

OracleTM8 Server

Joe Greene, Advanced Information
Systems, Inc., et al.

Expert Insight

The book presents focused explanations of the core features and complexities of Oracle8 Server, including:

- Oracle Database Architecture
- Installation, Upgrades, and Migration
- Oracle Components and Objects
- Oracle8 Administration
- Oracle8 and the Web
- Advanced Implementation Techniques

Authoritative Advice

Implement the advice from some of the industry's leading Oracle8 experts and meet all of your Oracle8 Server needs

SAMS

Unleashed

THE COMPREHENSIVE SOLUTION!

Oracle8 Server

Joe Greene, Advanced Information Systems, Inc., et al.

SAMS

201 West 103rd Street,
Indianapolis, Indiana 46290



Unleashed

Copyright © 1998 Sams

All works in Chapter 6 reproduced herein are Copyright© Oracle Corporation 1997. All rights reserved.

Oracle Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs or program segments that are included. This manual is not a publication of Oracle Corporation, nor was it produced in conjunction with Oracle Corporation.

All other chapters are Copyright© by Sams Publishing.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-31207-7

Library of Congress Catalog Card Number: 97-68737

Printed in the United States of America

First Printing: June 1998

00 99 98 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Oracle is a registered trademark, and Oracle8, Oracle7, Personal Oracle, Personal Oracle Lite, Designer/2000, Developer/2000, Programmer/2000, Oracle Objects, and Oracle Discoverer are trademarks or registered trademarks of Oracle Corporation.

EXECUTIVE EDITOR

Rosemarie Graham

ACQUISITIONS EDITOR

Rosemarie Graham

DEVELOPMENT EDITORS

Todd Bumbalough

Marla Reece-Hall

MANAGING EDITOR

Patrick Kanouse

SENIOR EDITOR

Elizabeth A. Bruns

COPY EDITORS

Nancy Albright

Patricia Kinyon

Tonya Maddox

Sean Medlock

San Dee Phillips

Linda Siefert

INDEXER

Rebecca Salerno

TECHNICAL EDITOR

Jonathan Gennick

PRODUCTION

Carol Bowers

Ayanna Lacey

Gene Redding

Overview

Introduction 1

PART I ORACLE8 SERVER 5

- 1 Oracle8 7
- 2 Oracle Database Architecture 15
- 3 Oracle8 Network Computing 31
- 4 Oracle8 Servers and Options 53

PART II INSTALLATION, UPGRADE, AND MIGRATION 73

- 5 Implementation Planning—PC to Enterprise Server 75
- 6 Installation and Upgrades 101
- 7 Migrating from Non-Oracle Databases 121
- 8 Coexisting with Non-Oracle Databases 145

PART III ORACLE8 COMPONENTS AND OBJECTS 169

- 9 Oracle Processes 171
- 10 Oracle Memory Areas 185
- 11 Oracle Files 199
- 12 Oracle Database Objects 217
- 13 Oracle System and Object Privileges 239
- 14 Roles and Grants 267
- 15 Object-Oriented Extensions in Oracle8 285
- 16 Replication 311
- 17 Query and Transaction Processing 337
- 18 Supplied PL/SQL Packages 371

PART IV ORACLE8 ADMINISTRATION 415

- 19 Oracle8 Administrative Requirements 417
- 20 Routine Maintenance and Scheduling Tasks 447
- 21 Administration Using Enterprise Manager 483
- 22 Oracle8 Tools 523
- 23 Backup and Recovery 557

24	Oracle8 Database Tuning	585
25	Oracle8 Application Tuning	623
PART V ORACLE8 AND THE WEB 639		
26	ODBC/JDBC	641
27	Oracle Web Application Server	665
PART VI IMPLEMENTATION OF ORACLE8 715		
28	Oracle8 Cartridges	717
29	Using Oracle8 Objects in PL/SQL and SQL	739
30	Data Warehouses	747
31	Large Online Transaction Processing Systems	781
32	Media and Complex Data Servers	815
33	Working with Very Large Databases	835
INDEX 863		

Contents

INTRODUCTION 1

PART I ORACLE8 SERVER 5

1 ORACLE8 7

The Latest Oracle	8
Major Improvements.....	10
The Oracle8 Product Family.....	13
Fitting Oracle8 into an Application Environment	14
Summary.....	14

2 ORACLE DATABASE ARCHITECTURE 15

Database Architecture.....	16
Instances	17
Oracle Processes	18
User Processes	19
Data Writing Processes	19
Logging Processes	19
Monitoring Processes	19
Memory and Speed.....	20
Software Code Areas.....	20
System Global Area (SGA).....	20
Program Global Area (PGA).....	21
Sort Areas	21
Disk Storage	21
Data Files.....	22
Log Files	23
Supporting Files.....	23
Control Files	23
SGA Definition File	24
Log and Error Files	24
Interfacing with the DBMS	24
Multiprocessor Configurations	26
Fault Tolerance	26
Microsoft Windows NT Configurations	27
Oracle8 Under Windows NT.....	28
Summary.....	29

3 ORACLE8 NETWORK COMPUTING 31

The Emergence of Network Computing.....	32
Convergence	32

The Web Changes Everything	34
Pitfalls of PC Computing	35
Network Centricity and Network Computing	36
The Appeal of Network Computing	37
Emerging Technologies.....	37
The Benefits of Object Technology.....	37
Java	38
CORBA	40
Enterprise JavaBeans	42
An Architecture for Network Computing.....	44
Network Computing Architecture	44
From 2-tier to 3-tier Computing	44
Client	45
Application Server.....	47
Database Server	48
The Network Computer	49
A Typical NC.....	50
The NC Reference Specification.....	51
Open Issues	51
Summary	52
4 ORACLE8 SERVERS AND OPTIONS	53
So Many Options—So Little Time.....	54
Scale—Bambi Meets Godzilla	54
Architecture—Welcome to the Jungle	54
Utility—Taking Care of Business	55
The Oracle8 Server	55
The Movement Towards Object-Oriented Paradigms	55
The Growing Complexity of Computer Architecture	56
The Proliferation of Internet Business	58
Oracle8 and Data Warehousing	60
Oracle8 Enterprise Edition—Luxury and Performance	62
Enterprise Manager Performance Pack	62
Incremental and Parallel Backups	63
Point In Time Tablespace.....	64
Bit-Mapped Indexes	64
Parallel Query	66
Parallel DML.....	66
Parallel Index Scans	67
Parallel Bit-Mapped Star Query Joins.....	68
Oracle Lite—Tastes Great, Less Calories.....	68

Operating System Wars—UNIX Versus NT.....	70
Microsoft NT and How It Compares to UNIX.....	70
Summary.....	72
PART II INSTALLATION, UPGRADE, AND MIGRATION	73
5 IMPLEMENTATION PLANNING—PC TO ENTERPRISE SERVER	75
Overview of the Installation Planning Process.....	76
Starting with Business Needs	77
The Installation Documentation	79
The README File	81
Developing Your Own Installation Plan	82
Some Good Reviewers	84
System Hardware	85
Software Module Planning	85
Optimal Flexible Architecture (OFA).....	86
Administrative Directories	87
Local Directories	88
Data and Log Directories	88
Memory Allocation and Alternatives	88
Logical Database Design	89
Physical Database Design.....	89
RAID and Other Storage Alternatives	90
Processing Options	92
Upgrades Versus New Installations	92
Other Factors to Consider	93
The Backout Plan	94
Lining Up Support.....	95
A Sample Plan	96
Summary.....	99
6 INSTALLATION AND UPGRADES	101
Overview of the Installation Process for Sun Solaris.....	102
Hardware and Software Requirements	102
Software Requirements for Networking Products	104
Optimal Flexible Architecture (OFA)	104
Installation Preparation and Installation	104
Installing Documentation	106
Post-Installation Steps for Oracle8 Server	106
New Installations—Recovery Manager.....	106
New Installations—Time Series Cartridge	107
Migration to Oracle8.....	108
Copying Data.....	109
Develop a Testing Plan	109

Migration Utility	109
Migration Preparation	111
Backup of the Existing System	112
Common Problems and Troubleshooting	117
Summary	119
7 MIGRATING FROM NON-ORACLE DATABASES 121	
System Migration—The Sum of All Our Fears	122
The Realities of System Migration	122
The Myth of ANSI SQL	124
The Great Operating System Barrier.....	126
Oracle Alliance—Design and Migration Services to the Rescue	126
Common Migration Scenarios	127
Downsizing	128
Front-End Considerations	129
Database Considerations	130
Management Considerations	134
Using Oracle's Data Conversion Tool—DCT	135
What the Conversion Tool Is and What It Is Not	135
A Step-by-Step Tour of a Data Migration Using the DCT	136
Migrating a Front-End Using DCT and Oracle*Forms 4.5	141
Oracle DCT—Is It Worth It?	142
Summary	142
8 COEXISTING WITH NON-ORACLE DATABASES 145	
Sleeping with the Enemy	146
Different Operating Systems	146
Different Hardware Vendors	147
Different RDBMS Vendors	147
Different Paradigms.....	148
The Gateway to Everywhere.....	149
What Kind of Gateway Do You Need?	149
Coexistence and Replication.....	164
Replicating What Is Different	164
Features of Oracle Replication Services	166
Summary	166
PART III ORACLE8 COMPONENTS AND OBJECTS 169	
9 ORACLE PROCESSES 171	
Map of the Oracle Processes	172
The Main Oracle Processes	172
System Monitor	173
Process Monitor.....	175

Database Writer	176
Log Writer	178
Optional Processes	180
Archiver	180
Recoverer	180
Lock Writer	180
Dedicated Server Processes	181
Net8 Processes	181
Parallel Query Processes	182
Web Server Processes	182
Summary	183
10 ORACLE MEMORY AREAS 185	
Uses of Memory Within Oracle8	186
Virtual Memory	187
System Global Area	187
Sharing Memory	187
System Global Area (SGA)	188
Database Buffer Cache	189
Shared Pool	191
Program Global Area (PGA)	194
User Work Spaces	195
Oracle Application Software Memory	196
Summary	197
11 ORACLE FILES 199	
Oracle Files and Their Purposes	200
Data Files	201
Log Files	204
Control Files	208
Initialization and Configuration Files	209
Log and Trace Files	212
Automatic Resizing of Files	213
Distribution of Data for Tuning	214
Standard File Locations	214
Summary	216
12 ORACLE DATABASE OBJECTS 217	
Oracle8 Database Objects	218
Tables	220
Indexes	223
Views	227
Synonyms	228
Stored Procedures and Packages	229
Clusters	230

	Sequences	231
	Tablespaces	232
	Partitions	235
	Object Datatypes	235
	Constraints	236
	Summary	237
13	ORACLE SYSTEM AND OBJECT PRIVILEGES	239
	Overview of Oracle Security	241
	System Privileges	244
	Object Privileges	246
	Table Privileges	250
	View Privileges	251
	Sequence Privileges	251
	Procedure Privileges	252
	Column Privileges	252
	User Privileges	252
	Developer Privileges	253
	DBA Privileges	254
	The Database Maintenance Privileges	259
	Monitoring Privileges	260
	Default Privileges	261
	Connecting Internally	261
	Dummy Object Owners	261
	A Typical Privilege Scheme	262
	Summary	266
14	ROLES AND GRANTS	267
	Managing Grants and Roles	268
	Grants	268
	Roles	271
	Grants Without Roles	274
	Use of Scripts to Capture Privilege Sets	275
	Enterprise Manager Role and Grant Management	276
	Command Line Role and Grant Management	276
	A Typical Privilege Scheme	276
	Summary	284
15	OBJECT-ORIENTED EXTENSIONS IN ORACLE8	285
	Introduction	286
	The Basic Elements of an Object	286
	Methods	287

Object Ordering	291
Map Methods.....	291
Order Methods.....	292
Type Versus Instantiation of a Type	294
REFS	296
Nested Tables and VARRAYS	296
Nested Tables.....	297
VARRAYS	298
Using Base Types	299
Object Views	301
INSTEAD OF Triggers	302
Large Objects	303
BLOB	305
CLOB	305
NCLOB	305
BFILE	305
Summary.....	309
16 REPLICATION 311	
Replication: "Why Ask Why?"	312
Replication and the Data Warehouse	314
Read-Only Replications and Snapshots	317
Snapshot Logs	320
Remote Databases	321
Complex Replication—Distributed Databases	321
Replication and the World of Virtual Databases	321
Creating a Replicated Environment "Replication God"	323
Managing Your Replication and the Mystery of the Two-Phase Commit.....	326
Conflict Resolution.....	327
Types of Conflicts	328
Managing Update Conflicts.....	329
Creating Column Groups.....	330
Defining a Conflict Resolution Method for a Column Group	330
Survivability	331
The Advantages of Oracle8 and the Replication Manager.....	333
Summary	336
17 QUERY AND TRANSACTION PROCESSING 337	
Transaction Processing	338
SQL Statement Processing	340
Cursor Creation	340
Statement Parsing	340

Query Processing	342
Bind Variables	343
Statement Execution	343
Parallelization	343
Fetch Rows to Be Returned	343
Using the Oracle Optimizer	345
How the Optimizer Works	345
Optimizer Initialization Parameters	346
Optimization Methods	347
Using the ANALYZE Command	350
How to Run the ANALYZE Command	350
Data Dictionary Statistics	353
Analyzing SQL Statements	356
Using EXPLAIN PLAN and SQL Trace	357
SQL Trace	358
Using EXPLAIN PLAN	360
Designing New SQL Statements	363
Using Indexes	363
Clusters	365
Hash Clusters	366
Range Partitioning	367
Packages, Procedures, and Functions	368
Using Hints	368
Summary	369
18 SUPPLIED PL/SQL PACKAGES 371	
Introduction to Supplied Packages	372
Creating the Packages	372
Package Summaries	373
Reading From and Writing To Files	373
Working with LOB Data	379
Issuing Dynamic SQL and Data Definition Language Statements	387
Communicating Between Processes	394
Advanced Queuing	402
Scheduling Jobs	410
Summary	413
PART IV ORACLE8 ADMINISTRATION 415	
19 ORACLE8 ADMINISTRATIVE REQUIREMENTS 417	
Overview of Oracle8 Administration	418
Backup and Recovery Concepts	419
Security and Access Concepts	421

Overview of Oracle8 Permissions and Profiles.....	422
Overview of SQL*Net, Net8, and Connection Manager	422
Tablespace Maintenance Concepts.....	424
Importance of Organization.....	426
Tuning Through Distribution.....	427
Sizing Defaults	427
Fragmentation and Design Considerations	428
Schema Maintenance Concepts	431
Object Maintenance.....	431
Referential Integrity.....	431
Loading/Unloading Tables	432
Sizing Defaults	435
Estimating the Size of Objects	436
Tuning Concepts	441
Who Is Responsible for Tuning Tasks?.....	441
Four Levels of Tuning	442
Miscellaneous Tasks	443
Monitoring	443
Troubleshooting	444
Summary	446
20 ROUTINE MAINTENANCE AND SCHEDULING TASKS 447	
User, System, and Batch Processing	449
User Processing	449
System Processing	450
Monitoring Processes	451
Initializing Processes	453
Scheduling Batch Processes	453
Auditing	453
Enabling and Disabling Auditing	454
The Operating System Audit Trail	455
The Database Audit Trail.....	455
The AUDIT Command.....	457
Types of Auditing	458
Audit Options	458
Managing the Audit Trail	463
Sample Audit Queries	465
Common Monitoring Scripts	467
Types of Monitoring	468
What to Monitor	468
Sample Monitoring Scripts	468
Monitoring with Oracle Tools	473

Scheduling Jobs	474
Managing Job Queues	475
Submitting and Manipulating Jobs	477
Job Queue Information in the Data Dictionary	480
Summary	482

21 ADMINISTRATION USING ENTERPRISE MANAGER 483

Introduction to Oracle Enterprise Manager	484
Installing Oracle Enterprise Manager	485
Configuring Enterprise Manager	486
Building the Repository	486
Configuring the Intelligent Agent	487
Discovering New Services	489
Setting Preferences	490
Managing Events	491
Creating and Modifying Event Sets	491
Registering an Event Set	493
Event Notification	496
Managing Jobs	497
Preparing to Run Jobs	497
Creating a Job	498
Creating a Fixit Job	500
Managing Database Instances	501
Preparing for Remote Administration	501
Shutting Down an Instance	502
Starting an Instance	503
Editing Stored Configurations	503
Managing Schema Objects	504
Creating Tables	504
Creating Object Types	505
Creating a VARRAY	506
Creating a Nested Table	508
Creating an Object Table	508
Managing Security	510
Managing Storage	511
Diagnostic and Performance Tools	512
Oracle Performance Manager	512
Oracle Lock Manager	514
Oracle Expert and SQL Analyze	515
Sample Net8 Configuration Files	517
Summary	521

22 ORACLE8 TOOLS	523
Overview of Oracle8 Tools	524
Server Manager	524
Export/Import	525
Data Movement	525
Incremental Backups	528
SQL*Loader	528
SQL*Plus	529
SQL*Plus Execute Commands	532
SQL*Plus Editing Commands	533
SQL*Plus Formatting Commands	534
Miscellaneous Commands	536
SQL*Plus Reporting Techniques	537
Advanced Reporting Techniques	539
SQL*Plus Additional Functionality	548
Third-Party Products	551
Database Administration	551
Data Movement	552
Tablespace Reorganization	552
Tuning Aids	553
Monitoring	554
Summary	555
23 BACKUP AND RECOVERY	557
Overview of Oracle Data Availability Requirements	558
Acceptable Data Loss	559
24x7 Needs	559
Data Warehouse Requirements	559
Scheduled Maintenance	560
Overview of Oracle Backup and Recovery Processes	560
ARCHIVELOG Mode	562
Cold and Hot Backups	564
Incremental Backups	565
High Availability Options	567
Backup Schemes	571
Overview of Various Backup Schemes	571
Deciding Which Scheme Is Best	573
Automating Backup Schemes	573
Summary	583
24 ORACLE8 DATABASE TUNING	585
Tuning Overview	586
Who Is Responsible?	586
Proactive Tuning Versus Reactive Tuning	586

Levels of Tuning	586
Operating System-Level Tuning	587
Oracle RDBMS-Level Tuning	592
Database Design-Level Tuning	598
Table/Index Relationships and Tablespaces	599
Partitioning	600
Controlling Fragmentation	603
SQL-Level Tuning	605
Application Code	606
Indexing	608
Parsing	611
Optimizers	611
SQL Tuning	616
Tuning Checklist	621
Summary	621
25 ORACLE8 APPLICATION TUNING 623	
Factors Designers Can Control	625
The Rule-Based Optimizer	627
The Cost-Based Optimizer	629
Execution Plans	631
Hints	634
Index Design	635
General Guidelines	636
The Value of Experimentation	637
Summary	638
PART V ORACLE8 AND THE WEB 639	
26 ODBC/JDBC 641	
Overview of ODBC	642
ODBC's Purpose	642
ODBC's Evolution	642
RDO, DAO, OLE DB, ADO, and Other Acronyms	643
Oracle's ODBC Evolution	643
ODBC Architecture	643
ODBC Setup, Usage and Limitations	644
Oracle8 ODBC Setup	645
Connecting to Oracle8	648
Oracle ODBC Driver Capabilities and Limitations (V8.0.3)	648
ODBC Functions Not Supported	649
SQL Syntax Not Supported	649

Oracle8 ODBC Optimization and Troubleshooting	649
Performance, General	649
Update Performance	650
Select Performance	650
Troubleshooting via Logging	651
Last Words on ODBC	652
Overview of JDBC	652
JDBC's Purpose	653
JDBC's Evolution	653
Oracle's JDBC Evolution	653
JDBC Thin	654
SQLJ	654
JDBC Architecture	654
JDBC Setup, Usage, and Limitations	655
Oracle8 JDBC Setup	655
Installing Oracle8 JDBC Drivers	655
Configuring Oracle8 JDBC Drivers	655
Capabilities and Limitations	657
JDBC, General	657
JDBC Thin	658
Oracle-Specific Functions	658
REFCURSORS as ResultSets	658
Execution Batching	658
PL/SQL Stored Procedures	659
Database Metadata	660
Streaming Data	661
prefetch	662
Define Query Columns	662
Last Words on JDBC	663
Summary	664
27 ORACLE WEB APPLICATION SERVER 665	
Background: The Internet, the WWW, and Intranets	666
Web Architecture	667
Hypertext Transport Protocol (HTTP)	668
Hypertext Markup Language (HTML)	671
Java	676
JavaScript	676
Network Computing Architecture	676
Background	678
Oracle Web Application Server Architecture	681
Web Listener	682
Dispatchers	682

Web Request Broker (WRB)	682
Cartridges	687
Accessing Non-Oracle Databases	702
Web Security	703
User Authentication	703
Persistent User Verification	704
Secure Sockets Layer (SSL)	705
General Guidelines	707
Developer/2000 and the Oracle WAS	708
Forms	708
Reports	710
Graphics	711
Designer/2000 and the Oracle WAS	711
WAS Product Purchase	712
Standard Version	712
Advanced Version	712
Web Developer Suite	713
The Future of Oracle Web Application Server	713
Summary	714

PART VI IMPLEMENTATION OF ORACLE8 715

28 ORACLE8 CARTRIDGES 717

Overview of Oracle8 Cartridges	718
NCA Cartridges	718
Cartridge Architecture	723
Application Designing Using NCA Cartridges	723
Data Cartridge Components	724
Developing Cartridges	726
Designing a Data Cartridge	727
Oracle8's Extensibility Feature	727
External Procedures	730
Example: Step by Step Data Cartridge Development	732
Testing the Developed Data Cartridge	734
Deploying Cartridges	734
Data Cartridge Installation and Operation	735
Definition of Libraries	735
Installation Flow	737
Summary	737

29 USING ORACLE8 OBJECTS IN PL/SQL AND SQL 739

The Advantages of Using PL/SQL Within Java	740
Different Ways That Java Can Package PL/SQL	743

	The Language of Cartridges	743
	RMI—Distributed Computing Tool	744
	The Future—J/SQL	745
	Summary	745
30	DATA WAREHOUSES 747	
	Introduction	748
	What Is a Data Warehouse?	748
	Why Would a Company Need a Data Warehouse?	748
	The Six Steps to Creating a Data Warehouse	750
	Document the Operational Environment	750
	Choosing the Data Warehousing Technologies	751
	Designing the Data Warehouse Model	752
	The Star Schema	752
	Designing the Data Refresh Process	757
	Scalability	758
	Utilizing Oracle8 Features in the Data Warehouse	758
	Monitoring Tools	760
	Develop the Extraction and Scrubbing Routines	760
	Auditing Logs	761
	Full Refreshes with Data Selection Performed Inside the Data Warehouse	761
	Code Changes in the Existing Applications	762
	What About Data Marts?	765
	Metadata	768
	System Metadata	768
	Management Metadata	768
	Business Metadata	769
	Executive Information Systems	770
	Easy to Use	770
	Easy to Access	771
	Fast	771
	Client- or Server-Based	774
	External Data	775
	Data Mining (Panning for Gold)	775
	Data Mining Models	776
	Summary	779
31	LARGE ONLINE TRANSACTION PROCESSING SYSTEMS 781	
	Online Transaction Processing (OLTP) Overview	782
	OLTP Database Design	783
	Parallel Server Architecture Review	783
	Parallel Server Design Practices	790

Database Configuration Issues	792
UNIX Configuration Issues	797
Database Schema Design Practices	802
OLTP Application Development.....	803
Application Design and Coding Practices.....	803
Application Security	807
OLTP Administration and Maintenance Considerations	809
Backups and Recovery Considerations	809
Batch Data Loading	810
Object Fragmentation	810
Periodic Maintenance	810
OLTP Implementation Issues	811
Stress Testing	811
Project Management and Planning Issues.....	812
Summary	812
32 MEDIA AND COMPLEX DATA SERVERS 815	
What Is a Media Server?	816
New Uses of Media Objects That Will Change Business	818
Presentations	819
Online Training	819
Help Desk Support	819
Kiosks	819
Video on Demand (VOD)	820
Home Shopping	820
Electronic Banking	820
Personalized News Service	820
Games	820
Reference	821
Oracle's Answer to Media Data	821
The Oracle Media Server	821
Oracle Media Net	825
Oracle Media Objects	825
The Components of Oracle's Media Objects	827
Oracle Video Custom Control	829
Oracle's Media Cartridge	833
Summary	834
33 WORKING WITH VERY LARGE DATABASES 835	
Introduction.....	836
Oracle8 and High-End Database Systems	836
Partitioned Tables and Indexes	837
Managing a Large Number of Users.....	837

Deferred Transaction Support	838
Parallel Query Enhancements	838
Parallel Server Enhancements	838
Oracle8 Features for Large Databases	839
Table Partitioning	839
Partitioned Indexes	842
Local and Global Indexes	843
Index-Only Tables	845
Parallel DML (PDML)	846
Bitmap Indexes	847
Index Fast Full Scans	849
Reverse Key Indexes	849
Tuning for Very Large Databases	850
Transaction Management in VLDB Systems	851
Rollback Segment Considerations	851
Archiving Considerations	852
Operation of a Large Database	854
Rollback Segments Design	855
Backup and Recovery Using Recovery Manager (RMAN)	856
types Within types	860
Object Views	861
VARRAY	862
Summary	862
INDEX	863

3 CHAPTER

Oracle8 Network Computing

by Cameron O'Rourke

IN THIS CHAPTER

- The Emergence of Network Computing 32
- Emerging Technologies 37
- An Architecture for Network Computing 44
- The Network Computer 49

This chapter focuses on how Oracle fits into the emerging world of network computing. First, the industry factors that have led up to network computing are discussed; then you survey some of the enabling technologies. This will be followed by an overview of Oracle's Network Computing Architecture and supporting products, and finally, some words on the Network Computer.

The Emergence of Network Computing

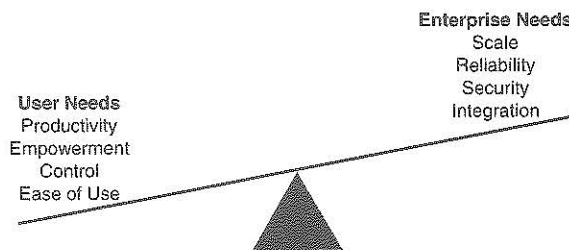
The information industry is maturing, and network computing is the convergence of much that we have learned during the last 30 years of computing. In this section, you explore the impact of new technology and other unique conditions that have brought about network computing.

Convergence

Since departments in large organizations began to purchase minicomputers to supplement the services provided by the glass house, information managers have been in a struggle to balance the needs of the enterprise with the needs of the user. The arrival of the personal computer further added to the conflict by setting new expectations for graphical interfaces, personal control, and ease of use. As shown in Figure 3.1, many information managers feel that the priorities have shifted in ways that are unhealthy for the enterprise.

FIGURE 3.1.

Are we unbalanced?



Now we are in the midst of a new convergence of attitudes and technology, as shown in Figure 3.2.

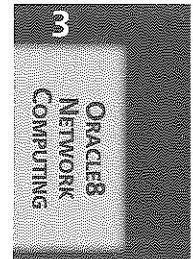
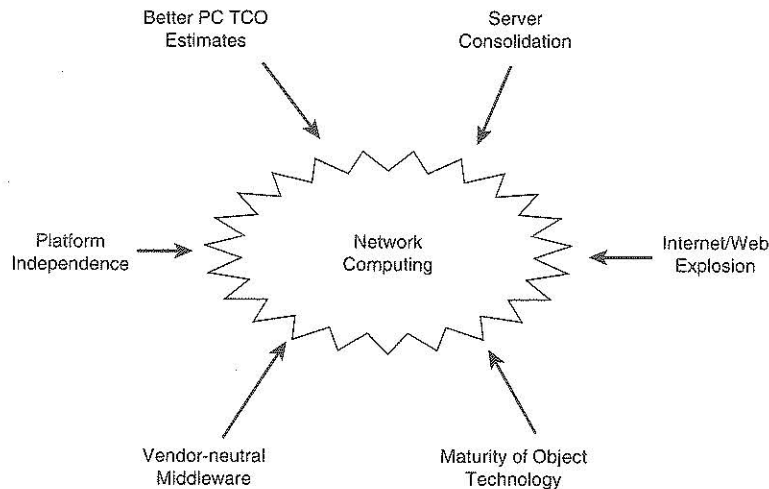
First, there is a unique set of conditions in the information industry. Both users and information managers now have some experience with, and an understanding of, the limitations of existing technology:

- Many businesses have begun to look closely at the total cost of ownership (TCO) of their personal computer purchases. Some are putting off upgrades until a better alternative can be found.
- Some businesses have found that an unintended consequence of having lots of low-cost servers is astronomical management costs. The topic of server consolidation is popular at seminars and in magazine articles. Perhaps coincidentally, mainframe computers are selling better than ever.
- The Web has shown end users how they can get real value by using services provided for them on a network. There may be some alternative to do-it-yourself computing.

Second, there is a wave of emerging technology that is network-centric, object-oriented, and based on standards. For the first time we have industrywide, vendor-neutral standards for data formats, code, middleware, and components. This could be as significant as interchangeable parts were to the industrial revolution, or as the integrated circuit was for the electronics industry.

FIGURE 3.2.

A convergence of circumstances that open the way for network computing.



These events signal a major shift that represents a maturing of the information industry, because a new style of computing, called network computing, is emerging.

Network computing represents the best of the last 30 years of computing technology. Some have called it the fourth great wave of computing (see Figure 3.3). Whether this proves true or not, network computing is refreshing, because the industry is addressing the practical needs of the enterprise, rather than simply hyping the latest new technology.

FIGURE 3.3.
*Four generations
of computing.*



Network computing promises to finally put an end to the struggle between the needs of the enterprise and the needs of the user, and is computing's next big evolutionary step.

The Web Changes Everything

It is amazing to consider the very rapid and wide-scale adoption of the Web. The Web is a great example of the social, cultural, and economic explosion that can take place, given an open standard.

From the Web, users get what they want: information, immediacy, ease-of-use, communications, and collaboration. Enterprises get what they want: a single point of control, low cost, simple administration, easier maintenance, better security, and at least a shot at systems reliability and scalability.

Whole new categories of information and services have been created. Web-enabled office suites now allow easy document publishing. Many businesses are quickly Web-enabling their internal applications. Why? Because everyone has a browser, it is easy to use, and it requires a minimum of hassle.

This underscores a big point: Most people simply need access to information, and they need to communicate. Most people don't need large amounts of personal computational power (except, perhaps, software developers, artists, and scientists); they need better access.

Prior to the Web, the only way for two computers to talk to each other was by prior arrangement. The Web, based on vendor-neutral networking standards, has made it effortless and transparent to fetch data from any location in the world.

Yet the Web is based on relatively simple technology.

Imagine what might happen if program components were as mobile as HTML and as transparent to invoke as Web pages? What if software components could easily collaborate and recombine over the Web to form more complex applications?

This is only the beginning. Vendors of network computing technology are currently integrating new technologies to deploy mission-critical applications to the Web. The future of the Web will be based on plug-and-play mobile components that will enable the next wave of "killer applications."

Pitfalls of PC Computing

Personal computers have many desirable properties, but few would argue that the PC was architected for enterprise computing. The PC was designed for personal productivity; adapting it to become a good network citizen and work in a large organizational environment has been a long and difficult process.

Nevertheless, PCs have become the most common client for client/server architecture, even though they are cumbersome to deal with in large numbers. The number of potential hardware combinations in today's PC is almost unlimited. Plus, all users are essentially free to install and remove both application and operating system software from their PCs. PCs have become like fingerprints: No two are ever the same.

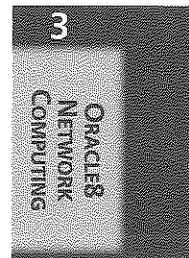
Consequently, 2-tier client/server has had a difficult time becoming the mainframe killer it was predicted to be, because it has run into several limitations of the PC platform. It has become an administrative nightmare and too costly to deploy on a widespread basis. The rapid adoption of the Web as an application platform speaks to this. (I sometimes wonder whether we would find ourselves in this same crisis had X.11 terminals and Motif taken off just a bit faster.)

Even the current state of the art in automated software distribution systems still requires some amount of installation and periodic maintenance at each workstation, a situation that is completely unacceptable to larger organizations with hundreds or thousands of connected users.

Business managers have recently placed much attention on the total cost of ownership (TCO) for desktop computing. The personal computer industry has responded with a raft of initiatives and products designed to try to lower costs by easing the maintenance burden. However, most of these initiatives involve taking some degree of control away from the operator, which has not proven popular with end users, because they don't get anything in return.

But even if users would accept them, managed or locked-down PCs are not enough. In order for users to want to give up control of their personal mainframes, they will have to be provided with a compelling array of services, applications, productivity tools, and personal freedom that is better than what they have today.

To remain competitive, companies must find a way to more quickly build and deploy applications at a lower cost, integrate them with existing systems, and manage them with a smaller staff, handling a much larger base of users with greater security. And, they need to get off the treadmill of constant software upgrades and the continual need for more memory and processor power.



A new architecture for rapidly developing and deploying reliable applications is needed. The computer industry is hard at work on the technologies to make network computing a near-term reality.

Network Centricity and Network Computing

First and foremost, network computing is about standards. Just as plumbers and electricians have developed standard parts, measurements, and construction techniques, so must those who produce computer programs begin to standardize. This is how the information industry, which has become the most important industry in the world, will begin to mature. Object technology and modeling standards, such as the Uniform Modeling Language (UML), will help.

Network computing is about universal access to data. By using standards for data and applications content such as HTML, Java, and XML, it is possible to configure a universal client. This client can be in the form of a variety of hardware devices, including personal computers, graphical terminals, Network Computers, or even mobile personal digital assistants. The client may also function as a terminal to a variety of host types, including UNIX, 3270, and Windows servers.

Network computing is also about universal access to applications. The location at which programs execute should be just as transparent to the end user as the location of a Web page is today. It should be easy to integrate all our business systems so that they can be uniformly accessed from the universal client. Standards for distributed processing and middleware services, such as CORBA, are providing, for the first time, a vendor-neutral way for applications to transparently run anywhere on the network. The choice of server operating system is becoming less relevant as the network becomes a single large computer system (witness the number of new commercial sites that are running versions of Linux and FreeBSD).

Network computing is about network services. The intelligence is in the network rather than all at your desk. High-powered, high-value services are centralized, as are access to massive content and computational power. Personal storage is professionally managed on secure, fast RAID. Powerful, reliable, and secure database servers and enterprise-class applications' platforms guarantee that mission-critical information keeps flowing.

Finally, network computing is about bringing new applications online faster and at lower costs. The buying and selling of companies, acquisitions, mergers, and a high rate of change are the norm. The good news is that standards for component software are rapidly emerging and generating a huge amount of excitement and venture capital in the computer industry. Powerful applications servers will take simple objects such as Enterprise JavaBeans and turn them into reliable, scalable, and robust applications services.

Programmers will not have to constantly reinvent the wheel. And software, or at least common facilities and services, will become plug-and-play, enabling organizations to mix and match best-of-breed components into their information architectures.

The Appeal of Network Computing

For users, network computing promises to relieve the end user from unpleasant tasks such as system configuration, management, backups, installing software, getting everything to work together, and software upgrades. All of this is possible while still providing excellent personal productivity, individuality, and control.

For IT managers, network computing addresses the integration of disparate systems so that a single universal client can access any application or perform any function that is needed. It lowers costs by placing systems management chores in the hands of professional network administrators who can leverage their time and expertise by providing these services for hundreds of users at a time.

For developers, network computing is about rapidly developing and deploying new solutions or new interfaces to existing systems. The technologies lend themselves to rapid prototyping and iterative development. By creating object-oriented components that run on powerful application servers, developers can concentrate on just the functionality needed to solve the business problem at hand, instead of having to constantly reinvent and reimplement infrastructure.

Emerging Technologies

To understand how network computing will achieve all these benefits, you need to know the underlying technology that makes it possible. In this section, you look at briefly the benefits of objects; then you explore Java, CORBA, and Enterprise JavaBeans.

The Benefits of Object Technology

It is assumed that you understand the basics of object technology; however, here is a reminder of some of the benefits that objects bring to a system. All the other technologies covered are object-based, and as such, share in these benefits.

Object technology has been delivering three main benefits to developers and the organizations for which they build applications. Developers of all types, using all sorts of tools on a broad range of projects, have consistently achieved ease of maintenance, ease of extensibility, and reuse:

- **Ease of Maintenance:** Because they are modular, object applications are easier to maintain than procedural applications. There tend to be fewer dependencies



between program components. The program logic can be expressed more naturally and in smaller chunks of code. Objects allow for uniform modeling techniques throughout the development life cycle. It's generally easier to isolate problems and test program revisions.

- **Ease of Extensibility:** The property of polymorphism makes object-oriented applications easy to extend. Developers can subclass existing interface definitions to add new object types. Through the use of common design patterns, existing capabilities continue to function. Integration and testing are so much simpler than with procedural applications.
- **Reuse:** Reuse is the most promising benefit of object technology. Developers have been able to reuse small objects, coarsely grained components or business objects, frameworks and templates, and even entire application designs. Objects enable component models that enable developers to implement applications with much less effort.

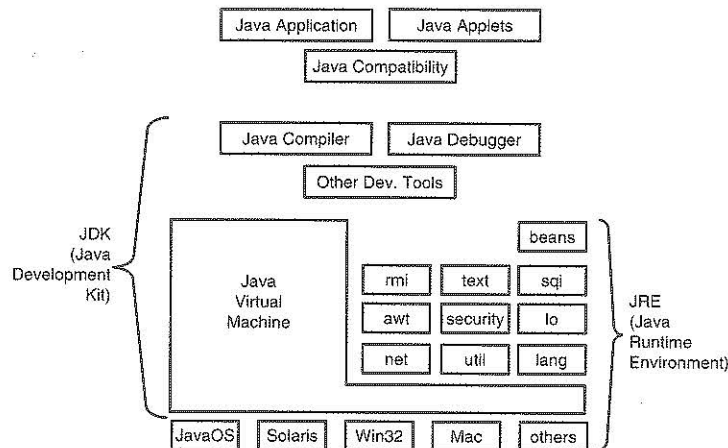
The bottom line is that object technologies allow larger and more complex systems to be built faster, more flexibly, and more reliably.

Java

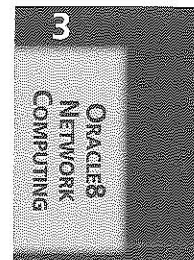
Java has many properties that make it an ideal platform for network computing. Let's look at these in detail and do a bit of comparison with C++:

- Java is a fully object-oriented programming language. Java is designed to look like C++, but Java is far simpler to use. Unlike C++, you cannot use Java to write non-object-oriented programs. Features such as automatic garbage collection make it easier and faster to write solid code.
- Java comes with a large set of classes arranged in packages that you can use in your programs. As shown in Figure 3.4, there are classes for creating graphical user interfaces (awt), handling I/O, providing network support (net), accessing a database (sql), and creating reusable components (beans). Java provides a lot of high-level support for networking and distributed objects. It is easy to make your programs network-aware, and the Java Remote Method Invocation (RMI) API enables a Java program to invoke remote Java objects as if they were local.
- Java is an interpreted language. The Java Development Kit (JDK) includes a Java compiler that produces bytecodes for the Java Virtual Machine (JVM) rather than native machine code. To actually run a Java program, you use the Java Runtime Environment (JRE), either standalone or as part of a Web browser such as Netscape.

FIGURE 3.4.
Components of
the Java platform.



- There are Just-in-Time compilers that boost the runtime performance, and Javasoft's upcoming "Hot-spot" runtime compiler may make interpreted Java as fast as compiled C++.
- The Java bytecodes are architecture-neutral, which allows portability to any system that has a Java runtime. Java also ensures that there are no implementation-dependent aspects of the language specification. This is different from C, where an int type can be 16, 32, or 64 bits, depending on the platform.
- Java is a dynamic language because it has no link phase. Any Java class can be incrementally loaded into a running program and dynamically instantiated at any time. This provides high-performance, incremental loading of programs over a network versus plug-ins and ActiveX. You also can obtain information about a class at runtime using the Reflection API.
- Java is a robust language. It provides built-in support for threads and makes it relatively easy and safe to build high-performance user interfaces and servers. By providing features such as strong-typing, exception handling, and extensive compile-time checking, whole classes of common programming errors are eliminated.
- Java is secure. It assumes the presence of a network. All bytecodes are verified at runtime. Untrusted code is run inside a "sandbox," where it has limited access to the local system resources, such as memory and disk. Java also supports digital signatures based on X.509 certificates.



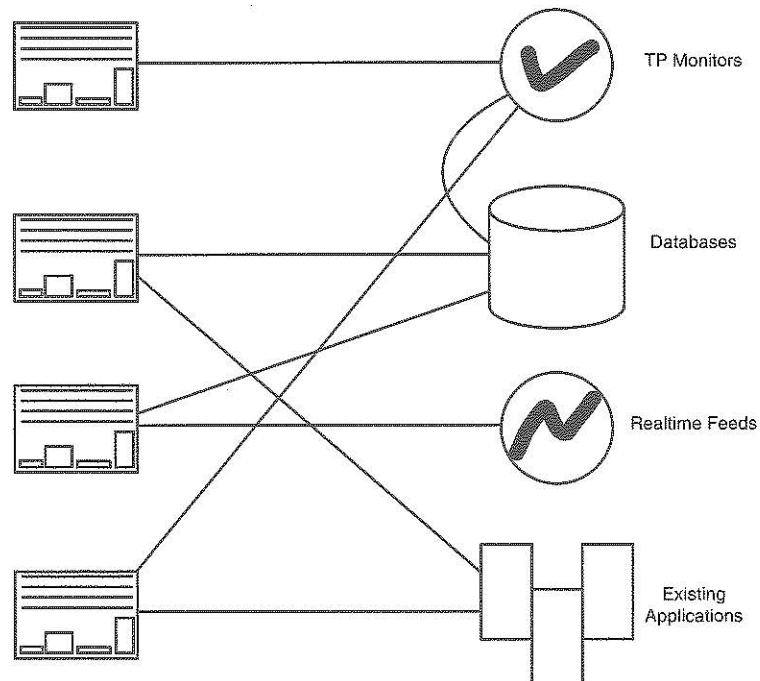
CORBA

Diversity in hardware and software is a fact of life, and our networked environment is becoming more diverse—not less—as computers evolve. Today, businesses spend a large percentage of their technology budgets trying to integrate different elements of their environment into a total solution. Instead of improving their business processes, the majority of their time is spent on infrastructure.

The problem is that every transaction, messaging, database, or system service had its own proprietary API. Every application that used one of these services had to individually implement the API (see Figure 3.5). This led to a morass of nonintegrated systems.

FIGURE 3.5.

Applications have been hardwired to servers using proprietary APIs.



Many good attempts to solve the integration problem have been offered, including remote procedure calls (RPC), message-oriented-middleware (MOM) systems, queuing systems, and TP monitors. However, these products were not able to solve the entire problem, because they were vendor specific, language specific, platform specific, or did not offer enough flexibility.

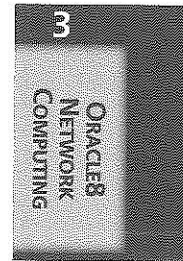
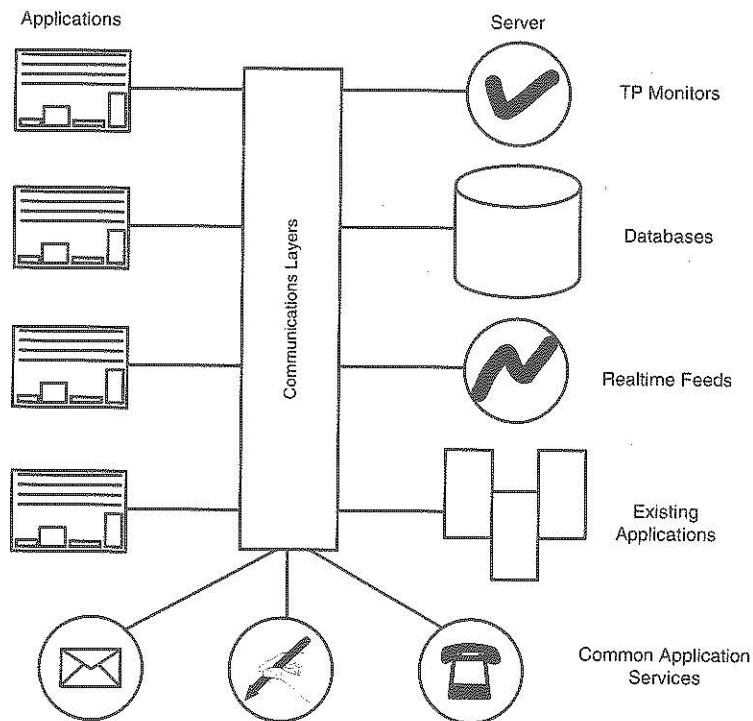
The Common Object Request Broker Architecture (CORBA) provides an open specification with multivendor support to enable objects and components from various platforms to communicate with one another across networks. CORBA provides platform, vendor, operating system, network, language, and location transparency. It is fully object-oriented and provides for dynamic binding and typing.

The CORBA specification is produced by the Object Management Group (OMG), a consortium of over 1,000 companies, including most of the major vendors of systems and software. The OMG was formed in 1989 as a nonprofit organization to create a component-based software market by accelerating object standards.

An Object Request Broker (ORB) is the heart of the CORBA architecture and provides the transport mechanism between objects. As shown in Figure 3.6, it provides a layer of abstraction that allows all applications access to all services. Client applications are coded to call and use services on the network as if they were local. Applications can also browse a repository of available object implementations and invoke them dynamically.

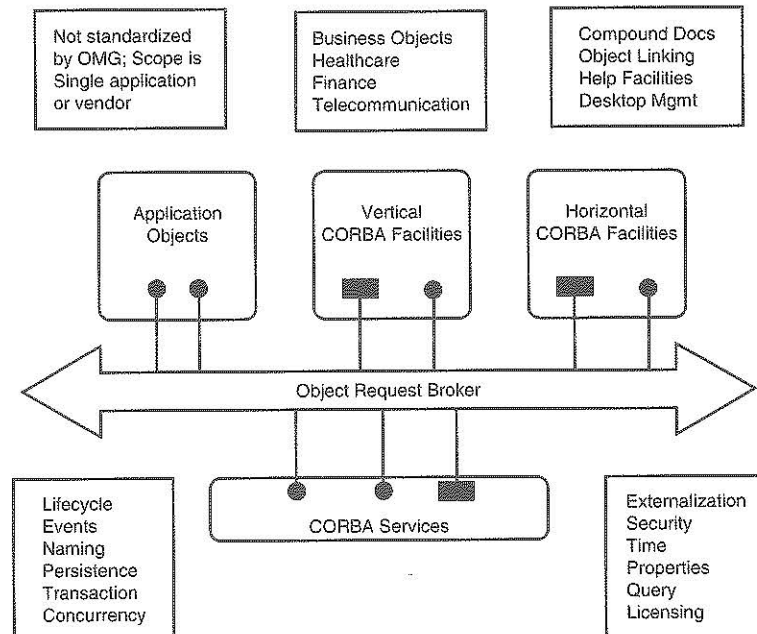
Client applications and services communicate using Internet Inter-ORB Protocol (IIOP), which runs on top of TCP/IP and is the standard way that CORBA components communicate.

FIGURE 3.6.
A common communications layer allows all applications to access all services.



In addition to the ORB and related components, the OMG provides for application objects, common facilities, and common services through its Object Management Architecture (OMA). As shown in Figure 3.7, these higher-level objects and facilities build upon the Object Request Broker foundation and CORBA Services.

FIGURE 3.7.
The Object Management Architecture (OMA).



Common services provide a way to factor out basic functionality from application components. A few of the better-known services are lifecycle, security, naming, directory, persistence, event management, and transactions.

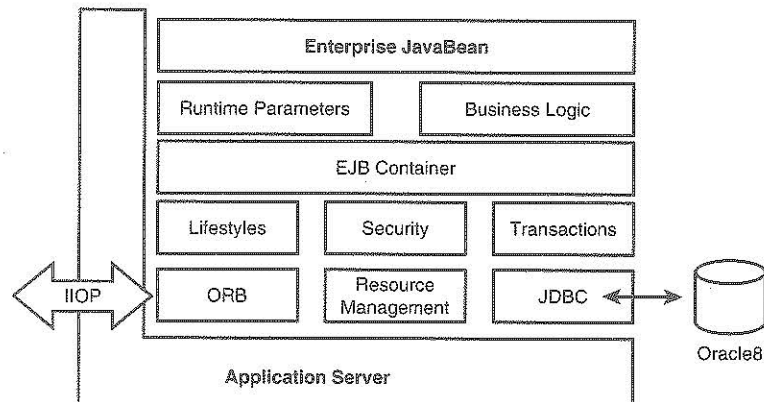
Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a component model for server-side objects that is part of the Java Enterprise API set. EJBs simplify the task of creating server objects that implement business logic for thin-client applications. With EJBs, developers don't have to worry about multithreading, state-management, scalability, distributed operations, concurrency, or many other aspects of system design that don't directly relate to the problem domain.

There are two kinds of EJB: a session bean and an entity bean. *Session beans* are conversational and retain their state for the duration of the session only. They are not shared between clients. *Entity beans* are persistent objects, are shared among many clients, and maintain their state indefinitely.

An application server implements an Enterprise JavaBeans container, which provides services to an EJB. As shown in Figure 3.8, those services may include lifecycle support, security, transactions, communications (ORB), resource management, and database access (JDBC).

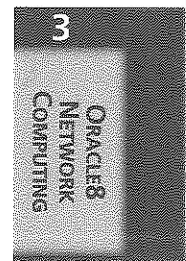
FIGURE 3.8.
*Enterprise
JavaBeans archi-
tecture.*



A key benefit of the EJB specification is that existing vendors of network servers can become interoperable and host industry-standard components via the EJB container abstraction. Vendors of transaction monitors, database servers, message queuing systems, security management systems, and other network services can expose their native capabilities simply by implementing an EJB container on top of their existing product.

Enterprise JavaBeans will help build the market for both components and containers, because vendors will compete on functionality rather than by locking out competitors with proprietary APIs. Application servers can differentiate themselves by offering different sets of price points and capabilities in areas such as

- Thread and process management
- Network access
- Transaction management
- Authentication and authorization methods
- Resource pooling
- Concurrency management
- Support for clustered systems
- Support for high availability



An Architecture for Network Computing

The emergence of vendor-neutral enabling technology is a great boon to the computing industry and will benefit all organizations who rely on information technology. However, the fact that you have an open standard says nothing about the implementation quality. Being compliant with a standard such as CORBA, for example, ensures compatibility and interoperability, but does not address a system's reliability, scalability, and performance. That's where Oracle's products and experience come into play. In this section, you look at Oracle's implementation of network computing.

Network Computing Architecture

Oracle has been in the business of helping organizations manage their information assets for the last twenty years and has a unique perspective on network computing. Oracle provides a highly reliable and scalable server platform, but it became increasingly frustrated by the limitations of the PC. The complexity and high cost of the PC was limiting the deployment of client software that could access its servers.

Oracle introduced the concept of network computing in 1996 to address its customers' concerns over administration cost, system reliability, integration, and inexpensive access to data.

Unfortunately, the Network Computer was also introduced at about the same time as network computing, and this has caused quite a large amount of confusion. As you'll see in a later section, the Network Computer is a particular type of client device. It is not a requirement for network computing; rather it is the icing on the network computing cake.

Oracle's Network Computing Architecture is a vision, brand, and product road map for enabling network computing. It does not depend on any particular type of computer or client device. It is based on the vendor-neutral, emerging Internet technologies such as Java, CORBA, and Enterprise JavaBeans that were discussed earlier. It is a way of arranging the technologies in such a way as to be useful for driving product development and helping customers make architecture decisions.

From 2-tier to 3-tier Computing

In the traditional 2-tier client/server model, the client (typically a PC) was responsible for both the user interface, as well as most of the application logic. The 2-tier approach forces a lot of complexity and computations onto the user's desktop and is generally referred to as *fat client*.

NCA is essentially a 3-tier distributed processing architecture defined by open standards and built upon Oracle's server products. The user interface is logically and physically separate from the application logic, which is run on a "middle-tier" application server.

Before you explore each platform, here are a few general attributes of the 3-tier model, shown in Figure 3.9:

- The client can be any device, rather than just a PC. Anything that can, at minimum, render HTML can be a network computing client. The client can even be a mobile device.
- Clients use either HTTP or IIOP (the CORBA standard protocol), rather than a proprietary client/server protocol, depending on whether they require lightweight or session connectivity.
- Application logic is moved to the middle tier. The middle tier can be either a Web server or an application server. The middle tier provides an abstraction layer for data and system services.
- In addition to the RDBMS, existing systems, TP Monitors, messaging systems, and so forth can be incorporated.

FIGURE 3.9.
General NCA architecture.



As NCA is based on component technology; each of the three tiers provides a framework for application-specific components and access to common services and facilities.

As shown in Figure 3.10, application-specific logic is programmed as components, and the runtime infrastructure is supplied by the server platforms. Common services are factored out of each application—they are provided through a common interface for all applications.

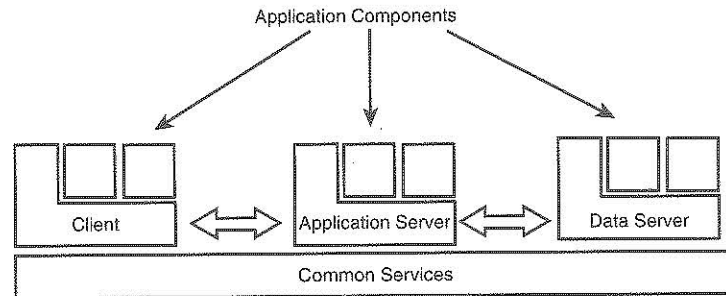
Next, we'll discuss each of the three tiers.

Client

The role of the client tier is to render a nice user interface, be responsive, and be easy to learn and easy to use. The ideal client UI would give useful indications and feedback, get updates automatically, save changes when offline, protect the work until it is safe, and be available anywhere.

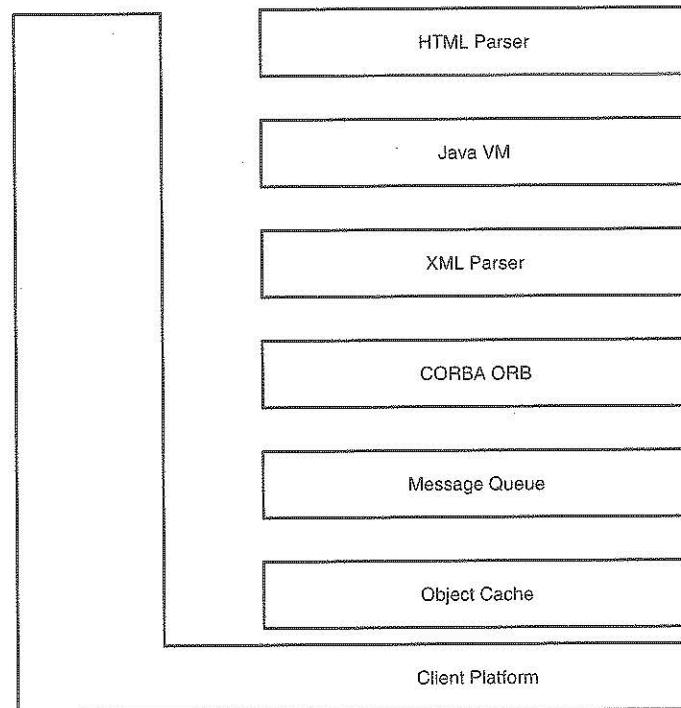


FIGURE 3.10.
NCA platforms support application components and access to common services.



As shown in Figure 3.11, there are various capabilities that may be implemented by any given client device.

FIGURE 3.11.
Capabilities in the client tier.



Any device that can render HTML can potentially be a (limited) NCA client. However, to support Oracle's Developer/2000 server or any other 3-tier application, the device will also require the Java Runtime Environment.

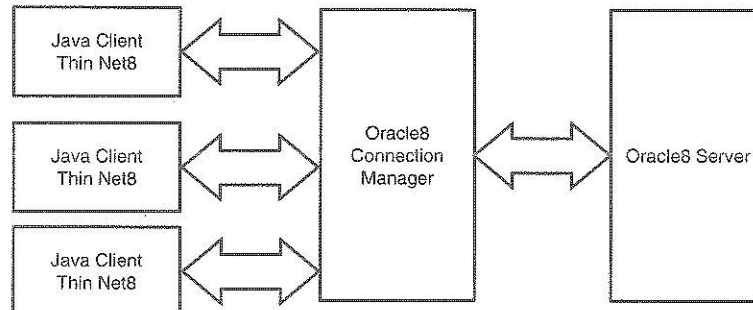
Future applications will likely rely on a CORBA ORB for communications and may use XML as a format for data interchange. Today, the Borland/Visigenic ORB is bundled with Netscape Communicator.

For mobile clients, Oracle offers Oracle Lite 3.0, which provides a Java-enabled object database that can be used to cache a user's data. Oracle Lite 3.0 runs in only 350KB of memory and is being ported to several handheld devices. If you can spare another 300KB, Oracle Lite can provide you with a full SQL parser and bidirectional replication with an Oracle8 Enterprise Server.

Oracle also has a 2-tier solution for Java applets that need to connect to the Oracle8 database. The ThinNet8 driver is an implementation of the Java Database Connectivity (JDBC) API that emulates portions of Oracle's Net8 protocol, written entirely in Java. This enables downloaded applets to connect to Oracle8, with no requirement that software be installed on the client.

To improve memory utilization and response time of the server, ThinNet8 applets can connect to the Oracle8 Connection Manager instead of connecting directly to the database. As shown in Figure 3.12, the Oracle8 Connection Manager can multiplex the incoming user connections to the Oracle8 server.

FIGURE 3.12.
Java applets being multiplexed to Oracle8.



Application Server

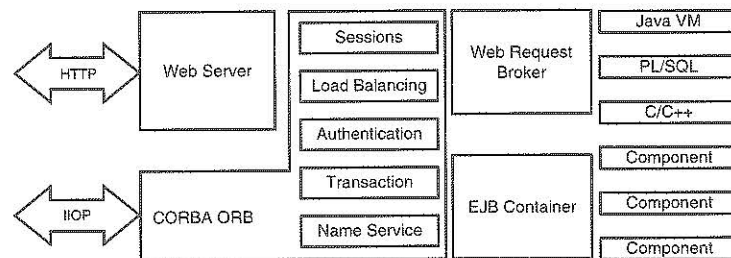
The role of the Web/Application Server tier is to run application code and abstract common services. The ideal application server would run application components fast and support lots of users in a safe, secure, and manageable way.

The Oracle Application Server (OAS) is actually two servers in one.

As a Web server, (as shown in the top half of Figure 3.13) it provides all the normal functions, including virtual path management, CGI, and realm-based security. But it goes beyond traditional servers by providing an ORB-based architecture for running Web

programs. Any URL that does not request a static page or CGI program is directed to the Web Request Broker (WRB). The WRB manages the startup and shutdown of cartridges that run Web programs written in Java, PL/SQL, C, or Perl. The PL/SQL cartridge is particularly good at dynamically creating Web pages based on data in the database and is very effective for creating simple Web-based applications.

FIGURE 3.13.
Oracle application server.



As an application server, (shown in the bottom half of Figure 3.13) OAS provides an industry-standard container for Enterprise JavaBeans and provides access to high-quality implementations of common services such as transactions, naming, security, and so forth.

Because many developers are not yet experts in CORBA and component models such as Enterprise JavaBeans, Oracle has extended the cartridge concept to include support for Java-CORBA server components. Oracle provides tools that take an ordinary Java class, package it into a “cartridge” so that it can be managed (instantiated, load balanced, and so forth), and install it into the application server. This makes it really easy to create new server-side components. Client applications access the components as if they were local objects.

All the various internal processes of the OAS (such as the resource manager, dispatcher, virtual path manager, logger, and so forth) communicate using CORBA. This means that they can be placed on separate physical machines if needed to increase capacity of the server. In this manner it is possible to create “cartridge farms,” a machine dedicated to running a particular set of application objects.

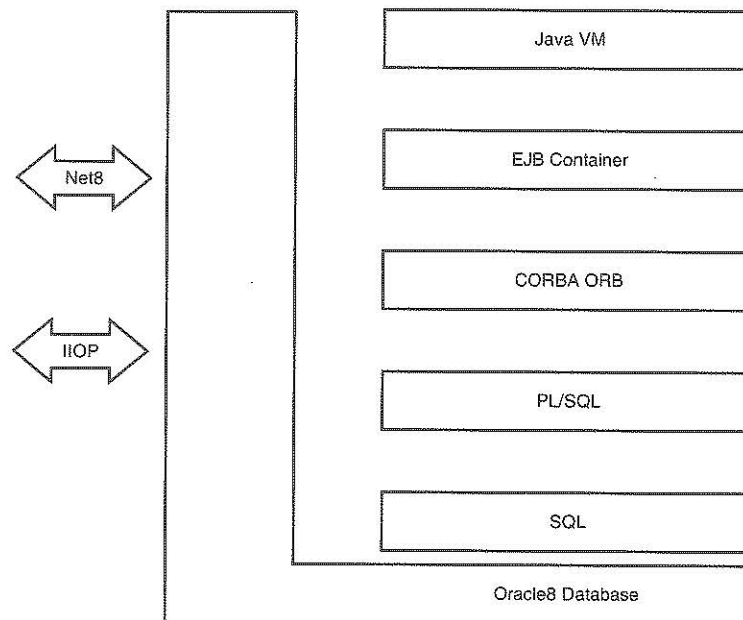
Database Server

The role of the Database Server is to provide very fast access to large amounts of data, respond to huge numbers of concurrent requests, never lose anything, keep it secure, and provide powerful development and administrative facilities.

The Oracle8 Server supports access through Oracle’s Net8 protocol. It runs the PL/SQL code used by the PL/SQL cartridge and supports data access via SQL.

As of release 8.1, the server also provides a Java Virtual Machine, an EJB execution environment, and a CORBA ORB in the kernel (see Figure 3.14).

FIGURE 3.14.
*Services available
in the database
kernel as of
release 8.1.*



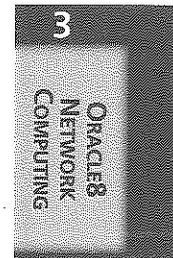
These new services provide many important benefits:

- Stored procedures, triggers, and other application logic can be coded using industry-standard Java in addition to Oracle's PL/SQL. Java classes will be able to call PL/SQL procedures and vice versa.
- The database will directly host EJB components, which will be ideal for Entity beans that maintain persistent state in the database.
- CORBA clients will be able to communicate with the database via IIOP in order to invoke stored procedures or EJB components.

These new Oracle8 capabilities will provide the developer with exciting new architectural options for implementing 2-, 3- and N-tier network computing systems.

The Network Computer

Less than half of our computer operators need personal productivity software in order to carry out their primary work processes. Studies have shown that approximately 40 percent of all installed networked PCs are primarily used as display terminals. The costs of



learning, configuring, and maintaining a PC are a nonproductive use of the operator's time and have become a nonproductive cost of doing business for the Enterprise. Beyond the constant fooling with hardware, installs, and backups, the PC is a single point of failure that can stop a knowledge workers' production completely.

And the hardware keeps becoming more complex. PC vendors are packing more and more features such as computer-telephony integration, voice control, and video conferencing to an increasingly overburdened architecture. PC ownership costs continue to rise, even as the initial cost of the hardware continues to plummet.

With the adoption of network computing technology, fewer people will need the power of a desktop machine and will instead opt for smaller, mobile, multifunction devices. The concept of the PC ball and chain will go away as vendors blur the lines between consumer electronics, communications devices, and Personal Digital Assistants (PDAs). These devices will need to be based on a more reliable architecture—people don't want to have to reboot their cell phones.

A good analogy of the potential for Network Computers is the telephone system. Telephones are relatively cheap and simple devices that plug into a vast and complicated network of switches, repeaters, satellites, and who knows what else. The telephone system is managed by professionals, and it is very reliable. I can go down to any local department store, pay about ten dollars for a simple telephone, take it home, and plug it in. It just works. I can call anywhere in the world and use my voice, send data, or send a fax.

A Typical NC

Deployment of Network Computers is one way to reap the benefits of implementing an architecture based on network computing.

A Network Computer typically contains no hard disk, and boots its operating system from a file server via NFS. Once booted, the NC prompts for login information and then displays the user's desktop, as configured by the system administrator.

Typical applications available on the desktop include

- A Java-enabled Web browser
- An email client and news reader
- A contact list, calendar, and task list
- A simple word processor and/or an HTML editor
- Telnet and 3270 emulation
- A file manager for organizing a user's personal files

The NC Reference Specification

The Network Computer Reference Profile was originally released by Oracle, Sun, IBM, Netscape, and Apple in July 1996 and transferred over to an independent standards body, The Open Group, in June 1997 as X/Open Document Number X975.

The specification is hardware-platform-independent, but includes a few minimum requirements, such as a pointing device, input device, sound, and so forth.

The latest version of the specification also includes the following software standards:

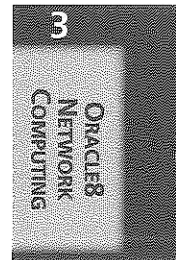
- Web support, including HTTP, HTML 4.0, JavaScript, and the following file formats: GIF, JPEG, AU, WAV, and Adobe Portable Document Format (PDF)
- Support for the Java Developer Kit 1.1
- A CORBA ORB with Java bindings
- Support for the Network File System (NFS)
- Network support protocols, including the Simple Network Management Protocol (SNMP) for network management and Dynamic Host Configuration Protocol (DHCP) for automatic network configuration
- VT100 Telnet and TN 3270 for connection to IBM Hosts
- Electronic mail protocols, such as the Simple Mail Transport Protocol (SMTP), Post Office Protocol (POP) version 3, and the Internet Message Access Protocol (IMAP) version 4

Open Issues

The term Network Computer has been coopted by some of the terminal vendors to include devices that act as dumb terminals for 3270, UNIX, or Windows hosts. Although these devices have been selling well as terminal replacements, users are not likely to give up their PCs until we have full-fledged Java-based NCs running compelling distributed application suites.

Unfortunately, the deployment of Java-based NCs has been quite slow as vendors struggle to learn the ins and outs of the new technologies and architectures. Currently, each of the available systems requires a different type of boot server—a situation that the major vendors have promised to resolve.

However, now that Sun is shipping its JavaStation and Lotus has released its eSuite productivity suite written in Java, we can expect other vendors to quickly follow suit.



Summary

In addition to NCA, Oracle is working with the other major players who are behind non-proprietary, network-centric computing, such as IBM, Sun, and Netscape, to create open standards for communications, applications components, and metadata. Creating a standard information infrastructure will make it easy to have a large number of choices in components that can all interoperate—just like buying plumbing fixtures at the local hardware store.

Also, Oracle is committed to providing enterprise class implementations of these emerging technologies. For example, in Oracle 8.1, Oracle intends to provide a relational database server that runs Java within the kernel and provides a container for running Enterprise JavaBeans. It also intends to provide access to all server functionality via CORBA.

The end result of implementing a network computing architecture might be that a company could deploy low cost, easy-to-use, and highly manageable Network Computers.

Oracle8 incorporates many of the features of TP monitors, message-oriented middleware, queuing systems, DataWarehouse engines and now application servers, ORBs and transactions servers. To use Oracle8 only as a repository for data is to waste a powerful resource for your enterprise.

8

CHAPTER

Coexisting with Non-Oracle Databases

*by Advanced Information Systems,
Inc.*

IN THIS CHAPTER

- Sleeping with the Enemy 146
- The Gateway to Everywhere 149
- Coexistence and Replication 164

Sleeping with the Enemy

Although any database vendor will tell you that his database should be running 100 percent of your systems, this isn't usually the case. In the first place, converting all a company's systems to one database at once will put too much strain on the existing resources. Secondly, it might not be logical to move all the systems to one database. Different applications demand different types of database software. This is why in most large or medium corporate operations, you need to connect disparate database software.

Because of this phenomenon, you might be in a situation where you purchased an Oracle RDBMS, and you have a number of separate systems that need to be interfaced to Oracle, systems that have major differences in their environment from yours. Here are some of the differences:

- Different operating systems
- Different hardware vendor
- Different RDBMS vendor
- Different paradigms

A project team and an RDBMS software application might need to contend with these four areas when dealing with outside systems.

Different Operating Systems

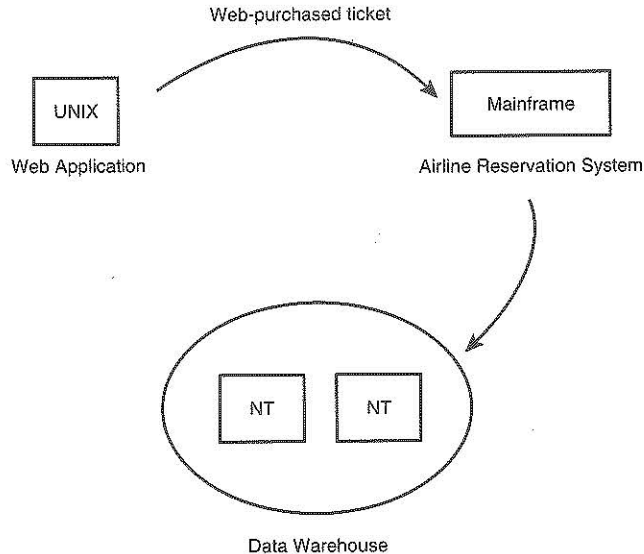
This is a more difficult obstacle if these two systems do not talk to each other for any other reason. In this case, linking a system with a different OS than your Oracle database means that your project needs to establish the links from the lowest networking level to the level of screen interfaces. Different operating systems can have many subtle differences.

Picture a fictitious airline that purchased Oracle as its Web server. Soon customers are ordering flights over the Web and querying flight information. This information processing has been traditionally performed on a huge mainframe, but now this new Oracle database needs to connect to a different mainframe. Say that the database is running UNIX. This is a classic example of two very different operating systems that many times need to talk to each other.

The situation can be compounded with three operating systems that need to communicate database application logic. If this fictitious airline purchased a huge Windows NT cluster for a data warehouse, you would need to send data directly to Windows NT from both the UNIX and MVS operating systems as shown in Figure 8.1.

FIGURE 8.1.

A corporate environment usually has many generations of different hardware and operating systems.



Connecting three databases over different operating systems is not inherently more difficult, but managing the underlying connection between the OSs and any data translation is.

Different Hardware Vendors

The complexity of a database gateway over different hardware systems isn't drastic if you run the same operating system. If you run the same operating system and the same database, the differences are minor. If you are running different RDBMS products by different vendors, the complexities increase.

Looking at the Microsoft NT operating system, differences in hardware evaporate when the Windows NT services are in place because Windows NT is the product of one vendor. UNIX, however, differs more between hardware vendors; therefore, linking a SUN SPARC to an HP-9000 does involve the consideration of minor operating system differences or kernel configuration differences.

Different RDBMS Vendors

Even when you solve any lower-level adjustments that you must make to realize communication between two databases on two separate machines, you still might be running two different RDBMSs. For instance, if you run Oracle on one Windows NT machine and SQL Server on another Windows NT machine, you need a translation process when



sending transactions from one machine to another. Most of gateway theory as related to this chapter concentrates on the similarities and differences between databases and common ways they connect with one another.

Different Paradigms

The first three differences listed between an Oracle RDBMS and another system were physical and objective in nature. Yet this final difference refers to the paradigms of the different human groups that manage applications and must talk to one another. The way that hardware architecture, a database, or an application are built relate very closely to this paradigm.

For instance, if your system was built using an object-oriented methodology, yet the system that you must interface with was not, you need more components for a successful interface. Furthermore, the most efficient solution might not be to force another system to think like you do but instead learn to think like the other system when communicating with it.

You see this paradigm split most commonly between mainframe systems and more *open* systems. Both systems have been successful doing certain tasks but need to talk to each other. In this case, if you use Oracle, you are probably on the “open” side wondering how you are going to communicate with a mainframe. Suddenly, your “open” system doesn’t appear very open when it needs to interface with a legacy mainframe.

If the paradigm of the people who manage the mainframe is similar to yours, in that they might also use another relational database such as DB2, you can communicate and build bridges rather quickly. If they come from a completely different paradigm of VSAM files or IMS, it will be harder to map their data to yours because both data sets are structured from different paradigms.

It is important for an Oracle project to have flexible methods for propagating data to other systems that add value to the Oracle system. Many times, your Oracle system will have to conform to the data structure of the system to which you need to interface. This will probably involve many table joins and other complex SQL to “flatten” and denormalize your data for a mainframe dataset.

Your system interface will need to communicate to a legacy system from which it is very different. But this foreign system also needs to talk to your system. Both systems need to understand the other’s protocol to communicate through a network. At a higher level, both databases on these two systems need to talk to each other and understand a part of the other’s language.

Instead of building this interface from scratch, you can purchase products that connect networks, isolating you from the problems of protocol differences. The Oracle network offering that isolates you from different protocols is SQL*Net. Likewise, you do not need to build the database interface; you can purchase one. This interface between two different databases is a *gateway*.

The Gateway to Everywhere

When you enter the world of gateways, you soon realize from all the vendors selling them that your database can talk to just about anything from a mainframe to a mountain goat. Therefore, you must determine which gateway you need and which gateway will be most cost-effective given your budget and resources.

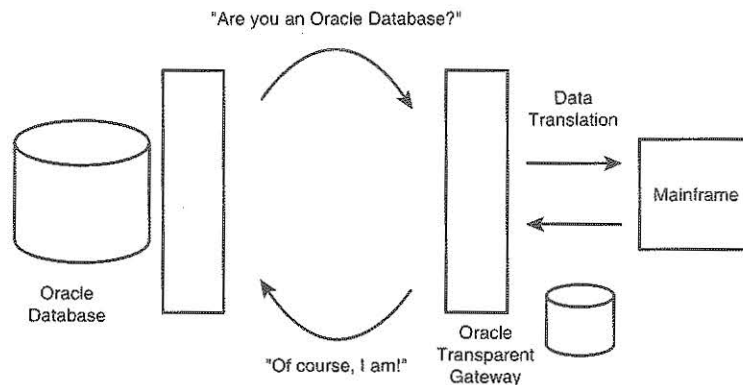
What Kind of Gateway Do You Need?

Gateways can be broken up into three major areas:

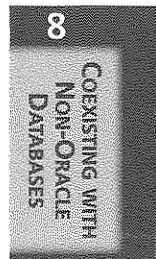
- **Transparent Gateways**—These are the top-of-the-line gateways where the interface is so perfect that your Oracle systems believe that they are talking to other Oracle instances, when they are communicating with different databases on different platforms: The gateway “tricks” the Oracle instance to think that another remote foreign database is actually an Oracle database, as in Figure 8.2.

FIGURE 8.2.

Oracle's Transparent Gateway appears as an Oracle database to the other Oracle databases it is linked to.



- **Procedural Gateways**—At this level, the Oracle database does not believe that it is talking to another Oracle database, but instead the database can communicate to a set of libraries written in either C or PL/SQL. These libraries provide a set of calls that need to be used explicitly when linking any portion of one database's activity to another.



- **Passive Gateways**—Most of these gateways are homegrown and involve moving data through a series of steps, first moving data to another machine, and then at a point in time having the other system read in the data. The gateway is passive because there is no direct communication between database systems. Data is simply being sent from one database to the other by an independent set of processes.

Now consider how these three forms coexist with your Oracle databases and other “foreign” databases. As you can probably guess, as you move from Passive Gateways up again to Transparent Gateways, the cost of them becomes greater as functionality and ease-of-use increase. You incur a greater cost for a more sophisticated gateway, so unless that sophistication adds value, the most profitable solution might be a less expensive gateway.

Transparent Gateways

As defined in the previous section, a Transparent Gateway “tricks” the Oracle instance into thinking that an Oracle machine exists remotely when, in fact, it is a completely different database offering. Here are some of the major databases where you can purchase this type of interface to Oracle:

- Microsoft SQL Server (offered free sometimes)
- Informix
- Ingres
- Sybase
- IBM DB2
- EDA SQL

Levels of Transparency Exhibited by the Oracle Transparent Gateways

These Transparent Gateways are indeed transparent to the Oracle RDBMS, and they can act as just another database. Oracle’s offerings for these gateway products are called the Oracle Transparent Gateways. These software products enable a foreign database to appear as an Oracle database and make the interface transparent to the users.

Access Method Transparency

This means that just as Oracle access is SQL-based, a user can use SQL on the gateway to retrieve data from a non-Oracle database using SQL just as if it were an Oracle database. The only catch with this is performance. Regardless of the gateway’s speed, if data on the remote machine is not indexed or sorted, the same SQL runs on the remote machine, and the local Oracle machine might vary a great deal in the time that it takes to retrieve the data.

Location Transparency

This simply is an extension of Access Method Transparency in that not even the physical location of the data needs to be known by the user. Oracle's gateway will map a request to the different physical machines and databases on which the data resides.

One note here: To realize Location Transparency, an administrator needs to set up SQL*Net on both machines and create the proper database links, grants, and synonyms.

Network Transparency

This really should be called SQL*Net Transparency. You still need Oracle's standard network layer running over all your different networks to shelter each database from network differences and to provide a common protocol. SQL*Net is the portion of the gateway that handles network communication. When SQL*Net is working, there is network transparency between databases.

Operating System Transparency

This means that for database functions, users do not have to be concerned with which operating system the two different databases reside on.

Benefits of Oracle's Transparent Gateways

As mentioned earlier, the Oracle Transparent Gateway is a top-line solution. It makes coexistence between an Oracle database and a non-Oracle database easier to manage. The potential buyer of these expensive systems should first ponder the benefits of this solution to determine whether the extra cost spent will add enough value to make the gateway profitable.

Full SQL Support

Advanced Oracle extensions to ANSI SQL such as the Outer-Join operator or the Decode function might be used to access a remote database that does not even use SQL.

Full PL/SQL Support and Trigger Support

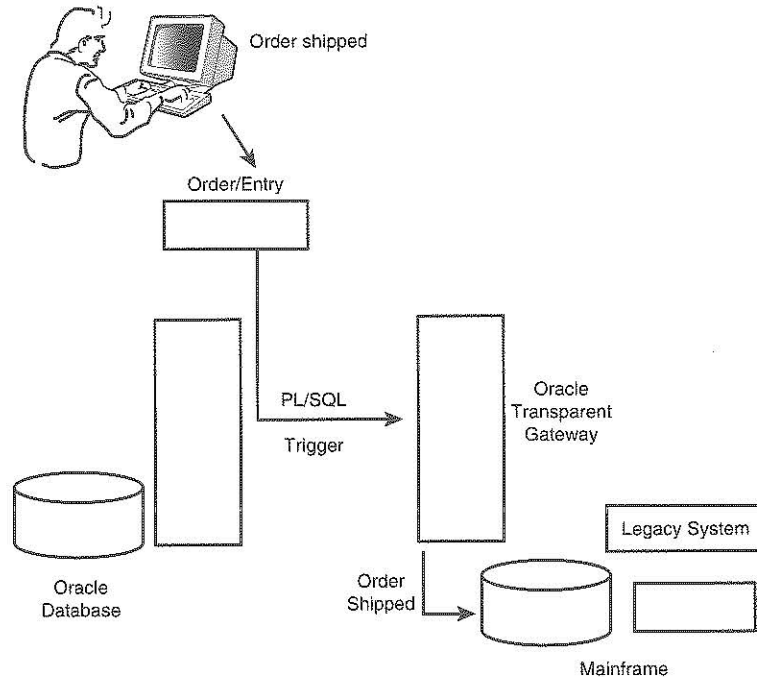
This is an important feature, in that you can now code referential integrity triggers that can create streams of business logic between two remote machines, and enforce table uniformity between two databases.

For instance, as in Figure 8.3, you can have an Order/Entry system residing on Oracle. Whenever an order is shipped, a trigger can fire out to the remote table on the other database and subtract from the inventory or the warehouse from which the goods will be shipped.



FIGURE 8.3.

With Oracle's Transparent Gateway, developers can use common tools such as PL/SQL to communicate to totally non-Oracle systems. The gateway handles the translation.



Distributed Transactions Use Two-Phase Commit

This refers to a change in data that must take place on two or more machines, as seen in Figure 8.4. Data is committed in the usual way on the first machine and then sent across the network and the gateway to the next machine. Not until the remote commits are all complete is the transaction considered final. This is useful for tables on two different databases that need to be replicated and/or need to always be synchronized.

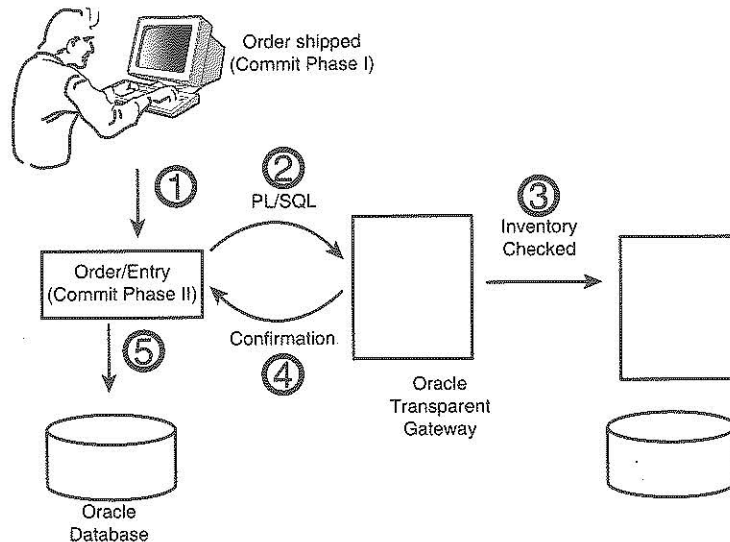
In the previous example, if you never wanted a sale to be shipped until you were sure that you subtracted it from the inventory, you would use the two-phase commit in the following way.

Cost-Based Distributed Joins

If you use cost-based optimization and have run the proper SQL to populate Oracle's internal tables, Oracle can choose a join across a distributed network in an efficient way.

FIGURE 8.4.

A simple request to ship an order results in a two-phase commit that will not be successful unless the mainframe database confirms that inventory exists for the shipment.



Transparent Schema Mapping

This refers to the gateway's capability to view a foreign database's tables and columns in the same way that Oracle tables and columns are defined. This includes mapping of dates, characters, and LONG data. This transparency can provide a seamless mapping between an Oracle character string and a DB2 character string residing on MVS. To do this, you need to account for vendor differences in data storage between DB2 and Oracle and convert to the ASCII character set from the mainframe EBCDIC character set.

Transparent Views of Data Dictionaries

According to Codd's original writings on Relational Theory, a relational data dictionary appearing as tables to the user is a vital aspect of a relational database. Oracle's gateway provides this illusion, so a remote non-Oracle schema, at least in part, appears to the user as an Oracle data dictionary.

Procedural Gateways

Procedural Gateways can be as powerful as Transparent Gateways; the only catch is that the logic linking Oracle to a non-Oracle database has to be explicitly coded. These are ideal gateways if you have, or are willing to purchase, the in-house talent to use these "procedures" to create a gateway.

What Is This Thing? You Mean I Have to Do Work?

Procedural Gateways don't make everything better right away like Transparent Gateways. Transparent Gateways sit both at your Oracle computer's site and at the remote site and translate what your database is doing to the language of the remote database. A Procedural Gateway does the same thing, but you need to invoke it from your applications manually. Think of a Transparent Gateway as a brand-new bicycle that you pick off the shelf, whereas a Procedural Gateway is a bike that you buy and need to assemble at home.

A Procedural Gateway is named such because what you are buying is a gateway for procedures. This is a gateway that enables communication through procedures that you must write to invoke a remote non-Oracle database.

The Oracle Procedural Gateway utilizes the Advanced Program to Program Communication (APPC) protocol. This means that your Oracle interface programs can speak to mainframe programs that don't use Oracle but have access to the APPC, which is a popular mainframe protocol.

With the APPC Oracle Procedural Gateway, you can communicate with the following Mainframe tools:

- CICS/MVS
- IMS/TM
- CA-IDMS
- CICS/VSE
- CICS/400
- REXX (for the REXX fans in the audience)

This means that within your CICS code, you can call these procedures to communicate with a remote non-MVS Oracle database. With the APPC, you can also communicate with the following databases and file systems:

- DB2
- IMS
- CA-IDMS
- ADABAS
- VSAM

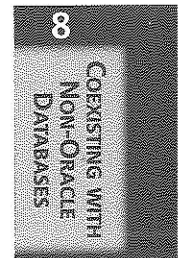
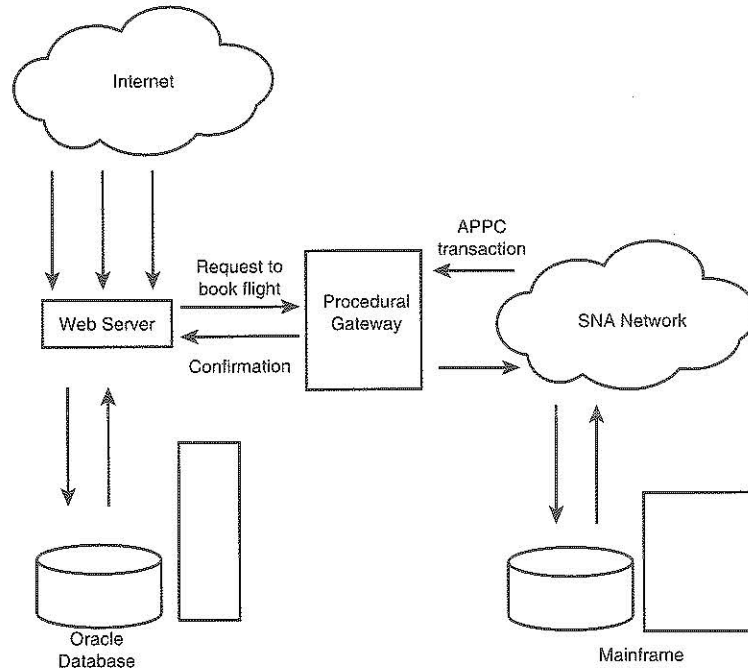
Oracle uses the X/Open Common Programming Interface for Communication. This makes the Oracle gateway very flexible because this is a common mainframe communications standard.

A Day in the Life of a Procedural Gateway

When using a Procedural Gateway, refer to the Web-based frequent-flier database (Figure 8.5). Reward your airline patron with a free flight. Assume that you cannot book flights over the Web. After you determine that your traveler has the necessary miles, call a PL/SQL procedure to book the flight on the mainframe. At this point, the procedure flies through SQL*Net to the Oracle Procedural Gateway waiting for you on the ancient mainframe. The gateway builds an APPC transaction and routes it through the SNA (mainframe) network. The response is returned via SQL*Net again to your application. This means that someone on the Web can cash in on free miles and book a flight on a mainframe. Here is how the whole “procedure” looks.

FIGURE 8.5.

With a Procedural Gateway, all you need to do is build the logic; the gateway takes care of all networking and request translations needed by the non-Oracle database and hardware environment.



IBM's MQSeries

If you use IBM's MQSeries messaging products, Oracle also has a Procedural Gateway that links a mainframe or newer IBM product using this protocol for databases on another system. Oracle's procedure library supplies you with PL/SQL when you buy this gateway, so your development staff does not need to learn a new set of APIs.

With these procedure calls, you can employ logic affecting a remote mainframe through PL/SQL, database triggers, Oracle Developer/2000, and third-party tools. This tool even enables you to build distributed databases or, at least, distributed tables because the MQSeries messaging supports two-phase commits. You can have your frequent-flier database share a table called REWARDS that exists on the mainframe and on your Oracle instance. With the MQSeries gateway and the two-phase commit, you can ensure that these tables remain in sync.

Why Purchase a Procedural Gateway?

Because you understand that Transparent Gateways ensure that your non-Oracle databases appear as Oracle databases, thus leaving you with an integrated database environment, it might be hard to understand why you might trade that power for Procedural Gateways and only the *ability* to create data routines that link your Oracle database to a remote one. Well, the simple answer is price. Procedural Gateways cost much less.

It is important when considering the cost of a gateway to factor in development time. Even if a Procedural Gateway costs less, the price of developing the routines to make it work might exceed the price of the Transparent Gateway. It is also important to realize that you will need to purchase SQL*Net that links your Oracle network to the remote network; this cost can be high if you price for a large mainframe. If you need only a few simple interfaces between your mainframe and your Oracle database, maybe both the preceding gateway solutions are not correct and all you need is a Passive Gateway (discussed in the next section). This gateway has virtually no costs outside of development; yet complex functionality such as replication, high availability, and distributed data would be tedious. That is why these higher-level gateway solutions exist—they give us solutions for complex problems.

Passive Gateways

Passive Gateways are the “Poor Man’s Gateway.” In most cases, they are simple programs written in-house to refresh portions of two different databases. However, if you do not plan your Passive Gateway project well, it can end up costing more than any other solution. Furthermore, if you need greater synchronization between databases, a Passive Gateway might just be a “redesign of the wheel” that can be avoided using one of Oracle’s gateway options. It is important to use these simple gateways only when the task is not too complex.

The Zen Gateway—Does It Really Exist?

I have termed a Passive Gateway as one where data is moved across one database to another remote database manually. This data movement can be termed as “passive”

because it does not occur within the structure of transactions but is only moved when a user-defined process is started. It is also passive because from the point of view of each database and each process, the gateway itself does not exist. The Passive Gateway comes into existence only for a brief moment in time.

A monthly data refresh between two separate computer systems is an example of a Passive Gateway. Data is simply transferred as a series of files by a user-defined set of programs. These programs move data first out of one computer's database into an operating system file and then move it into another computer's file system and database. Both databases consider these programs no different from any other database-related process that moves data from the database to a flat file.

The advantage of the Passive Gateway is that it is usually less expensive. To unload data from one database, transfer it, and then load it into another database is much simpler than the low-level translations of SQL*Net and Oracle's Transparent Gateways. You can perform this operation with standard tools that do not cost extra—no need to purchase SQL*Net with this method.

The disadvantage with Passive Gateways is that they are so simple and fit such a specified need that they usually need to be custom built. Yet if development costs are managed for a simple Passive Gateway, the cost of Oracle's SQL*Net product and of Oracle's Open Gateway products will be greater. The hours spent simply configuring SQL*Net and an Open Gateway can be higher than the development of a simple Passive Gateway.

It is important to decide which kind of gateway you need. If you want to keep replicated databases running at two sites, probably designing your own data-transfer and logging routines will be too complicated. Yet if you need data from one machine periodically sent to another, consider a Passive Gateway.

Building a Passive Gateway, Mainframe to Oracle Example

Look at your fictitious airline business (Figure 8.6). Assume that there is only a frequent-flier Oracle database that is running over the Web; it cannot book flights. The database to reserve and purchase flights exists separately on a mainframe.

The frequent-flier application is running on a UNIX server that is running the Oracle8 database and the Web server. The main database of airline purchases exists on a mainframe running DB2. The only reason that these applications need to talk to each other is because flight data must be moved monthly from the mainframe database and used to update the Oracle/UNIX frequent-flier database.

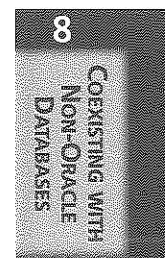
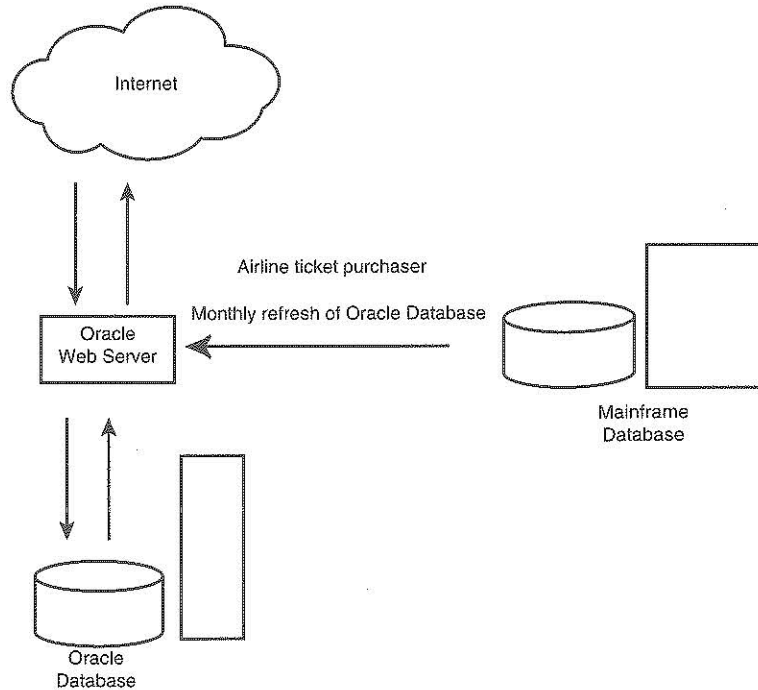


FIGURE 8.6.

With a transaction such as a monthly refresh that does not need to occur for each transaction, many times a Passive Gateway can handle this simpler relation.



Because data is moving from the mainframe to the UNIX environment, you first must extract the information from DB2 and convert it to a flat file. The query might be something such as this:

```

EXEC SQL
Select
Customer_name, sum(miles_traveled)
From flight_activity
Where month = :current_month
Group By Customer_name
END EXEC
  
```

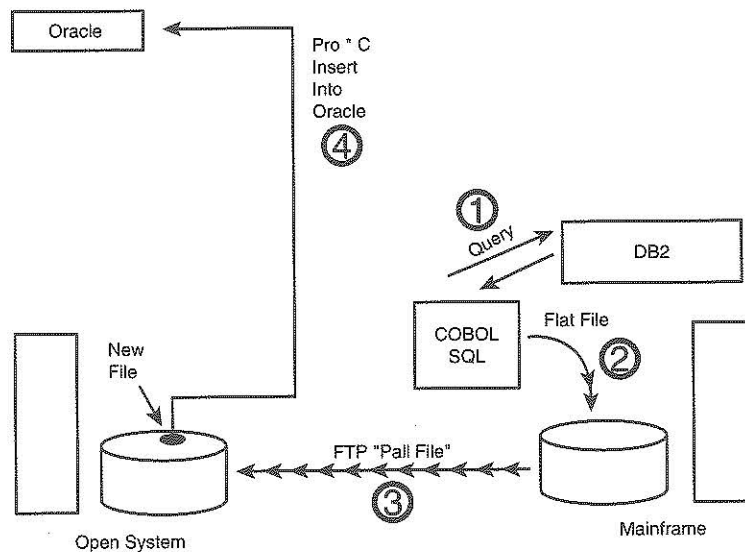
This query can be written with SQL embedded in COBOL that writes to a file (dataset) on the mainframe with the results of the query as in Figure 8.7. You can also use DB2's SPUIFI tool instead to execute this query and spool data to a dataset.

You would create a JCL script that would run once a month to execute this COBOL job, on the night of the last day of the month. The results of the job would sit in a dataset called monthact.

That same evening on the Oracle UNIX box, there would be a cron job command that might run every 15 minutes to ping the network and see if the MVS dataset existed and was available. After the data finished loading, a UNIX command from the Oracle box would pull the dataset through FTP with the Get command and move the information to a file on the UNIX file system.

FIGURE 8.7.

With a Passive Gateway for a periodic refresh, you see many different user-created programs working together yet not communicating with one another.



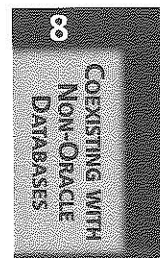
After this UNIX *cron* job finished and the UNIX file existed with the month of new flight activity, the *cron* script would call a PRO*C program. This program would read in the rows of the data that were extracted from the DB2 Select statement:

Our data:

```

1: Ernie Salesnutt      34000
2: Bill Gateway        12000
3: Garuda Deli         10000
4: GetMainframeData.pc - Program Shell:
5: EXEC SQL BEGIN DECLARE SECTION;
6: VARCHAR    customer_name[48];
7: int        miles;
8: EXEC SQL END DECLARE SECTION;
9:
10: EXEC SQL INCLUDE SQLCA;
11:
12: EXEC SQL WHENEVER SQLERROR GOTO err;
13:
14: FILE *fp_data, *fp_log;
15:

```



```

16: main()
17: {
18:     init();
19:
20:     open_file_from_mainframe(fp_data, "monthact.dat");
21:     open_log(fp_log, "monthact.log");
22:
23:     declare_cursors();
24:     while ( read_a_row() == 0)
25:     {
26:         update_frequent_fliers();
27:         add_to_log();
28:     }
29:     close_cursors();
30:
31:     close_files();
32:     clean_up();
33: }

```

For each row, the program would perform an update of the UNIX Oracle frequent-flier database and the table `Customer_Miles`. The C program would issue the following SQL:

```

EXEC SQL
Update Customer_Miles
Set total_miles = total_miles + :miles_just_read_in
Where customer_name = :name_I_just_read_in;

```

Notice that you also are writing to a log file. This is important for a homespun gateway, to keep track of every step of the process of data movement from the mainframe to the Oracle instance.

In a more realistic case, of course, you would just use the frequent-flier number of the customer and not a nonunique name. But using the name illustrates that you are updating a customer's miles based on new travel for the month. Leave it here for teaching purposes only.

Instead of a direct update from a file, you can simply load the DB2 file directly into an intermediate Oracle table called `monthly_upload`. Use a `SQL*Loader` script (Oracle Loader) to accomplish this. After this is tested, you can write a trigger to automatically update the `customer_miles` table whenever data is inserted into the `monthly_upload` table. This table is your record of the previous month's transfer. If late transactions need to be loaded later, they can be loaded without affecting your Passive Gateway. This would be the trigger used to update your `customer_miles` table:

```

1: Create or replace trigger customer_miles_ins
2: Before insert
3: on MYDATABASE.Monthly_load
4: For each row

```

```

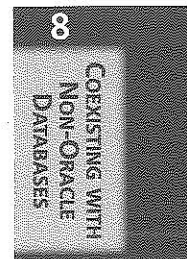
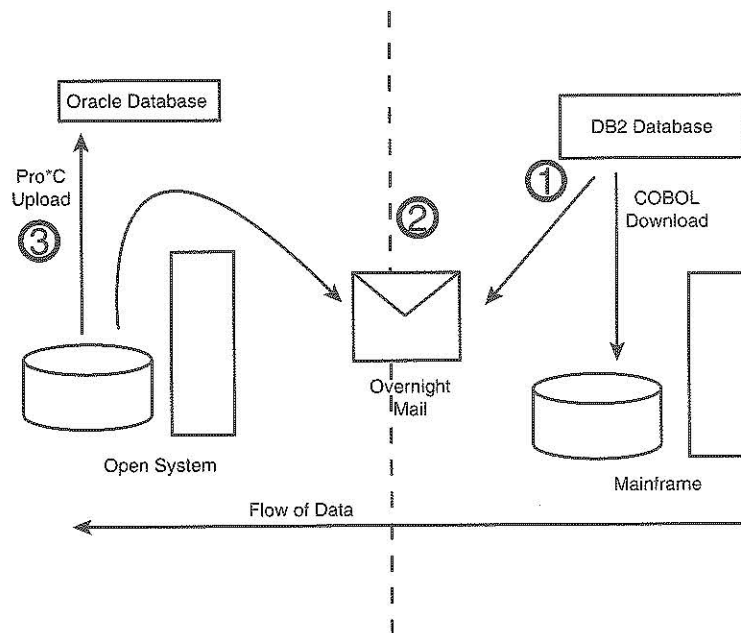
5: Begin
6:   Update customer_miles
7:   Set total_miles = total_miles + :new.miles_loaded
8:   Where customer_name = :new.customer_name;
9:
10: EXCEPTION
11:   WHEN others THEN
12:   Raise application_error(-20000,'update of ' || :new.customer_name ||
   ↳ for ' || :new.miles_loaded || ' failed. ');
13:
14: End;
15: /

```

In a production system, your error-handling would probably be more exact. Yet with these simple SQL scripts on the mainframe and the UNIX machine, you can create a Passive Gateway that both databases are unaware of, yet is vital in keeping both systems synchronized.

Of course, we are using the network software to convert the character sets; furthermore, we are planning our DB2 COBOL download so that problems such as “packed decimal” numbers won’t need to be translated on the Oracle end of the process. We can even eliminate the network and have a *very* Passive Gateway where each month a tape would be physically mailed to the Oracle site and read from tape by a PRO*C process (see Figure 8.8).

FIGURE 8.8.
Mainframe data transferred into the Oracle RDBMS via overnight mail due to unacceptable transfer rates from the mainframe.

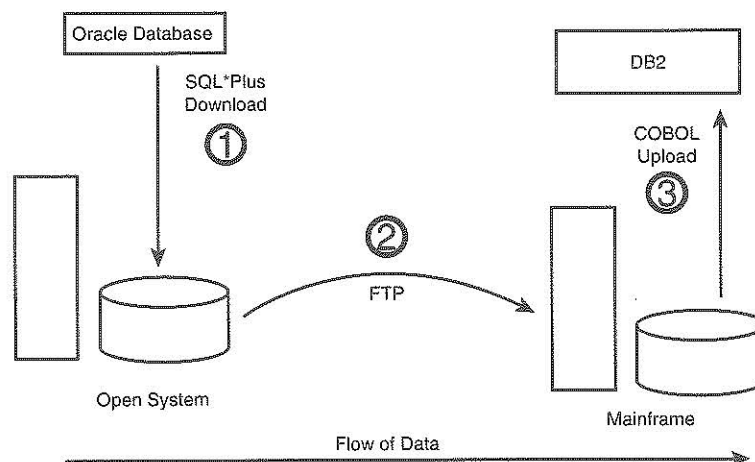


Building a Passive Gateway, Oracle to Mainframe Example

In today's computing world, a gateway from an Oracle application on an *Open System* back to a mainframe is more common. Many times, corporations have slowly moved applications off of mainframes in their *downsizing efforts*, yet need these new Oracle applications to communicate back to the mainframe and the other applications that have not been downsized.

Again assume we have a frequent-flier database over the Web, but we need to send reward flights for our best customers into the accounting system every month end. Our accounting system is running on an ancient system using VSAM files and is remote from us on a distant mainframe. In this case, the data needs to move from the Oracle RDBMS to the mainframe (Figure 8.9).

FIGURE 8.9.
*SQL*Plus extract of data for an FTP cooperative processing upload back into a mainframe.*



In this scenario, you need to convert Oracle data into a flat file and then move this file to MVS so a process can read it into a VSAM file format. To move data out of Oracle and into a flat file, use the trusted product SQL*Plus. SQL*Plus enables you to spool any SQL statement with formatting options. You need formatting options to create a fixed-length file that is easy for your mainframe friends to read.

In this example, you download data from the Oracle server on reward flights that have been awarded to customers in a given month. Each month, you will refresh the mainframe with this data:

You need to join two Oracle tables, `Customer_Miles` and `Customer_Flight`. Here's the SQL*Plus statement with the formatting commands necessary to create a fixed-length data file with this information.

```

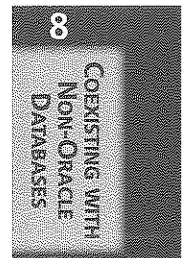
1: set pagesize 0
2: set linesize 80
3:
4: set term off
5: set feed off
6: set echo off
7: set show off
8: set veri off
9: set head off
10:
11: col customer_id      format 9999999999
12: col reward_miles     format 999,990.00
13: col depart_date      format a10
14: col return_date      format a10
15: col flight_no        format 99999
16:
17:
18: spool outputfile
19:
20: Select
21: cm.customer_id,
22: cm.reward_miles,
23: to_char(cf.depart_date, 'mm/dd/yyyy'),
24: to_char(cf.return_date, 'mm/dd/yyyy'),
25: cf.flight_no
26: from
27: Customer_Miles cm,
28: Customer_Flight cf
29: where cm.customer_id = cf.customer_id
30: and   to_char(cf.depart_date, 'MM/YYYY') = '&1'
31: /
32:
33: spool off
34:
35: exit
36: /

```

This SQL will be executed by a *cron file* at the end of each month. The constant '&&1' accepts the first parameter that, in this case, will be the two-digit month code and the year. When the script fires for December 1999, this six-digit code will be 121999. No year 2K problem here!

After this script creates the output file *outputfile*, the cron job fires again and calls an FTP process using the *put* command to move the output file to the mainframe.

At this point, the spool file can be loaded into a VSAM file and the mainframe accounting package can access the new data.



One important note here: We performed a join in our SQL because we are operating under the relational paradigm of normalized data. In most cases, nonrelational data appears “flatter” and might necessitate many joins of our tables. This is a common phenomenon when moving data from a relational database to an older mainframe database that is nonrelational.

By writing a few simple programs, we have bypassed the need to use expensive networking and gateway products. We can add more complexity, such as error and status logs, that would be passed along with the data between our “open” Oracle system and the legacy mainframe with which we want to coexist.

Coexistence and Replication

The purest form of database coexistence is a distributed database. A distributed database appears as one database to users and developers, yet is a series of separate databases at the physical level. To accomplish this task, we need to replicate data on these different databases so that at the higher levels data appears uniform regardless of which system we query or modify.

Replicating What Is Different

We have talked in this book about replication between two separate Oracle databases. Now we must tackle the task of replication among databases that might be totally foreign to Oracle in every way. The ideal form, where a large network of connected databases act as one database, replicating one another’s data, is a *distributed database*.

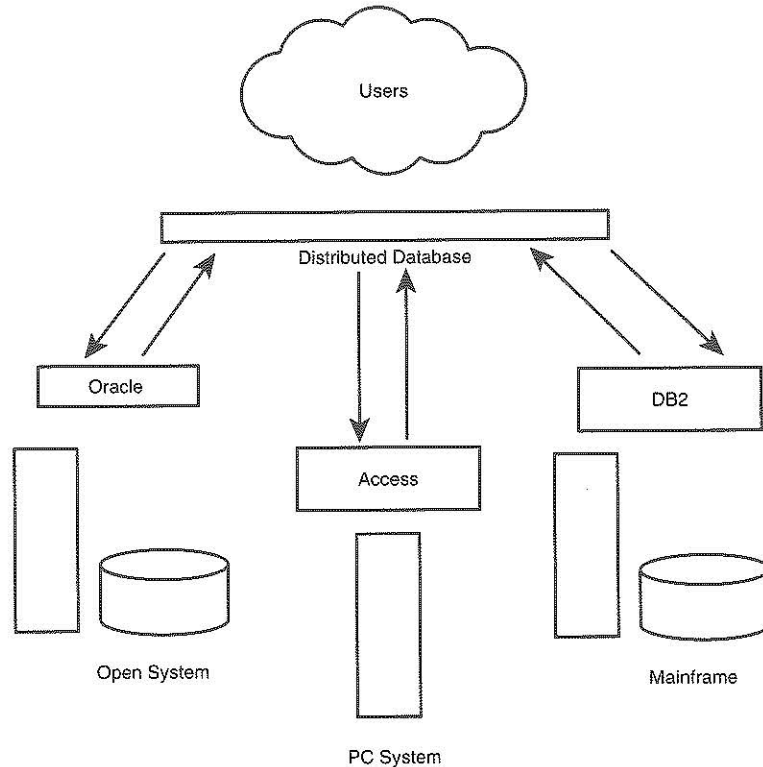
A distributed database is singular in that although data physically exists at many sites, it is still one database at the highest level that is simply distributed physically, as in Figure 8.10.

Oracle solves the mystery of the distributed database in a twofold way. Oracle first offers a Transparent Gateway to the foreign machines, so they all appear to be Oracle databases. This also involves the purchase of SQL*Net to make different network protocols appear the same. After these tasks are accomplished, proceed with replication in the same way as if the databases were all Oracle databases.

If instead you only need consistency, at day’s end, you don’t need to constantly refresh transactions but can *defer* them for propagation to other nodes of your distributed database. To do this, you need to store logs of your transactions and then at a given time use the log information to refresh remote nodes. This type of replication is *Asynchronous*.

FIGURE 8.10.

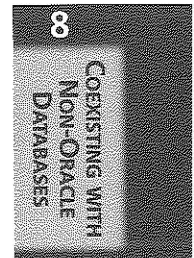
A distributed database appears to users as one database.



Obviously, Asynchronous replication is less sophisticated in that each transaction does not need to be propagated to other nodes of the distributed database but instead to a log file. Because of this, a two-phase commit will not need to exist for a transaction to complete. This makes for a faster transaction rate.

With Asynchronous replication, we do have the added worry of our transaction logs. Even though they fill up only with rowids, timestamps, and other logging information, after many transactions, log files can become large. Therefore, Asynchronous replication can take up more space. But today, with speed more valuable than space, this is not a major concern. A megabyte of storage can be purchased for a rupee.

One catch with Oracle's replication, even if you have a Transparent Gateway, is that you need the IBM Data Propagator to move data from the mainframe database back to Oracle, either synchronously or asynchronously. This is vital in configurations where a mainframe is still the central computer of an operation with new Oracle departmental servers existing to perform local processing.



Features of Oracle Replication Services

After you purchase a Transparent Gateway and the Replication Services software from Oracle, you gain many advantages in replicating Oracle data with data from a remote database. In the most common case, these remote databases will be mainframe databases. Here are some of the features of Oracle's Replication Services:

- On the Oracle end of things, if you use Windows NT or UNIX, your replication efforts are coordinated by a graphical front-end. This is far easier to use than a line-mode interface that forces you to execute commands. This includes automatic error-recovery.
- Column and Row subsetting is available. This is very important when interfacing a mainframe environment to an open system. Many times, mainframe columns are not as normalized and carry old-style keys. For instance, you might have a column on the mainframe called `flight_code`, which is 17 characters representing three different codes:

Position 1 - 2	Airline Code
Position 3 - 10	Coupon Number
Position 11 - 17	Reservation Code

Using Oracle's Replication Services for a distributed environment lets you map this one column on a remote mainframe to three columns in the Oracle database.

- Oracle's Replication for a distributed environment also enables you to write code to convert one type of data in the mainframe and map it to data in the Oracle database. For example, your mainframe might be in London using European dates, yet your Oracle Server in New York needs to read in these dates and convert them. You can code functions that perform complex translations on columns.

Summary

Most people don't need Transparent Gateways between systems; instead they will build a Passive Gateway between systems. It is a minority of Oracle shops that need a Transparent Gateway to turn their many Oracle and non-Oracle databases into one large distributed database.

One reason for this is price. If the Transparent Gateway were free, believe me, the majority of Oracle shops would use it; but a Transparent Gateway is instead more expensive. The Transparent Gateway is a high-end item that is very profitable for Oracle to maintain even if only a small portion of Oracle shops purchase it.

Another reason why the Passive Gateway is more popular is simply that the movement to downsize from the mainframe to an open system is more popular than the movement to hook up existing open systems to mainframes. In most downsizing, we don't strive to create one virtual distributed database with a mainframe; we instead seek to move applications off of the mainframe and eventually pull the plug on the old steel itself.

The issue of downsizing does fuel the need for coexistence between Oracle and mainframes when we realize that downsizing a system in phases leaves us with dependencies between parts of an IS Operation that have been migrated to an open system and those parts that await migration and still run on the mainframe. When these interdependencies exist, there is a need to share data between Oracle and a legacy system that, in turn, implies some sort of explicit (Transparent, Procedural) or implicit gateway (Passive) to bridge the two systems.

It's not wise to plan a brand-new distributed database with anything but one database software product running on separate nodes that all run the same operating system. Altering databases and/or operating systems simply slows down the distributed database. This slowdown occurs at all phases from design and planning to actual transaction processing. When we consider mixing mainframes and open systems, this slowdown can be due to an added layer of physical, software, and planning complexity needed to link the two systems.

In reality, most downsizings are planned to minimize dependency between open systems and mainframe components of an IS effort. The planning is such because a greater dependency between systems implies higher software costs. In effect, passive low-maintenance gateways are more popular than Transparent Gateways, replication, and distributed architecture and will remain so until the price of these glamorous and admittedly complex products goes down.

If Oracle gave away its gateway and replication products for free (or almost free), they would lose a great deal of revenue in the short term but would encourage more mainframe dependence with the Oracle database and open systems. It remains to be seen whether Oracle will take this strategic path in the future.

