

THE EXPERT'S VOICE® IN ORACLE

OakTable
PRESS

SPECIAL PREVIEW EDITION!
FULL EDITION AVAILABLE NOVEMBER 22, 2004

PeopleSoft for the Oracle DBA

A PeopleSoft Survival Guide for Oracle DBAs



David Kurtz

Apress®

Contents at a Glance

This is a special preview edition of David Kurtz's forthcoming book, *PeopleSoft for the Oracle DBA*. The full version will be published in November 2004 (Apress, 1-59059-422-3). Following is a table of contents for the complete book, with the chapters and sections included in this edition highlighted in italic type.

About the Book

About the Author

About the OakTable Press

<i>CHAPTER 1</i>	<i>An Overview</i>
CHAPTER 2	BEA Tuxedo: Peoplesoft's Application Server Technology
CHAPTER 3	Database Connectivity
CHAPTER 4	PeopleSoft Database Structure: A Tale of Two Data Dictionaries
CHAPTER 5	Keys and Indexing
CHAPTER 6	Peoplesoft DDL
CHAPTER 7	Tablespaces
CHAPTER 8	Locking, Transactions, and Concurrency
<i>CHAPTER 9</i>	<i>Performance Metrics</i>
CHAPTER 10	PeopleTools Performance Utilities
CHAPTER 11	SQL Optimization Techniques in PeopleSoft
CHAPTER 12	Configuring the Application Server
CHAPTER 13	Tuning the Application Server
CHAPTER 14	The Process Scheduler

Please note that the chapters included here are in beta form and are subject to modification and correction before final release.

About the Book

This book is a survival guide for the Oracle DBA who is responsible for the configuration, administration, and performance of a PeopleSoft system. The database is at the core of any such system, but a DBA must also understand how the PeopleSoft application is using the database, and how other parts of the PeopleSoft architecture interact with the database. Therefore, this book

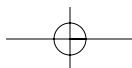
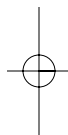
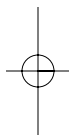
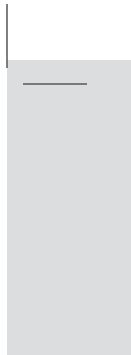
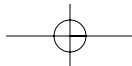
- Demonstrates how to implement common Oracle database administration techniques and strategies (indexing, DDL, managing tablespaces, and so on) specifically for a PeopleSoft system and using the PeopleSoft toolset
- Shows how to analyze application activity and obtain the critical data that will allow you to track down the causes of poor performance
- Provides essential details on the configuration and tuning of the Tuxedo application server and how it works with the database

This book is a reflection of what I've been doing most days for many years: helping customers get sufficient performance out of their PeopleSoft business systems. In this book, I provide answers to the questions that I am still repeatedly asked by DBAs, developers, and end users about what is going on inside these systems, for example:

- How do I measure what part of the application response time is spent in which tier of the system, and thus determine whether the database or another tier is the most significant performance issue?
- Who is running this piece of SQL, where does it come from, and what can I do about it?
- What changes can I safely make to the system, via the PeopleSoft development tools, without creating an administrative nightmare?

Ultimately, I hope this book will demonstrate how an Oracle DBA can work effectively *with* PeopleSoft, instead of continually fighting *against* it.

Yours sincerely,
David Kurtz (david.kurtz@go-faster.co.uk)



About the Author

DAVID KURTZ 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), a company 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.

You can talk to David about the book and find many useful PeopleSoft-related links, articles, and other resources at the book's dedicated website, <http://www.psftdba.com>.

About the OakTable Press

Apress and the OakTable network have joined forces to define a radical new direction in Oracle publishing: the *OakTable Press*. Put simply, the OakTable network is an informal organization consisting of a group of Oracle experts who are dedicated to finding ever better ways of administering and developing Oracle-based systems.

The members of the network have a few things in common. We take a scientific approach to working with the Oracle database. We don't believe anything unless we've seen it thoroughly tested and proved. We enjoy "moving boundaries," innovating, finding new and better ways to do things. We like good whiskey. These, in essence, are the ideals that we want to bring to the OakTable Press series (well, apart from the last one, possibly). Every book in the series will be written by and/or technically reviewed by at least two members of the OakTable network. Every book published by OakTable Press will be imbued by the qualities that we admire: they will be scientific, rigorous, accurate, innovative, and fun to read. Ultimately, we hope that each book is as useful a tool as it can possibly be in helping make your life easier.

Who Are the OakTable Network?

It all started sometime in 1998 when a group of Oracle experts, including Anjo Kolk, Cary Millsap, James Morle, and a few others, started meeting once or twice a year, on various pretexts. Each would bring a bottle of Scotch or Bourbon and in return earn the right to sleep on the floor somewhere in my house.

We spent most of our time sitting around my dining table, with computers, cabling, paper, and other stuff all over the place, discussing Oracle, relaying anecdotes, and experimenting with new and better ways of working with the database. By the spring of 2002, the whole thing had grown. One evening, I realized that I had 16 world-renowned Oracle scientists sitting around my dining table. We were sleeping three or four to a room and even had to borrow the neighbor's shower in the mornings. Anjo Kolk suggested we call ourselves the

vi ■ ABOUT THE AUTHOR

“OakTable network” (after my dining table), and about 2 minutes later, www.OakTable.net was registered.

Today, a total of 42 people have been admitted to the OakTable network, and perhaps half of them work for Oracle Corporation (there’s an up-to-date list on the website). A committee, consisting of James Morle, Cary Millsap, Anjo Kolk, Steve Adams, Jonathan Lewis, and myself, review suggestions for new members.

You can meet us at various conferences and user group events, and discuss technical issues with us or challenge the OakTable network with a technical question. If we can’t answer your question within 24 hours, you get a T-shirt that says, “I challenged the OakTable—and I won,” with the three last words printed in very, very small type! We still meet twice a year in Denmark: in January for the Miracle Master Class (2001: Cary Millsap, 2002: Jonathan Lewis, 2003: Steve Adams, and 2004: Tom Kyte), when one of the members will present for 3 days, and in September/October for the Miracle Database Forum, which is a 3-day conference for database people.

Many projects and ideas have come out of the OakTable network, with some of them resulting in courses (such as the Hotsos Clinic), others resulting in new software products, and one that resulted in the OakTable Press series. We hope you’ll enjoy the OakTable Press books we help produce in the coming years.

Best,
Mogens Nørgaard
Technical director of Miracle A/S
(<http://www.miracleas.dk/>)
and cofounder of the
OakTable network

About the Book

This book is a survival guide for the Oracle DBA who is responsible for the configuration, administration, and performance of a PeopleSoft system. The database is at the core of any such system, but a DBA must also understand how the PeopleSoft application is using the database, and how other parts of the PeopleSoft architecture interact with the database. Therefore, this book

- Demonstrates how to implement common Oracle database administration techniques and strategies (indexing, DDL, managing tablespaces, and so on) specifically for a PeopleSoft system and using the PeopleSoft toolset
- Shows how to analyze application activity and obtain the critical data that will allow you to track down the causes of poor performance
- Provides essential details on the configuration and tuning of the Tuxedo application server and how it works with the database

This book is a reflection of what I've been doing most days for many years: helping customers get sufficient performance out of their PeopleSoft business systems. In this book, I provide answers to the questions that I am still repeatedly asked by DBAs, developers, and end users about what is going on inside these systems, for example:

- How do I measure what part of the application response time is spent in which tier of the system, and thus determine whether the database or another tier is the most significant performance issue?
- Who is running this piece of SQL, where does it come from, and what can I do about it?
- What changes can I safely make to the system, via the PeopleSoft development tools, without creating an administrative nightmare?

Ultimately, I hope this book will demonstrate how an Oracle DBA can work effectively *with* PeopleSoft, instead of continually fighting *against* it.

Yours sincerely,
David Kurtz (david.kurtz@go-faster.co.uk)

About the Author

DAVID KURTZ 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), a company 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.

You can talk to David about the book and find many useful PeopleSoft-related links, articles, and other resources at the book's dedicated website, <http://www.psftdba.com>.

About the OakTable Press

Apress and the OakTable network have joined forces to define a radical new direction in Oracle publishing: the *OakTable Press*. Put simply, the OakTable network is an informal organization consisting of a group of Oracle experts who are dedicated to finding ever better ways of administering and developing Oracle-based systems.

The members of the network have a few things in common. We take a scientific approach to working with the Oracle database. We don't believe anything unless we've seen it thoroughly tested and proved. We enjoy "moving boundaries," innovating, finding new and better ways to do things. We like good whiskey. These, in essence, are the ideals that we want to bring to the OakTable Press series (well, apart from the last one, possibly). Every book in the series will be written by and/or technically reviewed by at least two members of the OakTable network. Every book published by OakTable Press will be imbued by the qualities that we admire: they will be scientific, rigorous, accurate, innovative, and fun to read. Ultimately, we hope that each book is as useful a tool as it can possibly be in helping make your life easier.

Who Are the OakTable Network?

It all started sometime in 1998 when a group of Oracle experts, including Anjo Kolk, Cary Millsap, James Morle, and a few others, started meeting once or twice a year, on various pretexts. Each would bring a bottle of Scotch or Bourbon and in return earn the right to sleep on the floor somewhere in my house.

We spent most of our time sitting around my dining table, with computers, cabling, paper, and other stuff all over the place, discussing Oracle, relaying anecdotes, and experimenting with new and better ways of working with the database. By the spring of 2002, the whole thing had grown. One evening, I realized that I had 16 world-renowned Oracle scientists sitting around my dining table. We were sleeping three or four to a room and even had to borrow the neighbor's shower in the mornings. Anjo Kolk suggested we call ourselves the

vi ■ ABOUT THE AUTHOR

“OakTable network” (after my dining table), and about 2 minutes later, www.OakTable.net was registered.

Today, a total of 42 people have been admitted to the OakTable network, and perhaps half of them work for Oracle Corporation (there’s an up-to-date list on the website). A committee, consisting of James Morle, Cary Millsap, Anjo Kolk, Steve Adams, Jonathan Lewis, and myself, review suggestions for new members.

You can meet us at various conferences and user group events, and discuss technical issues with us or challenge the OakTable network with a technical question. If we can’t answer your question within 24 hours, you get a T-shirt that says, “I challenged the OakTable—and I won,” with the three last words printed in very, very small type! We still meet twice a year in Denmark: in January for the Miracle Master Class (2001: Cary Millsap, 2002: Jonathan Lewis, 2003: Steve Adams, and 2004: Tom Kyte), when one of the members will present for 3 days, and in September/October for the Miracle Database Forum, which is a 3-day conference for database people.

Many projects and ideas have come out of the OakTable network, with some of them resulting in courses (such as the Hotsos Clinic), others resulting in new software products, and one that resulted in the OakTable Press series. We hope you’ll enjoy the OakTable Press books we help produce in the coming years.

Best,
Mogens Nørgaard
Technical director of Miracle A/S
(<http://www.miracleas.dk/>)
and cofounder of the
OakTable network

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 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 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 "PeopleSoft 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.

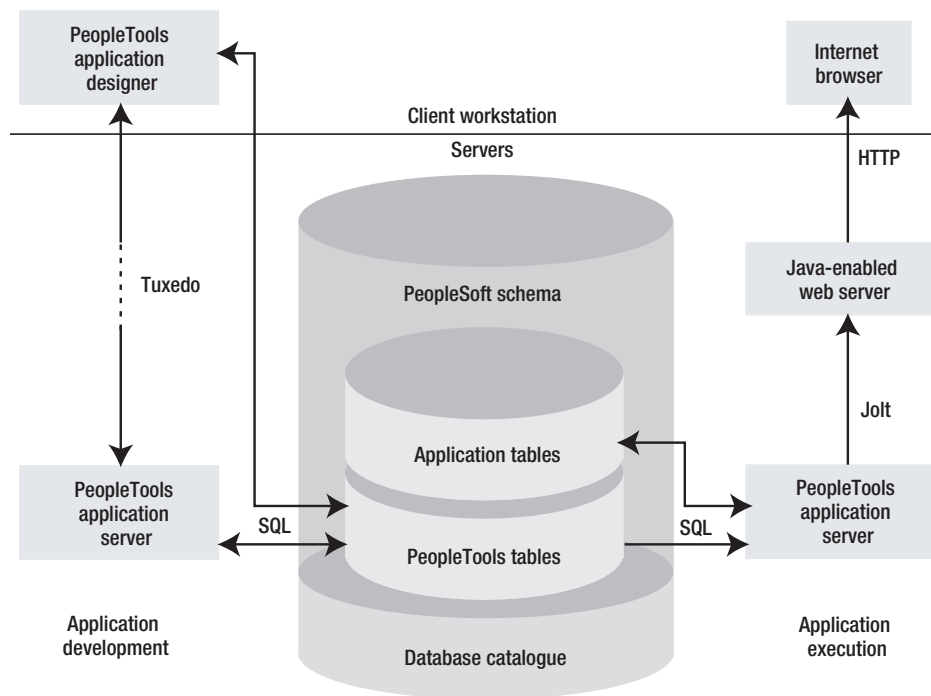


Figure 1-1. A simplified version of the PeopleTools architecture

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 the all 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 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 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** (formerly owned by SQRIBE, but now owned by PeopleSoft) 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 are defined via the PIA and stored in the database, rather than in Excel workbook files. They 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.

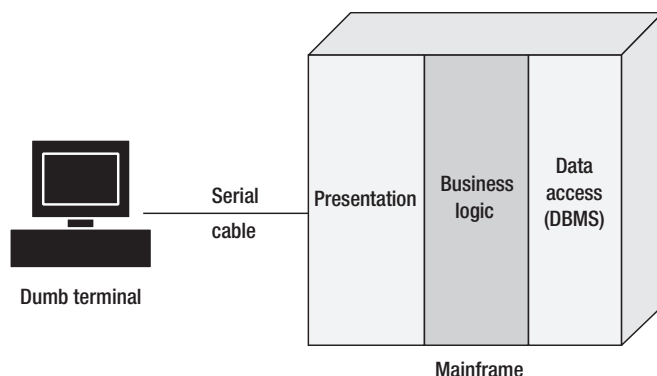


Figure 1-2. *Monolithic mainframe model*

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

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.

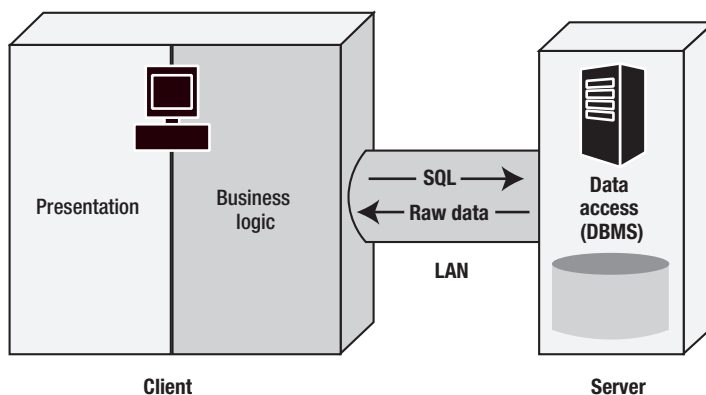


Figure 1-3. *The original client/server model*

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.

Until 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 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 centre, 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.

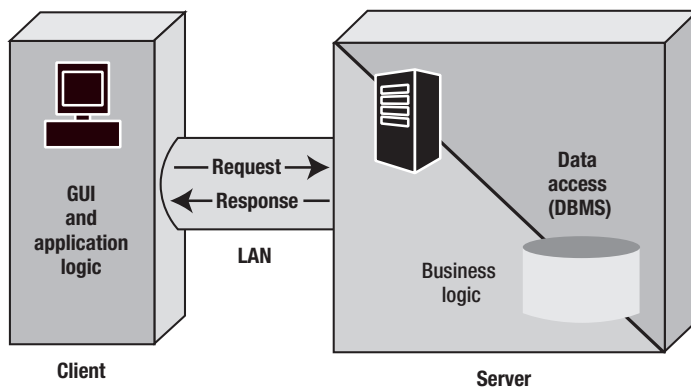


Figure 1-4. *A first-generation three-tier client/server model*

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.

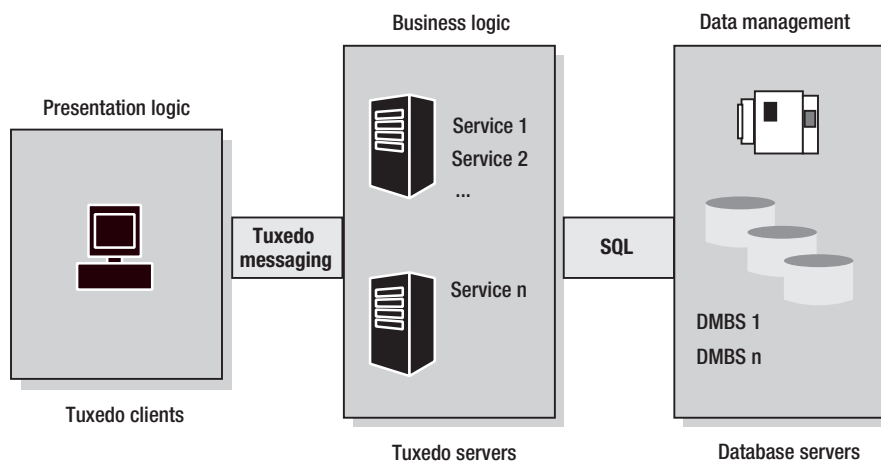


Figure 1-5. *The Tuxedo second-generation three-tier client/server model*

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 Java servlet that runs 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.

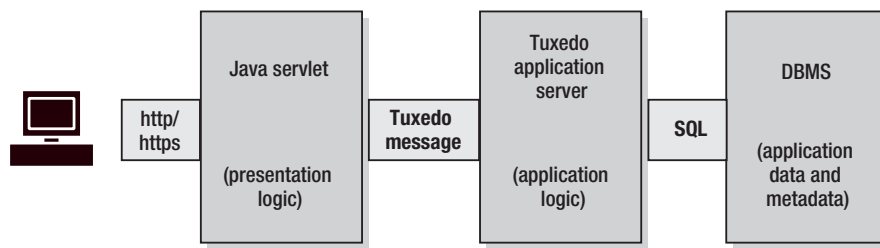


Figure 1-6. *The PeopleSoft Internet Architecture (PIA)*

“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 usually, but not always, the same. 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 3.x and 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 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 purely 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.

7. 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.

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 through 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 those 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 has 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 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 measurement² 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.

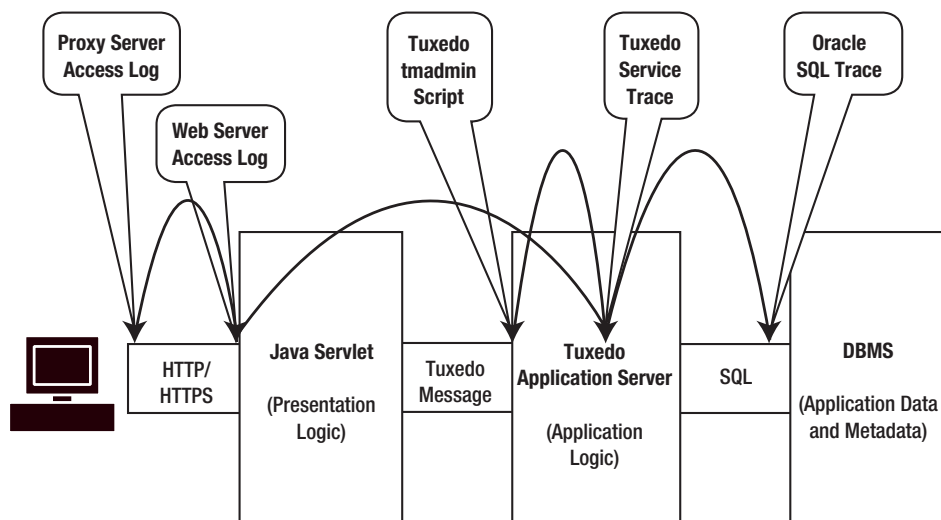


Figure 9-1. PeopleSoft Internet Architecture with sources of metrics

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 `tadmin` 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,PSPRCRV.exe,	PSPRCRV.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 processes⁴ 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`: List clients of the bulletin board.
- `pq` or `printqueue`: List queues within the application server and the number of requests queued.
- `psr` or `printserver`: List 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+

154 CHAPTER 9 ■ PERFORMANCE METRICS

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 Num/Avg	TOTALS 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
,timestamp  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,timestamp,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
,timestamp  "(:timestamp/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 timestamp 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:
```


158 CHAPTER 9 ■ PERFORMANCE METRICS

```

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 JavaMgrGetObject 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
@JavaMgrGetObject 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 to handle the messages took 0.02 seconds.

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 8.1x 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.

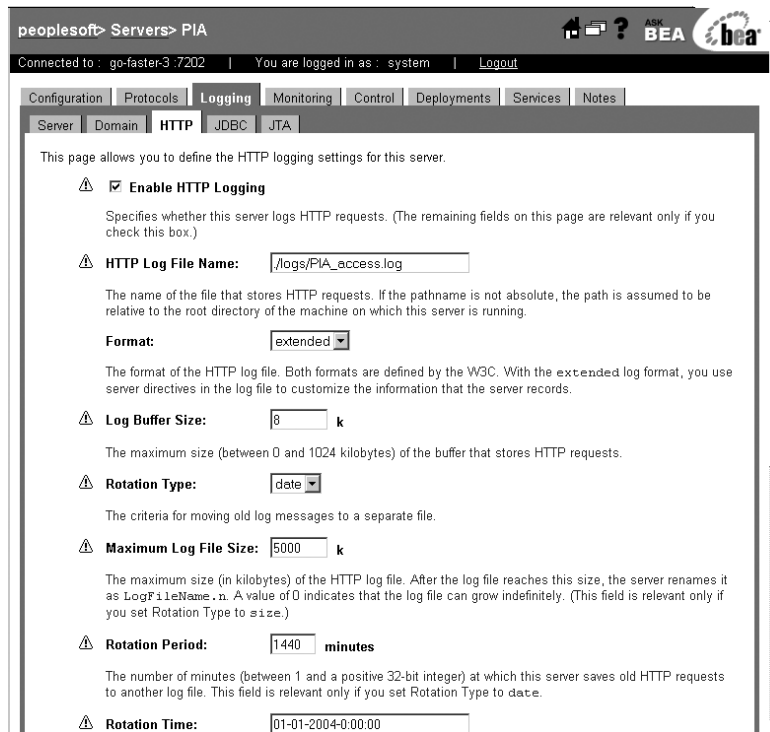


Figure 9-2. Part of the WebLogic 8.1 console

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=PFDKxd0Qe8E/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-
```

162 CHAPTER 9 ■ PERFORMANCE METRICS

```
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%2fpsc%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)
);
```

The WebLogic access log can then be imported into the table with the SQL*Loader control file shown in Listing 9-20.

6. See <http://www.w3.org/TR/WD-logfile.html>.

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.

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
                )
                , 'XXXXXXX'
            )
        )
        ||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
=PfDKxdQe8E/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/

```

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.

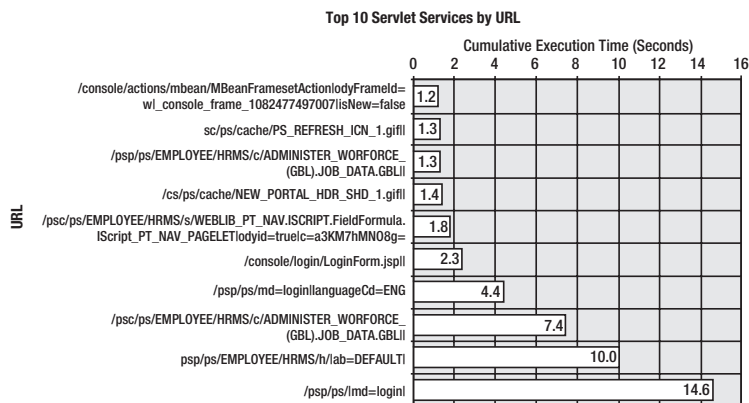


Figure 9-3. Top ten operations by cumulative execution time

Apache Web Server Access Log

PeopleSoft certified Apache 1.3 for PeopleTools 8.1x 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
<code>%t</code> For instance, <code>%{Y.m.%d %H:%M:%S}t</code>	Time, in the form given by the format, that should be in <code>strftime(3)</code> format (potentially localized). <code>%Y</code> = Year as a four-digit number <code>%m</code> = Month of the year as a two-digit number <code>%d</code> = Day of the month as a two-digit number <code>%H</code> = Hour of the day as a two-digit number <code>%M</code> = Minutes as a two-digit number <code>%S</code> = Seconds as a two-digit number
<code>%T</code>	Time taken to serve the request, in seconds.
<code>%B</code>	Bytes sent, excluding HTTP headers.
<code>%u</code> status [%s] is 401).	Remote user (from auth; may be bogus if return
<code>%h</code>	Remote host.
<code>%{User-Agent}I</code>	Web browser identifier string.
<code>%>s</code>	Status. For requests that got internally redirected, this is the status of the <i>original</i> request; <code>%...>s</code> for the last.
<code>%m</code>	Request method.
<code>%U</code>	URL path requested.
<code>%q</code>	Query string (prepended with <code>?</code> 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|
```

168 CHAPTER 9 ■ PERFORMANCE METRICS

```

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/iclientervlet/?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/iclientervlet/?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/iclientervlet/?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/iclientervlet/?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 2-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 3-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.

170 CHAPTER 9 ■ PERFORMANCE METRICS

However, SQL trace will only give you the SQL statement. If you have enabled `EnabledDBMonitoring` 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/iclientervlet/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/iclientervlet/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
/
SPPOOL 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_DESCRIPTIONS	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.

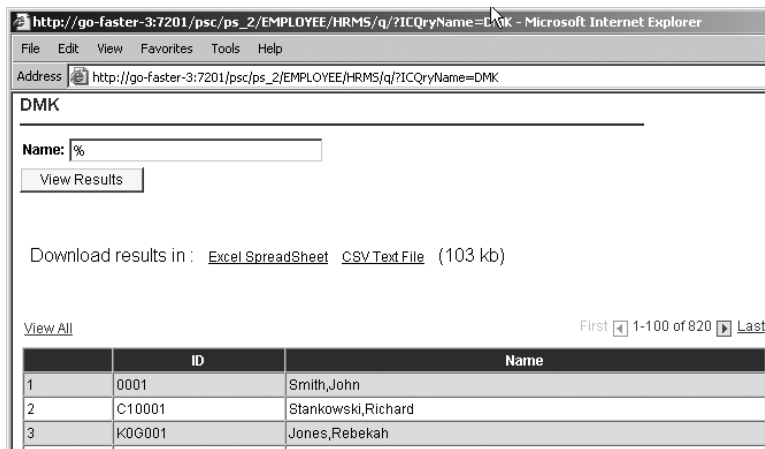


Figure 9-4. Query results

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 PSPRCRQST. 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.

174 CHAPTER 9 ■ PERFORMANCE METRICS

Process List **Server List**

View Process Request For

User ID: PS Type: Last: 1 Days Refresh

Server: Name: Instance: to

Run Status: Distribution Status ☒ Save On Refresh

Process List Customize Find View All First 1 of 7 Last

Select	Instance	Seq.	Process Type	Process Name	User	Run Date/Time	Run Status	Distribution Status	Details
<input type="checkbox"/>	166		SQR Report	SYSAUDIT	PS	26/04/2004 13:41:19 PDT	Success	N/A	Details
<input type="checkbox"/>	165		SQR Report	SWPAUDIT	PS	26/04/2004 13:41:19 PDT	Success	Posted	Details
<input type="checkbox"/>	163		SQR Report	PTSQRTST	PS	26/04/2004 13:41:19 PDT	Success	Posted	Details
<input type="checkbox"/>	162		Application Engine	AEMINTEST	PS	26/04/2004 13:41:19 PDT	Initiated	N/A	Details
<input type="checkbox"/>	158		PSJob	3SQR	PS	26/04/2004 13:41:19 PDT	Processing	N/A	Details
<input type="checkbox"/>	157		SQR Report	DDDAUDIT	PS	26/04/2004 13:40:18 PDT	Success	Posted	Details
<input type="checkbox"/>	156		Application Engine	PRCSYSPURGE	PS	26/04/2004 13:05:58 PDT	Success	Posted	Details

Figure 9-5. Process Monitor summary page

Within the Process Monitor, you can drill into a single process, as shown in Figure 9-6.

Process Detail

Process

Instance: 157 Type: SQR Report

Name: DDDAUDIT Description: Data Designer/Database Audit

Run Status: Success Distribution Status: Posted

Run **Update Process**

Run Control ID: 1

Location: Server

Server: PSNT

Recurrence:

☐ Hold Request

☐ Queue Request

☐ Cancel Request

☐ Delete Request

☐ Restart Request

Date/Time **Actions**

Request Created On: 26/04/2004 13:40:32 PDT [Parameters](#) [Transfer](#)

Run Anytime After: 26/04/2004 13:40:18 PDT [Message Log](#)

Began Process At: 26/04/2004 13:41:00 PDT [Batch Timings](#)

Ended Process At: 26/04/2004 13:43:33 PDT [View Log/Trace](#)

OK Cancel

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.

Process Detail

Process Name: 3SQR

Main Job Instance: 158

Left | Right

- 158 - 3SQR Processing
- 159 - XRFIELDS Queued
- 160 - XRFMENU Pending
- 161 - XRFRCFL Pending

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 on 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 processes 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 prcinstance
,      prcstype
,      prcsname
,      TO_CHAR(TRUNC(SYSDATE)+(enddtm-begindtm),'HH24:MI:SS') exec_time
,      enddtm-begindtm exec_days
,      (enddtm-begindtm)*24 exec_hrs
,      (enddtm-begindtm)*1440 exec_mins
,      (enddtm-begindtm)*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

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(enddtm-begindtm)*86400 dur
,      AVG(enddtm-begindtm)*86400 avg_dur
FROM      psprcsrqst p
WHERE     begindtm IS NOT NULL
AND       enddtm IS NOT NULL /*have completed*/
AND       runstatus IN(9,11,12,13,14,15,16,17) /*successful proc*/
AND       begindtm > 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 the source of batch performance metrics, so it needs to be retained in a different table.

I suggest creating a copy of the PSPCRSRQST record in the Application Designer and building this as an archive table, in this case called PS_PRCRQSTARCH. 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 PSPCRSRQST 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
```

180 **CHAPTER 9 ■ PERFORMANCE METRICS**

```
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.PRCINSTANCE
    ...
    ,: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.psprcsrqt
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 is 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.statpack.snap
    (i_snap_level=>5
    ,i_ucomment=>l_comment
    );
  COMMIT;
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

Note The autonomous transaction is required because `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

Instance	DB Name	Snap Id	Snap Started	Snap 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
/
```

184 CHAPTER 9 ■ PERFORMANCE METRICS

```
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 PSPRCRQST_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 a great deal of 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, PSPRCRQST. 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.
- <prcss 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. cobsq1_ptptedit_2.trc

PeopleSoft Batch Timings Report: cobsq1_ptptedit_2.trc

Encoding Scheme Used: Ansi

PeopleSoft Batch Statistics
(All timings in seconds)

Encoding Scheme Used: Ansi

Statement	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	
	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

188 CHAPTER 9 ■ PERFORMANCE METRICS

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_PRCQSB	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

```


PeopleCode	Call Count	Non-SQL Time	SQL Time	Total Time
------------	---------------	-----------------	-------------	---------------

AE Program: PER099

INIT.Step03	1	0.0	0.0	0.0
		-----	-----	-----
		0.0	0.0	0.0

PEOPLECODE Builtin/Method	E x e c u t e	
	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.

192 CHAPTER 9 ■ PERFORMANCE METRICS

Batch Timings - Summary


Process									
Instance: 181		Type: Application Engine							
Name: PER099		Description: Fill EMPLOYEES Table							
Time (in milliseconds)				Trace Level					
Elapsed: 48400		Application Engine: 1152							
In PeopleCode: 20		SQL & PeopleCode: 128							
In SQL: 27660									
Customize Find View All  First 1-25 of 25 Last									
Program	Detail line identifier	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	RELANG.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
PER099	UPDATE.Step11.S	1	10	1	1101	0	0	0	0

Figure 9-8. Batch timings in Process Monitor

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.

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*

Stmt Rank	Statement ID	Exec Count	AE SQL Time	PC SQL Time	PC Time	% Cum % SQL Time	% Cum % SQL Time	Total Time	Total Time
1	HR_FASTVIEW.SEC_UPD.S tep01.D	39	6.4	0.0	0.0	16.4	16.4	4.8	4.8
2	AE Internal.COMMIT	101	4.8	0.0	0.0	12.3	28.7	3.6	8.5
3	PER099.INSERT.Step01. S	1	3.8	0.0	0.0	9.7	38.5	2.9	11.4
4	PER099.COPY.Step03.S	5	3.3	0.0	0.0	8.6	47.0	2.5	13.9
5	PER099.UPDATE.Step05. S	1	3.1	0.0	0.0	7.9	54.9	2.3	16.2
...									

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.

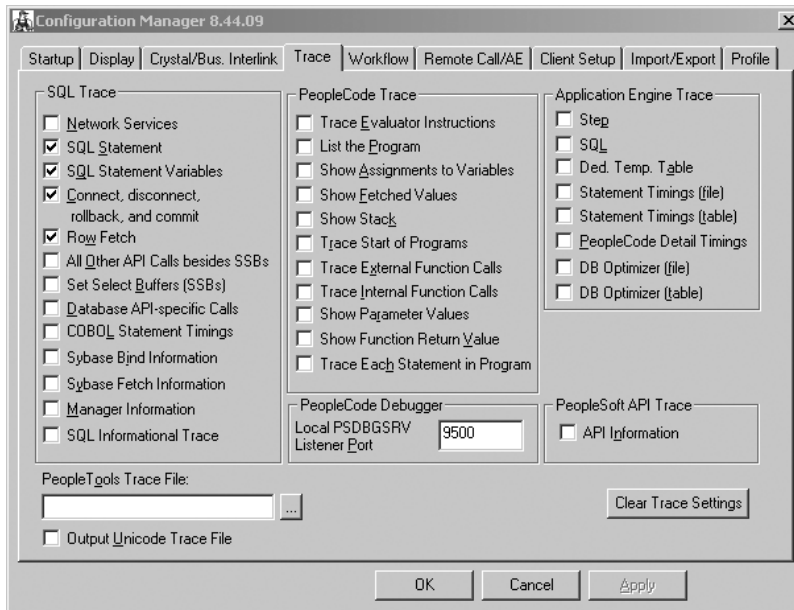


Figure 9-9. Configuration Manager

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.

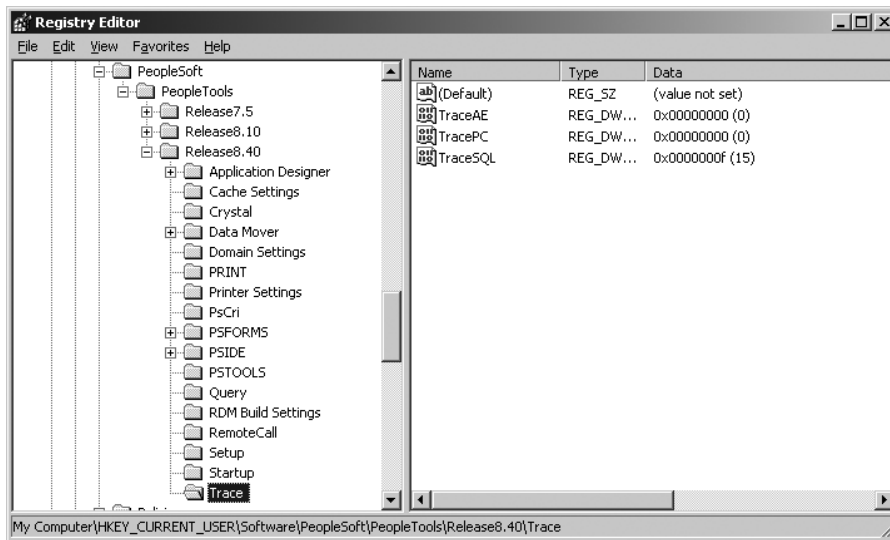


Figure 9-10. PeopleTools trace registry settings

196 CHAPTER 9 ■ PERFORMANCE METRICS

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.

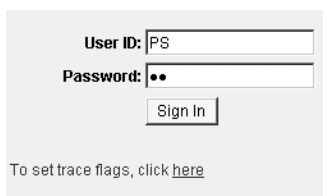
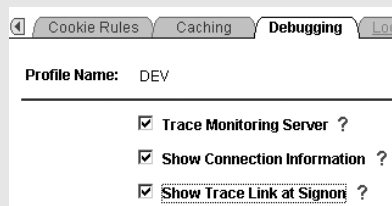


Figure 9-11. PIA login challenge

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.

SQL trace settings:	PeopleCode trace settings:	Component Processor trace settings:
<input checked="" type="checkbox"/> SQL statements	<input type="checkbox"/> Evaluator instructions	<input type="checkbox"/> Page Structures at Init
<input checked="" type="checkbox"/> SQL statement variables	<input type="checkbox"/> List program	<input type="checkbox"/> Component Buffers at Init
<input checked="" type="checkbox"/> SQL connect, disconnect, commit, rollback	<input type="checkbox"/> Variable assignments	<input type="checkbox"/> Component Buffers before/after service
<input checked="" type="checkbox"/> SQL fetch	<input type="checkbox"/> Fetched values	<input type="checkbox"/> Component Buffers after scrollselect
<input type="checkbox"/> All other SQL API calls except SSBs	<input type="checkbox"/> Evaluator stack	<input type="checkbox"/> Component Buffers after modal page
<input type="checkbox"/> Set select buffer calls (SSBs)	<input type="checkbox"/> Program starts	<input type="checkbox"/> Component Buffers before Save
<input type="checkbox"/> Database-specific API calls	<input type="checkbox"/> External function calls	<input type="checkbox"/> Component Buffers after row insert
	<input type="checkbox"/> Internal function calls	<input type="checkbox"/> Default Processing
	<input type="checkbox"/> Function parameter values	<input type="checkbox"/> PRM Contents
	<input type="checkbox"/> Function return values	<input type="checkbox"/> HTML errors
	<input type="checkbox"/> Each statement	<input type="checkbox"/> Memory stats at Init
		<input type="checkbox"/> Keylist Generation
		<input type="checkbox"/> Work Record Flagging
		<input type="checkbox"/> Related Displays

User ID:

Password:

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.

Trace SQL

Select Trace options below, then select Save.

Options	
<input checked="" type="checkbox"/> Trace SQL Statement (1)	<input checked="" type="checkbox"/> Trace SQL -- Database Level (64)
<input checked="" type="checkbox"/> Trace SQL Bind (2)	<input checked="" type="checkbox"/> Trace MGR -- Manager Level (4096)
<input checked="" type="checkbox"/> Trace SQL Cursor (4)	
<input checked="" type="checkbox"/> Trace SQL Fetch (8)	
<input type="checkbox"/> Trace SQL API (16)	
<input type="checkbox"/> Trace SQL Set Select Buffer (32)	

Trace Value: 4175

Figure 9-13. Selecting trace options

200 CHAPTER 9 ■ PERFORMANCE METRICS

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 1 millisecond 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,wmps);
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_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.

