record*

```
DELETE FROM PS_PRCSRUNCNTLDIST
WHERE OPRID='PS' AND RUNCNTLID='myruncontrol'
AND PRCSTYPE = 'SQR Report' AND PRCSNAME = 'BATTIMES';
INSERT INTO  PS_PRCSRUNCNTLDIST( OPRID,RUNCNTLID,PRCSTYPE,PRCSNAME,DISTID,DISTIDTYPE)
VALUES ('PS', 'myruncontrol', 'SQR Report', 'BATTIMES', 'PS', 2);
```

Process Scheduler Activity

The Process Scheduler periodically queries the request tables looking for work. The frequency of these queries is determined by the Sleep and Heartbeat times that are specified in the server definition. The PeopleSoft delivered settings are shown in Figure 14-4 (select Process Scheduler ➤ Servers ➤ Server Definition).

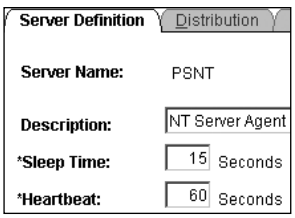


Figure 14-4. *Server Definition page*

Processing Between Sleeps

After each "sleep," the Process Scheduler performs a number of checks. For example, Listing 14-13 shows the query to determine which process requests (that are not PSJobs) are due to be scheduled, and which can be scheduled given the load characteristics of the server, in descending order of priority.

Listing 14-13. *Checking for processes, but not PSJobs, that are due to be scheduled*

```
SELECT R.PRCSINSTANCE ,R.ORIGPRCSINSTANCE ,R.JOBINSTANCE ,R.MAINJOBINSTANCE
,R.MAINJOBNAME ,R.PRCSITEMLEVEL ,R.PRCSJOBSEQ ,R.PRCSJOBNAME ,R.PRCSTYPE
,R.PRCSNAME ,R.PRCSPRTY ,TO_CHAR(R.RUNDTTM,'YYYY-MM-DD-HH24.MI.SS."000000"')
,R.GENPRCSTYPE ,R.OUTDESTTYPE ,R.RETRYCOUNT ,R.RESTARTENABLED ,R.SERVERNAMERQST
,R.OPSYS ,R.SCHEDULENAME ,R.PRCSCATEGORY ,R.P_PRCSINSTANCE ,C.PRCSPRIORITY
,S.PRCSPRIORITY ,R.PRCSWINPOP ,R.MCFREN_URL_ID
FROM   PSPRCSQUE R, PS_SERVERCLASS S, PS_SERVERCATEGORY C
WHERE  R.RUNDTTM <= SYSDATE
AND    R.SERVERASSIGN = :1
AND    R.RUNSTATUS = :2
AND    S.SERVERNAME = :3
AND    R.PRCSTYPE = S.PRCSTYPE
AND    R.PRCSCATEGORY = C.PRCSCATEGORY
AND    S.SERVERNAME = C.SERVERNAME
AND    C.MAXCONCURRENT > 0
AND    R.PRCSTYPE NOT IN ('PSJob')
ORDER BY C.PRCSPRIORITY DESC, R.PRCSPRTY  DESC,
         S.PRCSPRIORITY DESC, R.RUNDTTM ASC
```

Processing on Heartbeat

On each "heartbeat," the Process Scheduler server performs other tests. In particular, it checks its status record to determine whether it has been instructed to suspend or shut down.

Listing 14-14. *Checking the Process Scheduler status record*

```
SELECT SERVERSTATUS,SERVERACTION
FROM   PSSERVERSTAT
WHERE SERVERNAME = :1
ORDER BY 1
```

Process Scheduler Overhead

The Process Scheduler tables can become quite large, even if they are regularly purged. It is not uncommon to see in excess of 1,000 requests per day on a busy Financials system. The overhead of the queries submitted by the Process Scheduler can become quite significant.

It is not uncommon for several schedulers to run on several different servers on a single system. Each scheduler will submit these queries according to its own sleep and heartbeat times. There are a number of options that can be considered:

- **Purging the Process Scheduler request tables**: This is discussed in the next section.
- **Increasing the heartbeat, and particularly the sleep times to reduce the frequency of these queries**: Be aware that this might result in users waiting for longer before their report is scheduled.
- **Tuning SQL queries, but without changing the SQL**: Additional indexes can be helpful. Partitioning can be used in extreme cases.

Purging the Process Scheduler Tables

Unless the Process Scheduler request tables are regularly purged, they will grow, as processes are either scheduled by users or recur automatically. The Process Scheduler scans the request tables looking for work. Listing 14-15 shows one such query that has to scan the PSPRCSQUE table looking for recurring processes that have completed and should be rescheduled.

Listing 14-15. *A Process Scheduler query*

```
SELECT R.PRCSINSTANCE ,R.ORIGPRCSINSTANCE ,R.JOBINSTANCE ,R.PRCSJOBSEQ
,R.PRCSJOBNAME ,R.PRCSNAME ,R.RECURNAME
FROM  PSPRCSQUE R ,PS_PRCSRECUR S
WHERE R.RUNSTATUS IN ('9', '14')   -- requests are success (9) or posting (14)
AND   R.INITIATEDNEXT = 0          -- next occurrence not initiated
AND   R.OPSYS = :1                 -- operating system for request
AND   R.RUNLOCATION = '2'          -- run on server(2)
AND   R.RECURNAME <> ' '           -- a recurring process
AND   R.PRCSJOBSEQ = 0             -- not part of a job
AND   R.SERVERNAMERUN = :2         -- name of server where run
AND   R.RECURNAME = S.RECURNAME    -- join to recurrence definition record
AND   S.INITIATEWHEN = 1           -- schedule next when current initiated(1)
```

It is a feature of human nature that when users schedule a report, they will often follow the link from the component that scheduled the process to the Process Monitor, and sit there repeatedly clicking the Refresh button (see Figure 14-2) to see whether it has completed. However, every click will generate and run a query, like the one in Listing 14-16, on PS_PMN_PRCSLIST, which is a view on PSPRCSRQST.

Listing 14-16. *Query generated by the Process Monitor*

```
SELECT ...
FROM   PS_PMN_PRCSLIST
WHERE  PRCSJOBSEQ = 0
AND    OPRID = 'PS'
AND    (  SERVERNAMERUN  = 'PSNT'
       OR SERVERNAMERQST = 'PSNT')
AND    RQSTDTTM >=
            TO_DATE(SUBSTR('2004-07-06-23.43.47.000000', 0, 19),'YYYY-MM-DD-HH24.MI.SS')
ORDER BY SEQUENCENO DESC, PRCSJOBSEQ
```

The query is generated dynamically by the component using the parameters in the "View Process Request For" part of the component. So every query will be slightly different, and the database will have to parse every one each time, unless CURSOR_SHARING is set to FORCED (or SIMILAR from Oracle9*i*).

As in the previous example, this query is likely to use a full scan of the request table. You are unlikely to find any of these individual statements in an Oracle Statspack or `tkprof` report, because each will have a relatively small overhead. The problem is the cumulative effect of many similar queries. Some PeopleSoft customers have made the decision to customize the Process Monitor and disable the Refresh button, and they have used the JavaScript timer function to refresh the page at a fixed rate.[3]

In systems where the Process Scheduler tables are not regularly purged, these queries can become some of the heaviest statements in the system, and I have sometimes seen them impact the performance of the whole database, not just the Process Scheduler and Process Monitor. In the case of the Process Monitor, the issue is not just that the component performs poorly, but also that the application server processes are blocked by long-running queries that are generating physical I/O to the data files.

The Process Scheduler can be configured to purge the requests after a number of days. I would suggest setting this option to as small a number as the business requirements will permit to keep the Process Scheduler request tables as small as possible. When purging is enabled, the first purge may be exceptionally large, in which case it is advisable to rebuild the Process Scheduler tables and indexes in order to repack the data and reset the high-water marks.

The Process Scheduler requests on PSPRCSRQST are a valuable source of batch process performance metrics and should be retained in an archive table. Chapter 9 illustrates how to copy purged data into an archive table with a trigger. This should not be confused with the new functionality, introduced in PeopleTools 8.19, to control the purging of physical files from the Report Repository, which uses the table PSPRCSRQSTARCH.

Lowering Operating System Priority of Batch Processes

Some of the PeopleSoft batch processes can consume large amounts of CPU. On Unix systems, it is possible to lower the operating system priority of the Process Scheduler, which will then be inherited by the processes that it initiates. There are some scenarios in which this may be desirable in order to prevent batch processing contending with other processes for CPU.

- PeopleSoft is delivering less processing in COBOL and SQR, and is moving toward increased use of the Application Engine. From PeopleTools 8, the Application Engine is also capable of executing PeopleCode. PeopleSoft applications make widespread use of this facility to perform sequential processing in Application Engine programs. PeopleCode is also executed by the component interface, which loads data from a flat file into the system via a component, as if it had been typed into a component in the PIA, executing all validation PeopleCode. As illustrated by the discussion of PeopleSoft Ping in Chapter 10, PeopleCode execution is CPU intensive.

3. See the article "Using JavaScript with PeopleTools," by Paul Cowley of BDO Stoy Hayward, presented at the PeopleSoft UK Technology Product User Group, February 2003 (available from the PeopleSoft Customer Connection website).

- Many PeopleSoft-delivered SQR programs dynamically generate SQL statements in string variables. The CPU overhead of variable handling in SQR also appears to have increased in the versions shipped with PeopleTools 8.*x*.

The principle of lowering the operating system priority of batch processes is the same as lowering the priority of an application server process, as discussed in the last chapter. There are two architectural configurations where you are likely to want to consider this:

- The Process Scheduler is on the same server as the application server, and you do not want batch processing to contend for CPU with the application server because online performance would be degraded.
- The Process Scheduler is on the same server as the database, and you do not want to starve the database of CPU.

■ **Caution** The `UseLocalOracleDB` feature should not be used in conjunction with lowering the operating system priority of the client process. Otherwise, the Oracle shadow process will inherit the priority of the client, and it would be more likely to be taken off the processor run queue while holding a database latch. This could significantly increase contention. If batch processes are run on the same server as the database, then configure an IPC SQL*Net connection instead. The client processes still make an IPC connection, but now the priority of the Oracle shadow process is inherited from the Oracle Listener.

Process Scheduler Not Managed by Tuxedo

In PeopleTools 8.1, the Process Scheduler (PSPRCSRV) is started directly by the `psadmin` utility according to the instructions in the Process Scheduler configuration file, `psprcs.cfg`.

On a Unix system, the `nice` command can be used to alter the priority of a process. I have created a script called `nicePSPRCSRV`, shown in Listing 14-17, that simply executes the scheduler executable, passing through any command-line parameters and using the `nice` command to lower the process priority by 5.

Listing 14-17. `nicePSPRCSRV`*: Script to run the 8.1 Process Scheduler at a lower priority*

```
#nicePSPRCSRV
nice -5 PSPRCSRV $*
```

The name of the Process Scheduler executable is defined in the configuration file by the variable `ProgramName`; it would normally be PSPRCSRV. By changing the value of this variable to be the name of the script, as shown in Listing 14-18, I can instruct `psadmin` to execute the script, and the script will then execute the Process Scheduler via the `nice` command.

Listing 14-18. *Extract from the Process Scheduler configuration file,* `psprcs.cfg`

```
[Process Scheduler]
;=========================================================================
; General settings for the Process Scheduler
;=========================================================================
ProgramName=nicePSPRCSRV
PrcsServerName=PSUNX
```

Process Scheduler Managed by Tuxedo

If the Process Scheduler is managed in Tuxedo, in either release of PeopleTools 8.*x*, you can use Tuxedo to lower the priority of the server processes using the `-n` server option (described in Chapter 13).

Rather than put the setting directly into the Tuxedo template file (`psprcsrv.ubx`), I prefer to create additional variables in the configuration file (`psprcs.cfg`), as shown in Listing 14-19, and then reference them in the template.

Listing 14-19. *New parameter in the Process Scheduler configuration file,* `psprcs.cfg`

```
 [Process Scheduler]
;=====================================================================
; General settings for the Process Scheduler
;=====================================================================
PrcsServerName=PSUNX
;---------------------------------------------------------------------
;Reduce priority of Process Scheduler server process, set to 0 if not needed
Niceness=4
```

From PeopleTools 8.4, the Application Engine and the new Optimization Engine are Tuxedo server processes in the Process Scheduler domain. The priority of these servers can be controlled independently by creating additional variables in the appropriate section of the configuration file, although I have shown only the Application Engine in Listing 14-20.

Listing 14-20. *New parameter in the PSAESRV section of* `psprcs.cfg`

```
 [PSAESRV]
;=====================================================================
; Settings for Application Engine Tuxedo Server
;=====================================================================
;---------------------------------------------------------------------
;Reduce priority of application engine server process, set to 0 if not needed
Niceness=5
...
```

The new variables can then be added to the Tuxedo template (`psprcsrv.ubx`), as shown in Listing 14-21.

Listing 14-21. *Extract from* `psprcsrv.ubx`

```
{APPENG}
#
# PeopleSoft Application Engine Server
#
PSAESRV         SRVGRP=AESRV
                SRVID=1
                MIN={$PSAESRV\Max Instances}
                MAX={$PSAESRV\Max Instances}
                REPLYQ=Y
                CLOPT="-n {$PSAESRV\Niceness} -- -C {CFGFILE} -CD {$Startup\DBName}
```

```
-S PSAESRV"
{APPENG}
...
PSPRCSRV        SRVGRP=BASE
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="SCHEDQ"
                REPLYQ=Y
                CLOPT="-n {$Process Scheduler\Niceness} {$Trace\TuxedoServiceTrace}
-sInitiateRequest -- -C {CFGFILE} -CD {$Startup\DBName}
-PS {$Process Scheduler\PrcsServerName} -A start -S PSPRCSRV"
```

When the domain is configured, the variables are resolved in the Tuxedo configuration file (`psprcsrv.ubb`), and the `nice` command can be seen in the server command-line options (`CLOPT`) in Listing 14-22.

Listing 14-22. *Extract from* `psprcsrv.ubb`

```
#
# PeopleSoft Application Engine Server
#
PSAESRV         SRVGRP=AESRV
                SRVID=1
                MIN=1
                MAX=1
                REPLYQ=Y
                CLOPT="-n 5 -- -C psprcs.cfg -CD HR88 -S PSAESRV"
...
PSPRCSRV        SRVGRP=BASE
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="SCHEDQ"
                REPLYQ=Y
                CLOPT="-n 4 -sInitiateRequest -- -C psprcs.cfg -CD HR88 -PS PSUNX
-A start -S PSPRCSRV"
```

Mutually Exclusive Processing

It may be desirable to prevent certain processes from being scheduled when certain other processes are already running. This may simply be because one process changes data that the other relies upon. Sometimes batch processes can also fail when Oracle detects a deadlock.

In PeopleTools 8.1 and earlier, this effect could be achieved only by putting the mutually exclusive processes that were of the same process type into a new process class with an occurrence level of 1. Thus, only one process in that class could run at a time.

Figure 14-5 shows how the Process Scheduler server should be configured. A new process type of "Mutual Exclusive AE" has been defined, and the Process Scheduler is configured to

run on one such process concurrently. This process class must be available on only a single scheduler; otherwise, one process could run on one scheduler while the other ran on another scheduler.

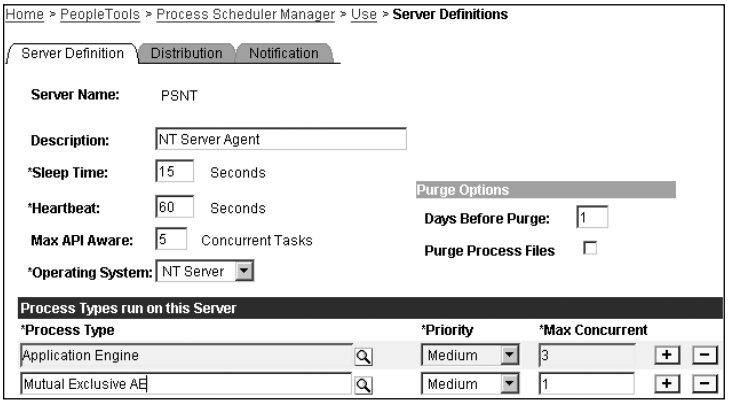


Figure 14-5. *Process Scheduler configuration in PeopleTools 8.1*

The preceding technique still works in PeopleTools 8.4, but it has been superceded. The process definition has been enhanced to specify the following:

- The number of instances of a particular process that can run concurrently across all schedulers
- Other processes that cannot already be running when this process is initiated

Figure 14-6 shows how the Global Payroll banking process, called GP_PMT_PREP, can be specified in PeopleTools 8.4. I have chosen to permit only a single instance of this process to run at the same time; otherwise, one process could corrupt or lock data required by another. Thus the banking process cannot start while the Global Payroll calculation, GPPDPRUN, is running. This is sensible because the banking process needs the result generated by the payroll calculation.

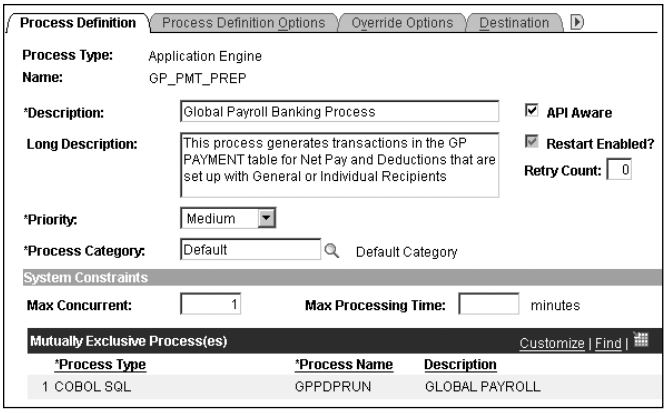


Figure 14-6. *Process definition in PeopleTools 8.4*

Application Engine Server Considerations

In PeopleTools 8.1, the Application Engine was initiated in the same way as any other batch process. It connected to the database, did its work, and then disconnected and terminated. From 8.4, like most other PeopleSoft Tuxedo server processes, PSAESRV maintains a single persistent connection to the database, so the same database session may be used for several Application Engine programs. This has various implications.

If you want to perform a SQL trace an Application Engine program, Chapter 11 describes a trigger that will enable trace. The use of TRACEFILE_IDENTIFIER to alter the name of the trace file for each Process Scheduler request is now essential because the same server, and hence the same session, could run different Application Engine programs, and so trace to the same file. When a new TRACEFILE_IDENTIFIER is specified, the shadow process writes trace information to a different file.

■ **Note** The Oracle MAX_DUMP_FILE_SIZE parameter is applied individually to each trace file that is generated. The trace file ceases to be written when MAX_DUMP_FILE_SIZE has been reached.

On Windows, when a new TRACEFILE_IDENTIFIER is specified, Oracle will start to write again to a new trace file, although there will be a gap in the trace information.

On Unix/Linux platforms, tracing cannot ever be resumed in a session after the maximum size has once been exceeded, even if MAX_DUMP_FILE_SIZE is subsequently reset. Thus if a SQL trace of an Application Engine program exceeds the maximum size, it will not be possible to trace any other Application Engine program that runs on that PSAESRV server process. If you set MAX_DUMP_FILE_SIZE=UNLIMITED, the trace will not be disabled, but there is an increased risk that the file system could fill up. Furthermore, if MAX_DUMP_FILE_SIZE is set back to a size less than the current trace file size, even after tracing has been stopped, Oracle will detect the condition and further tracing will be prevented.

A second trigger is now necessary to disable trace when the process terminates (see Listing 14-23) because the session does not terminate.

Listing 14-23. *Trigger to disable trace when an Application Engine program ends*

```
CREATE OR REPLACE TRIGGER sysadm.unset_trace
BEFORE UPDATE OF runstatus ON sysadm.psprcsrqst
FOR EACH ROW
WHEN (new.runstatus != 7 AND old.runstatus = 7
AND   new.prcstype IN ('Application Engine'))
BEGIN
   sys.dbms_session.set_sql_trace(FALSE);
   EXECUTE IMMEDIATE 'ALTER SESSION SET TRACEFILE_IDENTIFIER = ''''';
-- Only reduce MAX_DUMP_FILE_SIZE after changing the TRACEFILE IDENTIFIER again
-- EXECUTE IMMEDIATE 'ALTER SESSION SET MAX_DUMP_FILE_SIZE = 10000';
EXCEPTION WHEN OTHERS THEN NULL;
END;
```

If you used the trigger to set Oracle parameters when a particular Application Engine program begins, you may also need to reset them to the original values in the second trigger when it completes.

If you introduce Global Temporary Tables into an Application Engine program, in any version of PeopleTools, they must be created as `ON COMMIT PRESERVE` because Application Engine programs often commit at intermediate points.

From PeopleTools 8.4, the tables will not be cleared out automatically when the Application Engine program completes because the PSAESRV server process, and therefore the database session, does not terminate. PeopleSoft-delivered Application Engine programs often—but not always—truncate temporary working storage tables. The result will be increased and longer-lasting demands for space in the temporary tablespace. It may be necessary to add additional truncate statements to Application Engine programs.

It is tempting to set the recycle count on the PSAESRV processes to 1, so that they restart after every service request. However, the server is threaded, and the Application Engine program is called asynchronously and continues to execute after the service has completed. While it executes, the Process Scheduler periodically makes other synchronous service calls to the server to check the status of the program. These services will be counted toward the recycle count. Hence if a recycle count is specified, the server might try to recycle while an Application Engine program is executing.[4] Therefore, recycling should be disabled by setting the recycle count to 0.

Summary

As explained in this chapter, the Process Scheduler is of importance to the DBA because it can affect areas of the system for which the DBA is properly responsible. Just as the application server could be used to regulate online activity, so the configuration of the Process Scheduler can be used to regulate the load that batch and report processing can place on the database. The Process Scheduler tables need to be aggressively and regularly purged to prevent the Process Scheduler itself from becoming a source of database performance problems.

The conversion of the Process Scheduler to a Tuxedo domain, and the Application Engine to a server process within that domain, has complicated the way in which some performance-tuning techniques are implemented.

4. In PeopleTools 8.44, I have found that this caused the process to lock up itself, its queue, and the Tuxedo Bulletin Board.

Index

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U