

Gavin Powell



Oracle®

Performance

Tuning for

10^gR2

Second Edition

D
A
T
A

M
A
N
A
G
E
M
E
N
T

**Oracle® Performance
Tuning for 10gR2
Second Edition**

Oracle Database Related Book Titles:

Oracle 9iR2 Data Warehousing, Hobbs, et al,
ISBN: 1-55558-287-7, 2004

Oracle 10g Data Warehousing, Hobbs, et al,
ISBN 1-55558-322-9, 2004

Oracle High Performance Tuning for 9i and 10g, Gavin Powell,
ISBN: 1-55558-305-9, 2004

Oracle SQL Jumpstart with Examples, Gavin Powell,
ISBN: 1-55558-323-7, 2005

Implementing Database Security and Auditing, Ben Natan,
ISBN 1-55558-334-2, 2005

Oracle Real Applications Clusters, Murali Vallath,
ISBN: 1-55558-288-5, 2004

Oracle 10g RAC Grid, Services & Clustering, Murali Vallath,
ISBN 1-55558-321-0, 2006

Oracle Database Programming Using Java and Web Services, Kuassi Mensah
ISBN 1-55558-329-6, 2006

For more information or to order these and other Digital Press titles, please visit our website at www.books.elsevier.com/digitalpress!

At www.books.elsevier.com/digitalpress you can:

- Join the Digital Press Email Service and have news about our books delivered right to your desktop
 - Read the latest news on titles
 - Sample chapters on featured titles for free
 - Question our expert authors and editors
- Download free software to accompany select texts

Oracle® Performance Tuning for 10gR2 Second Edition

Gavin Powell



ELSEVIER
DIGITAL
PRESS

Amsterdam • Boston • Heidelberg • London • New York • Oxford
Paris • San Diego • San Francisco • Singapore • Sydney • Tokyo

Elsevier Digital Press
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2007, Elsevier Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com.uk. You may also complete your request on-line via the Elsevier homepage (<http://elsevier.com>), by selecting "Customer Support" and then "Obtaining Permissions."

- ∞ Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data
Application Submitted.

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN-13: 978-1-55558-345-3

ISBN-10: 1-55558-345-8

For information on all Elsevier Digital Press publications visit our Web site at www.books.elsevier.com

06 07 08 09 10 9 8 7 6 5 4 3 2 1

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

Contents at a Glance

Preface	xxiii
Introduction	xxix
Part I: Data Model Tuning	I
1 The Relational Database Model	3
2 Tuning the Relational Database Model	27
3 Different Forms of the Relational Database Model	73
4 A Brief History of Data Modeling	81
Part II: SQL Code Tuning	89
5 What Is SQL?	91
6 The Basics of Efficient SQL	115
7 Advanced Concepts of Efficient SQL	167
8 Common-Sense Indexing	209
9 Oracle SQL Optimization and Statistics	255
10 How Oracle SQL Optimization Works	281
11 Overriding Optimizer Behavior Using Hints	347
12 How to Find Problem Queries	367
13 ^(10g)Automated SQL Tuning	409
Part III: Physical and Configuration Tuning	427
14 Tuning Oracle Database File Structures	429
15 Object Tuning	459
16 Low-Level Physical Tuning	475
17 Hardware Resource Usage Tuning	497
18 Tuning Network Usage	549
19 Oracle Partitioning and Parallelism	569
Part IV: Tuning Everything at Once	589
20 Ratios: Possible Symptoms of Problems	591
21 Wait Events	615
22 Latches	669
23 Tools and Utilities	685
24 The Wait Event Interface	739
25 ⁽⁹ⁱ⁾Tuning with STATSPACK	771



Appendices	589
A Sample Databases	799
B Sample Scripts	831
C Syntax Conventions	839
D Installing Oracle9i Database	841
E Sources of Information	879

Contents


Preface	xxiii
Introduction	xxix
Part I: Data Model Tuning	I
I The Relational Database Model	3
I.1 The Formal Definition of Normalization	3
I.1.1 Anomalies	4
I.1.2 Dependence and Determinance	5
I.1.3 First Normal Form (1NF)	6
I.1.4 Second Normal Form (2NF)	7
I.1.5 Third Normal Form (3NF)	7
I.1.6 Boyce-Codd Normal Form (BCNF)	8
I.1.7 Fourth Normal Form (4NF)	8
I.1.8 Fifth Normal Form (5NF)	8
I.1.9 Domain Key Normal Form (DKNF)	9
I.2 A Layperson's Approach to Normalization	9
I.2.1 First Normal Form	13
I.2.2 Second Normal Form	14
I.2.3 Third Normal Form	17
I.2.4 Beyond Third Normal Form	19
I.2.4.1 One-To-One NULL Separation Relationships	20
I.2.4.2 Separating Object Collections in Entities	22
I.2.4.3 Multicolumn Composite Keys	22
I.2.4.4 Summarizing a Layperson's Form of Normalization	24
I.3 Referential Integrity	25

2	Tuning the Relational Database Model	27
2.1	Normalization and Tuning	27
2.2	Referential Integrity and Tuning	28
2.2.1	Using Referential Integrity or Not	29
2.2.2	How to Implement Referential Integrity	31
2.2.2.1	Using Constraints (Primary and Foreign Keys)	33
2.2.2.1.1	Efficient Keys	34
2.2.2.1.2	Indexing Foreign Keys and Locking Issues	36
2.2.2.1.3	Sacrificing Referential Integrity for Performance	37
2.2.2.2	Coding Business Rules in the Database	39
2.2.2.2.1	Using Triggers for Referential Integrity	40
2.2.2.2.2	Using Triggers for Event Trapping	41
2.2.2.2.3	Using Stored Procedures and Functions	42
2.3	Optimizing with Alternate Indexes	44
2.4	Undoing Normalization	47
2.4.1	Denormalization	49
2.4.1.1	Reminding Ourselves about Normalization	49
2.4.1.2	Why Denormalize?	50
2.4.1.3	What to Look for to Denormalize	50
2.4.1.3.1	Mutable and Complex Joins	50
2.4.1.3.2	Mutable Joins to Find Few Columns	51
2.4.1.3.3	Adding Composite Keys	52
2.4.1.3.4	One-to-One Relationships	52
2.4.1.3.5	Many-to-Many Join Resolution Entities	53
2.4.1.3.6	Application Functions versus Entities	53
2.4.1.3.7	Static Data in Multiple Entities	53
2.4.1.3.8	Intermediary Entities Covering Summary Groupings and Calculations	54
2.4.1.4	Denormalizing by Reversing Normal Forms	54
2.4.1.4.1	Denormalizing Beyond Third Normal Form	55
2.4.1.4.2	Denormalizing One-to-One NULL Separation Relationships	55
2.4.1.4.3	Denormalizing Contained Object Collections	56
2.4.1.4.4	Denormalizing Multicolumn Composite Keys	57
2.4.1.4.5	Denormalizing Extra Entities for Common Columns	58
2.4.1.4.6	Denormalizing Formal Third Normal Form Transitive Dependencies	59
2.4.1.4.7	Denormalizing Calculated Columns	60
2.4.1.4.8	Denormalizing Formal Boyce-Codd Normal Form	61
2.4.1.4.9	Denormalizing Third Normal Form Many-to-Many Join Resolution Entities	61

2.4.1.4.10	Denormalizing Second Normal Form	63
2.4.2	Some Useful Tricks	64
2.4.2.1	Copying Columns between Entities	65
2.4.2.2	Placing Summary Columns into Parent Entities	66
2.4.2.3	Separating Active and Inactive Data	68
2.4.2.4	Mixing Heavily and Lightly Accessed Columns	68
2.4.2.5	Focus on Heavily Used Functionality	68
2.4.2.6	Using Views	69
2.4.2.7	Local Application Caching	70
2.4.3	Using Special-Purpose Oracle Database Objects	70
3	Different Forms of the Relational Database Model	73
3.1	The Purist's Relational Database Model	73
3.2	Object Applications and the Relational Database Model	75
3.2.1	The Object Database Model	75
3.2.2	The Object-Relational Database Model	78
3.2.3	The Benefits of Overlaying Objects onto Relations	78
4	A Brief History of Data Modeling	81
4.1	The History of Data Modeling	81
4.1.1	The Different Types of Data Models	82
4.2	The History of Relational Databases	85
4.3	The History of the Oracle Database	86
4.4	The Roots of SQL	87
	Part II: SQL Code Tuning	89
5	What Is SQL?	91
5.1	DML and DDL	91
5.1.1	DML Statement Syntax	92
5.1.1.1	The SELECT Statement	93
5.1.1.1.1	Logical Operators	93
5.1.1.1.2	Comparison Conditions	94
5.1.1.1.3	Types of SELECT Statements	96
5.1.1.1.4	Simple Query	97
5.1.1.1.5	Filtering Queries Using the WHERE Clause	97
5.1.1.1.6	Sorting Queries Using the ORDER BY Clause	97
5.1.1.1.7	Joining Tables	97

5.1.1.1.8	Types of Joins	98
5.1.1.1.9	Subqueries	100
5.1.1.1.10	Table and View Creation	102
5.1.1.1.11	Hierarchical Query	103
5.1.1.1.12	Set Operators and Composite Queries	103
5.1.1.1.13	Flashback	104
5.1.1.1.14	 Flashback Versions Queries	104
5.1.1.1.15	 Flashback Database	105
5.1.1.1.16	Using DISTINCT	105
5.1.1.1.17	The DUAL Table	105
5.1.1.1.18	NULLs	105
5.1.1.1.19	Pseudocolumns	106
5.1.1.1.20	Using Functions	107
5.1.1.2	The INSERT Statement	107
5.1.1.2.1	Multiple-Table INSERT Statements	107
5.1.1.3	The UPDATE Statement	108
5.1.1.4	The DELETE and TRUNCATE Statements	108
5.1.1.5	The MERGE Statement	109
5.2	Transaction Control	110
5.2.1	COMMIT versus ROLLBACK	111
5.2.2	Transaction Control between Multiple Sessions	113
5.3	Parallel Queries	113

6 The Basics of Efficient SQL 115

6.1	The SELECT Statement	117
6.1.1	A Count of Rows in the Accounts Schema	121
6.1.2	Filtering with the WHERE Clause	122
6.1.3	Sorting with the ORDER BY Clause	130
6.1.3.1	Overriding WHERE with ORDER BY	131
6.1.4	Grouping Result Sets	135
6.1.4.1	Sorting with the GROUP BY Clause	137
6.1.4.2	Using DISTINCT	138
6.1.4.3	The HAVING Clause	139
6.1.4.3.1	 The MODEL Clause	141
6.1.4.4	ROLLUP, CUBE, and GROUPING SETS	141
6.1.5	The FOR UPDATE Clause	144
6.2	Using Functions	145
6.2.1	The COUNT Function	145
6.2.2	The DECODE Function	147
6.2.3	Datatype Conversions	149
6.2.4	Using Functions in Queries	152


6.2.4.1	Functions in the SELECT Statement	152
6.2.4.2	Functions in the WHERE Clause	153
6.2.4.3	Functions in the ORDER BY Clause	153
6.2.4.4	Functions in the GROUP BY Clause	155
6.3	Pseudocolumns	155
6.3.1	Sequences	155
6.3.2	ROWID Pointers	157
6.3.3	ROWNUM	157
6.4	Comparison Conditions	158
6.4.1	Equi-, Anti-, and Range	158
6.4.2	LIKE Pattern Matching	160
6.4.3	Set Membership	161
6.4.4	Groups	165
7	Advanced Concepts of Efficient SQL	167
7.1	Joins	167
7.1.1	Join Formats	167
7.1.2	Efficient Joins	172
7.1.2.1	Intersections	172
7.1.2.2	Self-Joins	175
7.1.2.3	Equi-Joins and Range Joins	175
7.1.3	Inefficient Joins	176
7.1.3.1	Cartesian Products	176
7.1.3.2	Outer Joins	177
7.1.3.3	Anti-Joins	178
7.1.3.4	Mutable and Complex Joins	178
7.1.4	How to Tune a Join	179
7.2	Using Subqueries for Efficiency	179
7.2.1	Correlated versus Noncorrelated Subqueries	179
7.2.2	IN versus EXISTS	180
7.2.3	Nested Subqueries	180
7.2.4	Replacing Joins with Subqueries	181
7.2.4.1	Remove Tables without Returned Columns Using EXISTS	182
7.2.4.2	FROM Clause Subquery Nesting	187
7.3	Using Synonyms	191
7.4	Using Views	191
7.5	Temporary Tables	198
7.6	Resorting to PL/SQL	199
7.6.1	Tuning DML in PL/SQL	201
7.6.1.1	The RETURNING INTO Clause	202

7.6.2	When to Resort to PL/SQL and Cursors	203
7.6.3	Java or PL/SQL	204
7.7	Object and Relational Conflicts	206
7.7.1	Large Binary Objects in a Relational Database	206
7.7.2	Object-Relational Collections	207
7.8	Replacing DELETE with TRUNCATE	208
8	Common-Sense Indexing	209
8.1	What and How to Index	209
8.1.1	When Not to Use Indexes	210
8.1.2	Utilizing Referential Integrity Indexes	212
8.1.2.1	Alternate and Secondary Indexing	213
8.2	Types of Indexes	213
8.3	Types of Indexes in Oracle Database	214
8.3.1	The Syntax of Oracle Database Indexes	215
8.3.2	Oracle Database BTree Indexes	216
8.3.3	Read-Only Indexing	219
8.3.3.1	Bitmap Indexes	220
8.3.3.1.1	Are Bitmap Indexes Faster Than BTree Indexes?	222
8.3.3.1.2	Bitmap Index Locking	224
8.3.3.1.3	Using Composite Bitmap Indexes	224
8.3.3.1.4	Do Bitmap Indexes Overflow?	228
8.3.3.1.5	Bitmap Join Indexes	231
8.3.3.2	Clusters	231
8.3.3.2.1	Hash Clusters	232
8.3.3.2.2	^{10g} Sorted Hash Clusters	232
8.3.3.2.3	Index-Organized Tables	232
8.4	Tuning BTree Indexes	237
8.4.1	Overflow and Rebuilding	238
8.4.1.1	Lost Index Space	239
8.4.2	Reverse Key Indexes	240
8.4.3	Compressed Composite Indexes	241
8.4.3.1	Compressed Indexes and DML Activity	245
8.4.4	Function-Based Indexes	247
8.4.5	NULLs and Indexes	251
8.5	Summarizing Indexes	253
9	Oracle SQL Optimization and Statistics	255
9.1	What Is the Parser?	256
9.2	What Is the Purpose of the Optimizer?	257

9.2.1	What Does the Optimizer Do?	258
9.2.2	What Are Statistics?	259
9.2.3	Query Plan Access Paths	260
9.3	Rule-Based versus Cost-Based Optimization	261
9.3.1	Setting the Optimization Mode	261
9.3.2	What Was Rule-Based Optimization?	263
9.3.2.1	Outlines	264
9.3.2.2	9i Hints and Rule-Based Optimization	264
9.3.3	What Is Cost-Based Optimization?	266
9.3.3.1	Configuration Parameters and Cost-Based Optimization	266
9.3.3.2	The Importance of Statistics and Realistic Statistics	267
9.3.3.2.1	Dynamic Sampling	267
9.3.3.3	Generating Statistics	269
9.3.3.3.1	What to Generate Statistics For	269
9.3.3.3.2	Tables	269
9.3.3.3.3	Indexes	270
9.3.3.3.4	Columns	270
9.3.3.3.5	The ANALYZE Command	271
9.3.3.3.6	The DBMS_STATS Package	271
9.3.3.3.7	Automated Statistics Gathering	272
9.3.3.3.8	9i Automatic Statistics Generation in Oracle Database 9i	272
9.3.3.3.9	10g Automatic Statistics Generation in Oracle Database 10g	272
9.3.3.3.10	The SAMPLE Clause	274
9.3.3.3.11	Timed Statistics	275
9.3.3.4	Histograms	275



10 How Oracle SQL Optimization Works 281

10.1	Data Access Methods	281
10.1.1	Accessing Tables and Indexes	282
10.1.1.1	Full Table Scans	282
10.1.1.1.1	Reading Many Blocks at Once	283
10.1.1.1.2	Small Static Tables	284
10.1.1.1.3	Reading Most of the Rows	289
10.1.1.1.4	Reading Deleted Rows	291
10.1.1.1.5	Parallel Full Table Scans	293
10.1.1.2	Sample Table Scans	296
10.1.1.3	ROWID Scans	297
10.1.1.4	Index Scans	297



10.1.1.4.1	Index Unique Scan	298
10.1.1.4.2	Index Range Scan	299
10.1.1.4.3	Reverse-Order Index Range Scan	300
10.1.1.4.4	Index Skip Scan	301
10.1.1.4.5	Index Full Scan	303
10.1.1.4.6	Fast Full Index Scan	305
10.1.1.4.7	The DISTINCT Clause	307
10.1.1.4.8	The COUNT Function	308
10.1.1.4.9	Retrieving with NOT NULL	308
10.1.1.4.10	Parallel Index Scan	309
10.1.1.4.11	Index Join	311
10.1.1.4.12	Bitmap Join	312
10.1.1.5	Cluster and Hash Scans	313
10.1.2	Joining Tables	314
10.1.2.1	Join Order Execution	315
10.1.2.2	Types of Joins	315
10.1.2.2.1	Nested Loop Join	317
10.1.2.2.2	Hash Join	321
10.1.2.2.3	Sort-Merge Join	322
10.1.2.3	Mutable Join Nesting	325
10.1.2.4	Semi-Join	329
10.1.2.5	Joins to Avoid	330
10.1.2.5.1	Cartesian Join	330
10.1.2.5.2	Outer Join	331
10.1.2.5.3	 Grouped Outer Join	333
10.2	Sorting	333
10.2.1	Unique Sort	334
10.2.2	ORDER BY Sort	335
10.2.3	GROUP BY Sort	336
10.2.4	Sort Merge Join Sort	336
10.2.5	Aggregate Sort	337
10.3	Special Cases	337
10.3.1	Concatenation	337
10.3.2	The IN LIST Operator	340
10.3.3	UNION, MINUS, and INTERSECT	342

11 Overriding Optimizer Behavior Using Hints **347**

11.1	How to Use Hints	347
11.2	Hints: Suggestion or Force?	350
11.3	Classifying Hints	352
11.4	Influencing the Optimizer in General	353

11.4.1	Altering Table Scans	355
11.4.2	Altering Index Scans	356
11.4.3	Altering Joins	359
11.4.4	Cause Parallel SQL Execution	362
11.4.5	Altering Queries and Subqueries	362
11.5	 Naming Query Blocks for Hints	363
11.5.1	 Global Table Hints	363

12 How to Find Problem Queries 367

12.1	Tools to Detect Problems	367
12.2	EXPLAIN PLAN	368
12.2.1	What Does EXPLAIN PLAN Produce?	369
12.2.2	What to Look for in Query Plans	370
12.2.3	Problems Producing Query Plans	371
12.2.4	EXPLAIN PLAN Command Syntax	372
12.2.5	How to Create the PLAN_TABLE	376
12.2.6	What Is Not Provided in Query Plans?	376
12.3	SQL Trace and TKPROF	377
12.3.1	Setting up SQL Trace	377
12.3.1.1	Session-Level Tracing	379
12.3.1.2	Finding Trace Files	379
12.3.2	Using SQL Trace	380
12.3.3	TKPROF	383
12.3.3.1	Syntax of TKPROF	383
12.3.3.2	Using TKPROF	384
12.3.3.3	Interpretation of TKPROF Output	384
12.4	 TRCSESS	392
12.4.1	 End-to-End Tracing	393
12.5	Autotrace	394
12.6	Oracle Database Performance Views for Tuning SQL	396
12.6.1	Finding Cached SQL Code	396
12.6.1.1	Examining SQL Code	397
12.6.1.2	Hard-Hitting SQL Code	399
12.6.1.2.1	Using V\$SQLAREA	399
12.6.1.2.2	Executions	399
12.6.1.2.3	Disk + Buffer Reads per Row	400
12.6.1.2.4	Rows per Sort	401
12.6.1.2.5	Rows per Fetch	401
12.6.1.2.6	Parses per Execution	402
12.6.1.2.7	Disk versus Logical Reads	403
12.6.1.2.8	Using V\$SQL	404

12.6.1.2.9	Optimizer Cost	404
12.6.1.2.10	CPU Time	404
12.6.1.2.11	Elapsed Time	405
12.6.1.3	Examining Cached Query Plans with V\$SQL_PLAN	406
13	10g Automated SQL Tuning	409
13.1	Automatic Gathering of Statistics	410
13.2	The AWR and the ADDM	411
13.2.1	The AWR	411
13.2.2	The ADDM	413
13.3	Automating SQL Tuning	421
Part III: Physical and Configuration Tuning		427
14	Tuning Oracle Database File Structures	429
14.1	Oracle Database Architecture and the Physical Layer	429
14.1.1	The Oracle Instance	429
14.1.1.1	Buffers	430
14.1.1.2	Processes	430
14.1.2	The Oracle Database or File System Layer	432
14.1.2.1	How Oracle Database Files Fit Together	433
14.1.2.1.1	Special Types of Datafiles	434
14.1.2.2	Tuning Datafiles	435
14.1.2.3	Controlfiles	436
14.1.2.4	Tuning Redo Logs and Archive Logs	437
14.1.3	The Networking Layer	440
14.2	Tuning and the Logical Layer	440
14.2.1	Tablespaces	441
14.2.1.1	9i Dictionary-Managed Tablespaces	443
14.2.1.2	10g Locally Managed Tablespaces	443
14.2.1.2.1	Auto Extend	445
14.2.1.2.2	Minimum Extent Sizes	446
14.2.1.2.3	Block Size	446
14.2.1.2.4	Logging	446
14.2.1.2.5	Extent Management	447
14.2.1.2.6	Segment Space Management	448
14.2.1.2.7	10g BIGFILE Tablespaces	449
14.2.1.2.8	Avoiding Datafile Header Contention	449
14.2.1.3	Temporary Sort Space	450

14.2.1.3.1	(9i) Temporary Tablespaces in Oracle Database 9i Database	450
14.2.1.3.2	(10g) Temporary Tablespaces in Oracle Database 10g	451
14.2.1.3.3	Tablespace Groups	453
14.2.1.4	Manual Rollback and Automatic Undo	453
14.2.1.4.1	(10g) Automated Undo	454
14.2.1.4.2	(9i) Manual Rollback Segments	455
14.3	Automating Database File Structures	456
14.3.1	Oracle Managed Files	456
14.3.2	(10g) Automatic Storage Management	457

15 Object Tuning 459

15.1	Tables	459
15.1.1	Caching	460
15.1.2	Logging	460
15.1.3	Table Parallelism	461
15.1.4	Storing LOBs Separately	464
15.1.5	Dropping Columns	465
15.1.6	Deallocating Unused Space	466
15.2	Indexes	467
15.2.1	Monitoring	467
15.2.2	Index Parallelism	469
15.2.3	Fragmentation and Coalescing	470
15.3	Index-Organized Tables and Clusters	471
15.4	Sequences	472
15.5	Synonyms and Views	472
15.6	(10g) The Recycle Bin	473

16 Low-Level Physical Tuning 475

16.1	What Is the High-Water Mark?	475
16.2	Space Used in a Database	476
16.3	What Are Row Chaining and Row Migration?	477
16.4	Different Types of Objects	478
16.5	How Much Block and Extent Tuning?	479
16.6	Choosing Database Block Size	479
16.7	Physical Block Structure	481
16.7.1	What Is in a Block?	482
16.7.2	Block Space Management	484
16.7.2.1	Assessing PCTFREE Settings	489
16.7.3	Block Concurrency	490

16.8	Extent Level Storage Parameters	493
16.8.1	Setting Extent Sizes	493
16.8.2	Minimum and Maximum Extents	494
16.8.3	Variable Extent Sizes	494
16.8.4	Managing Concurrency	495
16.8.5	Minimizing Rollback Resizing	495
16.8.6	Different Cache Recently Used Lists	496









17 Hardware Resource Usage Tuning 497

17.1	Tuning Oracle CPU Usage	497
17.1.1	Busy I/O and Intense CPU Activity	498
17.1.1.1	Swapping and Paging	498
17.1.2	Possible Causes of High CPU Activity	499
17.1.2.1	Poorly Tuned SQL Code	499
17.1.2.2	Poor Index Usage	500
17.1.2.3	10g Automated Undo and 9i Manual Rollback	501
17.1.2.4	Temporary Sort Space	506
17.1.2.5	Row Locks and Latch Waits	507
17.1.2.6	High Network Activity	508
17.2	How Oracle Database Uses Memory	509
17.2.1	The System Global Area	514
17.2.1.1	10g Automated SGA Memory Management	514
17.2.1.1.1	Automated SGA Performance and Monitoring	514
17.2.1.2	9i Manual SGA Memory Management	516
17.2.1.2.1	The Database Buffer Cache	517
17.2.1.2.2	Database Buffer Cache Advice	518
17.2.1.2.3	The Shared Pool	522
17.2.1.2.4	The Library Cache	524
17.2.1.2.5	The Metadata or Dictionary Cache	527
17.2.1.2.6	Pinning Objects in the Shared Pool	529
17.2.1.2.7	Shared Pool Advice	529
17.2.1.2.8	The Large Pool	532
17.2.1.2.9	Shared Servers and Virtual Circuits in the Large Pool	532
17.2.1.2.10	The Streams Pool	532
17.2.1.2.11	The Java Pool	532
17.2.2	The Program Global Area	532
17.2.2.1	10g Automated PGA Memory Management	533
17.2.2.1.1	Automated PGA Performance and Monitoring	535
17.2.2.2	9i Manual PGA Memory Management	537
17.2.3	Other Memory Buffers	538



17.2.3.1	The Redo Log Buffer	538
17.3	Tuning Oracle I/O Usage	540
17.3.1	RAID Arrays	545
17.3.2	^{10g} Oracle Automatic Storage Management	546
17.3.2.1	ASM Performance	547
17.3.2.2	Administrative Pros and Cons of ASM	547
17.3.2.3	ASM High-Availability Features	548
18	Tuning Network Usage	549
18.1	The Listener	549
18.1.1	Listener Queue Size	550
18.1.2	Switching Off Listener Logging and Tracing	551
18.1.3	Multiple Listeners and Load Balancing	552
18.2	Network Naming Methods	553
18.2.1	Local Naming	555
18.2.1.1	Dedicated Versus Shared Servers	555
18.2.1.2	The Session Data Unit Buffer (SDU)	556
18.3	Connection Profiles	557
18.4	Shared Servers	560
18.4.1	⁹ⁱ Shared Server Configuration Parameters	560
18.4.1.1	Oracle Database 9i Shared Server Configuration	561
18.4.2	Network Performance Views	562
18.4.2.1	Shared Servers	563
18.4.2.2	Dispatchers	564
18.4.2.3	Virtual Circuits	565
18.4.2.4	Using Events	565
19	Oracle Partitioning and Parallelism	569
19.1	What Is Oracle Partitioning?	569
19.1.1	Why Is Oracle Partitioning Beneficial?	570
19.1.2	How Are Tables and Indexes Partitioned?	571
19.1.3	Oracle Partitioning Methods	573
19.1.3.1	Partitioning by Range	573
19.1.3.2	Partitioning by List	580
19.1.3.3	Hash Partitions	583
19.1.3.4	Composite Partitions	584
19.2	Tricks with Partitions	586

Part IV: Tuning Everything at Once 589

20 Ratios: Possible Symptoms of Problems 591

20.1	Database Buffer Cache Hit Ratio	592
20.1.0.1	Multiple Database Buffer Cache Pools	595
20.1.0.1.1	The Default, Keep, and Recycle Pools	595
20.1.0.1.2	 Multiple Block-Sized Caches	599
20.2	Table Access Ratios	603
20.3	Index Use Ratio	606
20.4	 Dictionary Cache Hit Ratio	607
20.5	 Library Cache Hit Ratios	607
20.6	 Disk Sort Ratio	608
20.7	 Chained Rows Ratio	609
20.8	 Parse Ratios	610
20.9	 Latch Hit Ratio	611
20.10	 Ratios in the Database Control	612

21 Wait Events 615




21.1	Idle Events	616
21.1.1	 Idle Events in Oracle 9i Database	617
21.1.2	 Idle Events in Oracle 10g Database	619
21.2	Significant Events	620
21.2.1	Buffer Busy Waits	628
21.2.1.1	Causes of Buffer Busy Waits	634
21.2.1.2	Decreasing Buffer Busy Waits	637
21.2.2	Datafile Scattered and Sequential Reads	645
21.2.3	Direct Path Reads and Writes	649
21.2.4	Free Buffer Waits	649
21.2.5	Row Cache Lock Waits	650
21.2.6	Library Cache Waits	650
21.2.7	Redo Log Waits	652
21.2.8	Undo and Rollback Waits	656
21.2.9	Enqueue Waits	658
21.2.10	Latch Free Waits	666
21.3	 Wait Events in the Database Control	667

22 Latches 669

22.1	What Is a Latch?	669
22.1.1	Latch Misses, Spins, and Sleeps	670

22.1.2	Latch Performance Views	673
22.1.3	Latches in Real Time	674
22.2	The Most Significant Latches	676
22.2.1	The Database Buffer Cache	677
22.2.2	The Shared Pool	680
22.2.2.1	Library Cache Latches	680
22.2.2.2	Metadata Cache Latches	683
22.2.3	The Redo Log Buffer	683
22.2.4	Network and Database Connection Latches	683

23 Tools and Utilities 685

23.1	 Oracle Enterprise Manager	685
23.1.1	Diagnostics Pack	686
23.1.1.1	Event Monitoring	686
23.1.1.2	Lock Monitoring	686
23.1.1.3	TopSessions	687
23.1.1.4	TopSQL	687
23.1.1.5	Performance Manager	687
23.1.2	Tuning Pack	688
23.1.2.1	Tablespace Map and the Reorg Wizard	689
23.1.2.2	Tools Useful for Tuning SQL Code	693
23.1.2.2.1	Index Tuning Wizard	694
23.1.2.2.2	SQL Analyze	700
23.1.2.2.3	Oracle Expert	708
23.2	 The Database Control	709
23.2.1	Proactive Maintenance	713
23.2.2	Performance Architecture of the Database Control	713
23.2.2.1	Statistics Automation	714
23.2.2.2	Performance Metrics	718
23.2.2.2.1	Baseline Metrics	719
23.2.2.2.2	Metric Thresholds	719
23.2.3	Advice Performance Tools	720
23.2.3.1	The Segment Advisor	721
23.2.3.2	Undo Management (Undo Advisor)	722
23.2.3.3	The Memory Advisor	723
23.2.3.4	The SQL Access Advisor	726
23.2.3.5	The SQL Tuning Advisor	727
23.2.3.6	The MTTR Advisor	728
23.2.4	Dealing with Locks	728
23.3	 Spotlight	729
23.4	Operating System Tools	729

23.4.1	Windows Performance Monitor	731
23.4.2	Unix Utilities	733
23.5	Other Utilities and Tools	733
23.5.1	Data Pump, Import, Export, and SQL*Loader	733
23.5.1.1	Data Pump Import and Export	734
23.5.2	Resource Management and Profiling	734
23.5.3	Recovery Manager (RMAN)	735
23.5.4	Transportable Tablespaces	735
23.5.5	Oracle Streams	735
23.5.6	Oracle RAC and Oracle Grid	735
23.5.7	9i STATSPACK	735
24	The Wait Event Interface	739
24.1	What Is a Bottleneck?	739
24.2	Detecting Potential Bottlenecks	740
24.3	What Is the Wait Event Interface?	741
24.3.1	The System Aggregation Layer	743
24.3.2	The Session Layer	750
24.3.3	The Third Layer and Beyond	758
24.4	10g Oracle Database Wait Event Interface Improvements	760
24.5	9i Oracle Enterprise Manager and the Wait Event Interface	762
24.6	10g The Database Control and the Wait Event Interface	765
25	9i Tuning with STATSPACK	771
25.1	Using STATSPACK	771
25.1.1	An Example STATSPACK Report	772
	Appendices	589
A	Sample Databases	799
B	Sample Scripts	831
C	Syntax Conventions	839
D	Installing Oracle 9i Database	841
E	Sources of Information	879
	Index	881

Preface

This book is about tuning Oracle databases. Three areas of Oracle Database tuning are data model tuning, SQL code tuning, and physical and configuration tuning. The author began his career as an applications developer, not as a systems or network administrator. As a result, this book is written from an applications rather than an operating system perspective.

The objective of this book is to cover all three areas of Oracle database tuning. Currently, no title on the market completely covers all of these areas. This book will cover all three by explaining both problem detection and resolution.

The approach in this book is to present something that appears to be immensely complex in a simplistic and easy-to-understand manner. Both reference material and examples are utilized appropriately in order to expedite understanding for the reader.

Reference material is not overused. Oracle software has been in general use commercially for many years and is intended to provide for a very large and diverse customer base. Features are often not removed from Oracle software between versions, and new features are continuously being added. The result is that Oracle software contains a plethora of available options and features. Using only reference information to explain Oracle Database tuning would therefore be difficult to read, contrary to the approach of this book, and would not provide the reader with much practical advice. This book contains a lot of examples, with realistic databases and data, sometimes even very large amounts of data. After all, if your production database needs tuning, you probably have more data than you first expected.

A broad-based tutorial on the subject of tuning Oracle Database is much needed. Most database administrators have operating system administration experience, and little SQL code or data modeling experience. On the other hand, developers have the opposite. This book targets both devel-

opers and database administrators because it includes all three areas essential to tuning Oracle installations effectively. The important factor is that all tuning skills—both administration and development skill sets—are required for best performance.

Being a broad-based tutorial, this title is written to reach the widest possible audience, including data modelers, developers, database administrators, and system administrators. Each of these audiences is very specialized, but all are related and interdependent. No existing titles include tuning for data models, tuning of SQL code, and physical and configuration tuning all in one book.

People who would benefit from reading this book are database administrators, developers, data modelers, systems or network administrators, and technical managers. Technical people with these different skills are often ignorant of the skills of each other. This is a great pity because all skill sets are very much dependent on each other for well-constructed databases and applications. Let's take a common example situation. Developers cannot simply hand off SQL code tuning to database administrators when application coding is complete. Database administrators, more often than not, know absolutely nothing about SQL code tuning. The result is that no SQL code tuning is ever done, and too much time is spent squeezing out an extra 10% of performance, with the database administrator doing physical and configuration tuning. Targeting a few hard-hitting SQL statements will probably result in much more than a 10% performance improvement, which is much more productive.

What is in this book?

Data Model Tuning

What is the data model?

The data model is the table structure and the relationships between those tables. Tuning the data model for performance involves normalization and denormalization. Different approaches are required depending on the type of database installation, such as OLTP (the focus of this book) or data warehouse-type databases. Inappropriate database design can make SQL code impossible to tune. If the data model is poor, changing the data model can have the most profound effect on database performance. All SQL code is constructed from the underlying tables. The big problem is that altering the data model after completion of development is expensive because application code may require extensive rework.

Note: OLTP refers to online transaction processing. OLTP generally implies the Internet. Within the scope of this book, OLTP is used to represent both OLTP architectures and perhaps client/server architectures as well.

What in the data model causes problems, and what is data model tuning?

Data model tuning is most effectively performed by a combination of both database administrators and developers. It is seldom the case that both skill sets are involved, however. The result is that table structures are often either development-centric (top-down), or administration-centric (bottom-up) in design. Java development is often top-down and attempts to impose an object structure over a relational framework. Bottom-up design often results in overnormalization and too much granularity. People with different skills should be working together.

What are the benefits of data model tuning?

Tuning the data model can often provide performance improvements in excess of 100%, but it is expensive because application code can be drastically affected.

SQL Code Tuning

What is SQL code?

SQL code is the code directly accessing the database, embedded either in applications or in stored procedures. Sometimes generic SQL code is used, which is SQL code generated by an application on an ad hoc basis. Generic SQL code can cause serious performance issues.

What causes SQL code performance problems, and what is SQL code tuning?

As with data modeling, it is often confusing to determine which personnel skill sets are responsible for SQL code tuning. This is one of the causes of poorly performing SQL code. Performance is served most effectively when developers and database administrators work together to a certain extent.

Poorly written SQL code is often the biggest culprit of performance problems, because it is expensive to rectify, but it is cheaper than changing the data model. SQL code tends to be contained inside independent blocks within applications or stored procedures. This containment is commonly known as embedded SQL code. Tuning SQL code is in general a two-step process, as follows:

1. Isolation and recoding of the worst-performing SQL statements, perhaps the slowest-performing 10% of SQL code.
2. General tuning of SQL code involving changes to SQL statements throughout applications and the database, plus adjustments to alternate (secondary) indexing. Alternate indexing is not specifically part of the steps of normalization and denormalization but can be designated as data modeling or SQL code tuning. It is important that database administrators have some involvement with respect to alternate indexing, at the very least in an advisory capacity. Too many or inappropriately constructed alternate indexes can completely destroy performance.

What are the benefits of SQL code tuning?

SQL code tuning can increase performance between 25% and 100%, sometimes much more. In rare situations, I have seen enormous performance improvements when tuning SQL code. One or two projects I have worked on in the past have been sped up 30 to 500 times, for both individual SQL code statements and sometimes even the applications in general. Additionally, SQL code is often embedded in applications in such a way that changing the SQL code does not affect application functionality.

Physical Database and Configuration Tuning

What is physical and configuration tuning?

Physical database tuning involves hardware resource usage, networking, and various other administration tasks, such as configuration and file distribution.

What causes physical and configuration performance problems?

Physical configuration is usually a culprit of poor performance where Oracle software is installed with defaults and never altered by an expert. Developers often build table structures and SQL code. In this case, physical tuning is relied on to solve performance problems. This is usually a mistake because physical configuration tuning usually only provides at most 10% to 20% performance improvement.

What are the benefits of physical and configuration tuning?

Physical and configuration tuning usually only results in at most a 25% performance improvement—and usually a lot less. The relative cost of using physical and configuration tuning only is usually not cost effective. Hardware upgrades are common in these situations.

Hardware Upgrades

As a side issue, there are always potential hardware upgrades, which are sometimes a short-term solution because this approach does not necessarily tackle the real problems. Sometimes a combination of hardware upgrades and Oracle installation tuning is the most cost-effective option. Hardware upgrades can often be more expensive than tuning. Three months of SQL code tuning by an expert may be much more cost effective than purchasing new machines, RAID arrays, and all the other bits and pieces that go with it. Additionally, an expert can teach developers and database administrators to build properly tuned databases and applications in the first place.

Tuning with the Database Control

Oracle Enterprise Manager has now reached its maturity in Oracle Database 10g, with the introduction of the Database Control. The Database Control runs in a browser, across a network. The Database Control is a magnificent tool that is useful for administration, maintenance, and performance tuning.

Sample Databases Used in This Book

Several sample databases are utilized in this publication. Some are simple and some are highly complex, depending on their use when explaining aspects of tuning. All details and diagrams are included in full in Appendix A.

Please note that this book does not cover operating system tuning or data warehouse tuning, even though they may be mentioned or alluded to in many places. I do not claim to know everything about Oracle Database performance tuning. However, I do have 18 years of custom applications development and database administration experience. What I do know is shared in this book.

Note: Multiple versions of Oracle Database are covered in this book, up to and including Oracle Database 10g, Release 2.

I received a few complaints from the first edition of this book with respect to it not being up to date with Oracle Database 10g. This second edition is written with the most recent production release of Oracle Database 10g Release 2.

Note: The statements and opinions expressed in this book are often my own and do not necessarily represent the opinion of any corporation, company, or anyone else. And don't run anything without testing it first!

Let's get started.

Introduction

Let's begin by looking at what we need to examine in order to tune Oracle installations:

- A tuning environment
- When to tune
- What to tune
- When to stop tuning
- Tuning from development through to production

Finally, we will briefly describe how this book is organized.

A Tuning Environment

What is a tuning environment? A tuning environment is an environment in which your tuning efforts can be productive.

What Is Required When Tuning Oracle Database?

- Good software tools
- Skilled personnel
- Staging (testing) environments
- A realistic duplication of the production environment:
 - *Actual and expected production environments.* These can often be different if growth is rapid or requirements change.

- *If possible, databases should be of the same size and content.* If this is impractical, then at least development and testing databases should be proportionate to production.
- *Are the statistics the same?* Statistics can be copied or executed using the same time intervals as production.

What Tools Are Available?

Excellent software tools for tuning and monitoring Oracle databases are numerous. Oracle Enterprise Manager has many very useful bells and whistles. Spotlight is excellent for visual and informative real-time monitoring of busy production systems. Both Oracle Enterprise Manager and Spotlight are very useful as tuning aids for physical and SQL code performance analysis.

Many other tools are available. The most important tools in the tuning process are the developers and the administrators. That is why you are reading this book. The best software tools are usually the most expensive, but that does not mean that the less expensive tools are useless. In general, the more expensive tools tend to do more for you. However, when something is being done for you automatically and you do not understand the internals, it is unlikely that your tool set can do better than well-trained, experienced database administrators and developers.

Skilled Personnel

Good skills have their correct places. Database administrators tend to have roots as either system administrators or developers. Each skill set has its pros and cons. Developers tend to know a lot more about coding SQL and building data models. System administrators have extensive knowledge of operating systems such as UNIX and tend to concentrate tuning efforts on the physical aspects of Oracle Database. Developers tend to concentrate on tuning the data model and building efficiently performing SQL code. Unfortunately, this is not always the case, because there is sometimes a tendency for developers to place the burden of tuning SQL code and the data model into the database administrators' responsibility. Confusion can result, and perhaps nothing gets done.

Staging (Testing) Environments

You need as many testing and staging environments as you can get. As the resident DBA, you should not be expected to perform database tuning on an active production database that is required to be up and usable 24x7x365. Tuning on a production database in this situation will limit your scope and could cost you your job! Insist on at least one extra machine and

always test anything you do, no matter how trivial. This is the most important difference between production and development. Developers do everything quickly because they have to. Production database administrators are expected to get their tasks done just as fast, but additionally, everything must be perfect all the time. So make sure you insist on extra hardware and extra time.

Duplicating Production Databases for Effective Tuning

It is absolutely essential to have the most recent and most realistic copy of a production database for tuning purposes. Tuning on a production database is risky, and using a development database for tuning can be completely useless. It is extremely rare that development and production databases are alike. Testing databases can be useful when development and production databases cannot be made the same.

Statistics are also important. In a production environment such as an online transactional database, the data in that production database could be changing constantly. Even if your database is not changing too much, statistics could change or rapidly become out of date. The more dynamic the data is, the more often statistics should be updated. The SQL code optimizer utilizes statistics to compile the most efficient methods of executing SQL statements. Statistics are measurements of the data itself, such as how large a table is and how useful an index is. When a SQL statement accesses a table, both the table and index states are important. States of database objects such as tables and indexes are contained within statistics. If statistics are out of date, the optimizer is not functioning realistically. Out-of-date statistics would have the same effect on all types of databases. It is very important to duplicate statistics from production to tuning environments, either by copying or by executing statistics gathering on a tuning database, consistent with the production database.

Making a copy of a production database to a development database is not an issue when the production database is small. When the production database is large, however, continuous copying to a development database could be very time consuming. Be aware that using the database import utility for even single-schema imports on even a small database can take a lot longer than the production database export.

Note: The DBMS_STATS package can be used to copy statistics between databases.

When to Tune

When should you tune? Constantly and throughout the life cycle of your product, perhaps only when performance is a problem. On the other hand, the database should be configured and organized properly in the first place to avoid having to continuously put out fires. If you are always putting out fires, the chances are you will make little progress with anything else. It is much better to be proactive and preempt potential problems rather than to react when they occur. It is preferable to tune your database during development rather than after development in production.

Tuning during development will lengthen the life cycle of any system. Why is this the case? If different tasks can be compartmentalized during distinctly separate phases of a project, such that those phases do not overlap each other, then a better product will result. For instance, any developer knows that changing code in order to tune SQL code, after completion of development, could change the integrity and neatness of that code. This may not seem too important, but the more times a piece of code is altered, the more it will deteriorate, not only because the code is changing but also because different coders may be changing that code. Every coder has a different style and approach, and subsequent coding is often confused by code written by other people and vice versa. Errors can be introduced into application code when it is changed. The more changes, the more potential errors.

We do not live in a perfect world. Distinct lines cannot be drawn between development and production. Changing requirements often cause changes in different project phases, making gray areas. There is often overlap between development and production.

It is best to tune during development, particularly data models and SQL code. When tuning after development in a postdevelopment requirements change phase, or in production, tuning should be done carefully, especially when it comes to changing application code. If tuning cannot be done fully in the development cycle, which it probably cannot, take the following approach when tuning in production:

- Set performance targets.
 - Use test environments.
 - Use test environments that match production as closely as possible.
 - Tune with care.
-

What to Tune in Production

There are five general stages in tuning an Oracle production database:

1. Resolve obvious bottlenecks.
2. Examine basic configuration. Configuration is often inappropriate as a result of initial Oracle installation and database creation.
3. Physical space can be wasted. Often a database can be better organized and become much smaller, even as much as one-tenth of current size, and sometimes more.

Note: **10g** Oracle Database is becoming more automated with respect to physical configuration.

4. Poorly written SQL code in both applications and in the database can only be counteracted partially by physical and configuration tuning. SQL code tuning can help performance immensely but should be performed in development and testing environments first.
5. Data model tuning may be required if SQL code tuning does not resolve production performance problems. As with SQL code tuning, data model tuning should be done in development and testing environments first.

The easiest approach to tuning during production is Oracle Database physical and configuration tuning. In general, tuning the data model is the most expensive because it will require changes to SQL code, application code, and the production database. If SQL code is not tunable, then your data model may be too granular or not properly normalized. Physical and configuration tuning on your database server will not require SQL code and data model tuning changes. However, the purely database-oriented tuning approach may eventually lead to expensive hardware upgrades, which can often be the best tuning option. However, costs can sometimes be greater when the hardware becomes so complex that highly skilled, expensive administrators are required. Hardware upgrades are often a short-term solution, and their cost effectiveness can be appropriate if the software life cycle is short or money is tight, assuming the hardware is reusable or sellable.

When to Stop Tuning During Production

When do you stop tuning? This is always debatable. You could stop tuning when performance targets are met, depending on what needs to be tuned. The resolution of obvious bottlenecks is a clear indicator that no more tuning is required. Physical tuning (i.e., configuration, physical database structure, networking, and hardware bottleneck issues) can often only amount to as little as 10% of total effective tuning activity, both in development and production.

The simple approach to when to stop tuning is to teach your developers to build properly tuned SQL code from the outset, and make sure that the data model is sound before you do even that. This is probably impossible, but the more SQL code tuning that is done during the development process, the fewer problems you will have later on. Many software projects are discarded because they take too long to develop. However, many other software projects are thrown out or rewritten because they do not meet acceptable performance standards, rendering them useless. Tuning data models and SQL code after completion of development can sometimes simply be too expensive.

When to stop tuning depends on your situation and the skills you have. If the company and your database size and activity grow, you will have performance problems anyway, but you can be better prepared. Let us examine the steps in tuning production databases in order to decide when to stop tuning the different areas.

Bottlenecks

Solving performance bottlenecks is usually reactive rather than proactive. The term *bottleneck* is technical computer jargon that usually deals with a particular facet of your environment, which is overloaded, be it within or outside of your database.

Note: Stop tuning when the bottleneck is removed.

Configuration

If there are immense configuration and physical problems, some downtime may be required. Configuration issues are easier to resolve than physical problems, and both are easier than tuning data models and SQL code.

Configuration can be as simple as setting parameters correctly in the Oracle Database configuration parameters file and Oracle networking software configuration files. Make absolutely sure that configuration parameters are completely understood before changing them. First, incorrectly formed configuration can prevent the database from starting and perhaps cause a crash. Second, some parameters have very specific functions; incorrect settings can cause a totally different and probably undesired effect.

Stop tuning when configuration parameters are correct. Experimenting with changing configuration parameters on a production database is risky, so test, test, test!

Physical Space Usage

Physical space usage and growth tuning for performance includes tuning of datafiles, redo logs, archive logs, and rollback segments.

Note: (10g) Manual rollback is deprecated. Use automated undo.

Resolving physical problems with space usage and growth can cause a lot of downtime but may be the only solution. Small databases have small initial configuration parameter settings. If those small databases suddenly start to get very busy, then immense problems can occur. The sooner the physical issues are resolved for a rapidly growing database, the better. If the database continues growing, the temptation is often to spend enormous amounts of money on hardware. Organizing a database physically can essentially save a lot of disk space.

Note: (10g) The current trend in Oracle Database is veering toward automated management of physical space. As a result, physical space management to save disk space is becoming less important.

The smaller a database is, the less disk space is needed to search through when finding data, thus the faster your data searches will be. Additionally, highly fragmented data can cause a lot of “bouncing around” when retrieving data.

Stop tuning when performance is acceptable, as long as requirements for uptime are not compromised. Only extreme situations of database growth cause problems with use of physical space.

SQL Code Tuning

Poorly constructed SQL code usually causes most database performance problems. When SQL code is poorly tuned, database administrators can do little to improve performance using physical and configuration tuning alone. Database administrators can tune SQL code contained in PL/SQL stored procedures. The most ideal approach to SQL code tuning is to teach your developers to tune SQL code as they build applications.

Sometimes developers will build applications rapidly without much consideration for building efficient SQL code. Developers do not really have extra time to make sure the SQL code is as efficient as possible. Most production database administrators tend to have roots as either operating system or network administrators. These skills are essential for production database administration, and there's the rub! The administrators sometimes do not know how to tune SQL code, and they cannot change the application code because the code belongs to the developers. The developers do not have the time or the skills to produce efficient SQL code. Most Oracle Database tuning experts perform physical tuning on the database and the operating system, not SQL code tuning. The result is often only a 10% performance improvement. I have seen SQL code tuned for an application in its entirety and performance increases of 30 to 500 times. That is 500 times faster. One hundred percent is twice as fast. This is an extreme from a consulting job I worked on a few years ago. There were other projects in the past with similar performance issues.

Stop tuning SQL code when development stops, if you can. Teach and encourage your developers to write efficient SQL code during development. It is much more difficult and expensive to tune SQL in production, especially when SQL is embedded in applications.

Data Model Tuning

SQL code performance depends largely on the data model, especially if the data model is poorly structured. Beware of copying older, previously invented data models or using a data model because it is the accepted standard for a particular application. Relational databases, normalization, and denormalization have existed for many years. However, the structure of the applications overlaying those relational databases has changed recently and is still changing. What in the past was COBOL and C is now C++, Java, Perl, and even object Perl.

Object-oriented design application development languages such as Java tend to confuse what used to be accepted relational database design. Object applications tend to impose an unwilling structure on top of a relational database. The result is a data model that is a hybrid between relational and object database methodologies. This hybrid structure can be an advantage for OLTP and client/server transactional applications but a performance hog for any type of reporting or data warehouse applications.

I have seen many OLTP applications with Java object structures imposed onto relational databases. Sometimes this top-down application to data model approach works extremely well, sometimes very poorly. Object methodology promotes breaking things down into their smallest manageable parts. This approach can be a complete disaster for efficiency in a relational database. Imposing an object structure onto a relational database in its purest form is relational database normalization in its extreme. In these situations, third, fourth, and fifth normal forms are common. A partial solution is often two databases—one OLTP and the other a data warehouse—doubling costs. The second database is a denormalized data warehouse database. Denormalizing a database can enhance performance from a reporting perspective.

Tuning the data model is the most difficult and most expensive option because SQL code depends on the structure of the data model; extensive application code changes can result. Tuning the data model is more effective than physical and configuration tuning but can easily escalate into a full rewrite.

If further data model tuning is required after production release, you may want to look at data warehouse-type options. Tuning the data model for an OLTP database after production release will generally involve a partial or complete rewrite. A data warehouse-type approach generally involves duplicating a database and restructuring that duplicate for output performance in terms of processing many rows at once. Transactional databases in OLTP and client/server environments are often required to be tuned for small-response reaction time to keep your customers happy—very few rows retrieved at a time. If a Web page takes more than seven seconds to load, then your customers may lose interest and go elsewhere, straight to the competition. Data warehouse databases, however, are designed for rapid throughput of large amounts of information for analytical and reporting purposes.

Stop tuning your data model, preferably before development starts or before the first production release. Do not build your relational database to mimic the object structure of a Java application because

relational and object methodologies are completely opposed to each other with respect to methodology. Object structures break down for simplicity, and relational structures are efficient when summarizing information into groupings. These structural types are completely contrary to each other.

So when should you stop tuning in general? As long as there are problems or if users are not satisfied—never! Tune whenever something changes if there are problems or when you have not reached a performance target. There is no harm in going past a target level of performance. Then again, time might be better utilized doing other tasks. You should stop tuning the data model before SQL code development starts. You should stop tuning SQL code when development finishes and the code goes into production.

Tuning from Development to Production

Remember this: Probably most Oracle client installations are very small databases, even for some large companies. For instance, many databases for a lot of the Silicon Valley-based dot-com companies of the late 1990s were well under 10 gigabytes, sometimes even as small as being less than a single gigabyte. They expected growth. The point is this: Many databases are small. A lot of Oracle clients are small companies. The result is that Oracle installation software and database creation tools tend to cater to those small companies. Oracle Corporation's strategy in this respect is perfectly sensible because the smaller companies lack the funds for highly skilled staff. The smaller companies need more done for them. Large end-user software vendor companies often take this approach. The result is that in the Oracle installation software, most of the configuration parameters and physical settings for files are much too small for any database that experiences a reasonable amount of growth. If a database is over 10 Gb, is highly active, or is growing rapidly, then configuration created by Oracle installation software and database creation tools is probably inappropriate.

Tuning an Oracle database is not just tuning the database. As we can see from the stages of tuning already discussed, tuning the database also includes tuning the data model, SQL code, and thus applications. Tuning Oracle Database is a combination of tuning both the Oracle database server and the applications accessing that database. There is a process of tuning an Oracle Database environment including a large number of separate steps or phases. These phases are repetitive but should be performed as closely as possible to the sequential order as shown in Figure I.1.

The Steps in Tuning

Tuning is a set of distinct steps, which can be repeated in any order but are preferably completed in the order shown in Figure I.1. The order and existence of these steps could vary depending on the application type, skills resources, time for development, and capacity of available hardware.

Figure I.1
*The steps in tuning
an Oracle
installation*

- ➔ **Physical tuning**
 - ➔ Initial check of configuration parameters
- ➔ **Data model tuning**
 - ➔ Normalization
 - ➔ Denormalization
 - ➔ Alternate (secondary) indexing
 - ➔ Constraints, procedures, event and state change triggers
 - ➔ Implementing referential integrity
- ➔ **SQL code tuning**
 - ➔ PL/SQL embedded SQL in the database
 - ➔ Applications embedded SQL
 - ➔ Alternate (secondary) indexing
- ➔ **Physical tuning**
 - ➔ Configuration parameters
 - ➔ I/O contention
 - ➔ Space usage
- ➔ **Operating system and network tuning**

The steps in tuning an Oracle installation should more or less follow the same path as in the development cycle of software development, namely analyze, design, build, test, implement, and verify.

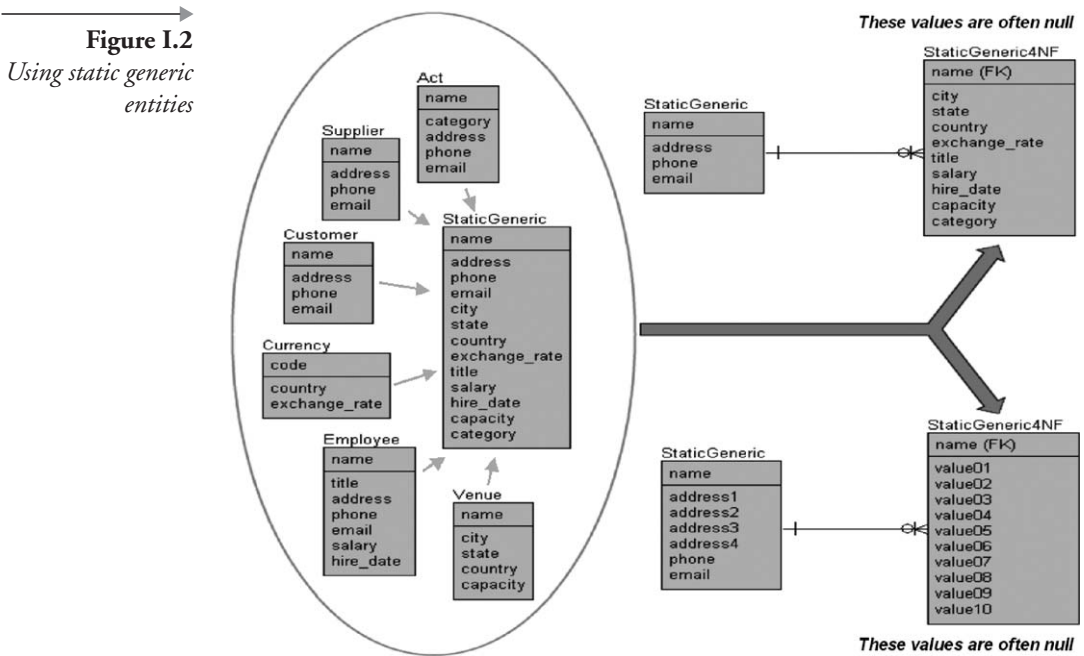
Data Model Tuning

- A data model is used to represent a set of business rules. Business rules are implemented using entities (tables) and the enforcement of relationships between those entities. Additionally, business rules can be implemented using database-encapsulated stored procedures plus event or state change triggers.

Note: Using triggers can cause serious performance problems.

- Business rules can be tuned into a more mathematically correct design using normalization and referential integrity. Referential integrity ensures that relationships between data items are conformed to.
 - Denormalization is the removal of the more granular results of normalization. Granularity causes complex mutable multiple table joins. Multiple table joins can be difficult to tune effectively.
 - Alternate or secondary indexing to cater for SQL code not complying directly with a normalized structure can cause problems. This is common for object applications. This step is part of both the data modeling and the applications coding stages, not either. Alternate indexing is generally enhanced in the applications development stage but should be strictly controlled. Creating too many indexes can cause as many problems as it resolves.
 - Constraints, PL/SQL stored procedures, functions, and event or state change triggers should be tuned for performance in accordance with entities, relationships, and alternate indexing structures. Triggers are best avoided. Database-level coded PL/SQL will perform better in some situations than others. PL/SQL can be compiled into binary, stored in the database as BLOB objects, possibly increasing PL/SQL execution performance. PL/SQL should only cover business rule functionality and database access code, not applications functionality. Business rules sometimes match relational database structure, whereas applications functionality often does not.
 - Implementing referential integrity:
 - *Should referential integrity be implemented?* Not necessarily. Referential integrity can ensure the correctness of data, but it will slow down data changes somewhat because of verification.
 - *How should referential integrity be implemented?* It can be implemented using constraints or triggers. Constraints are the faster and much more reliable method. All sorts of things can go wrong with triggers, and their performance is highly questionable.
 - *Can referential integrity be partially implemented?* Yes, it can. Very small tables containing static, referential data can often have their foreign keys removed. Additionally, noncritical tables or highly used tables can avoid referential integrity to help performance. Generic static tables, when used, probably should avoid referential integrity as well. An example of a generic static table is shown in Figure I.2. These types of tables, in the interests of performance at the database level, are best avoided.
-

- *Where should referential integrity be implemented?* Referential integrity can be enforced in the database or at the application level. The benefit of implementing referential integrity at the database level is simple implementation in one place and one place to change in the future. Developers may not necessarily agree with this philosophy, but database administrators generally would.



SQL Code Tuning

SQL coding requirements should fit the specifications of the data model based on entities, relationships, and alternate indexing. SQL code can be in both database-based PL/SQL coding and applications-embedded SQL code. What are the steps in tuning SQL code?

- Identify the worst-performing SQL code and tune only those SQL statements.
- When the worst-performing code is tuned, SQL code can be tuned in general if there is still a problem.
- Create, remove, and tune alternate indexing to suit SQL code performance requirements without jeopardizing the data model. When

matching SQL code to indexing, it is best to attempt to map SQL code filtering, sorting, and joining to primary and foreign index structures (referential integrity). Using already existing referential integrity keys will reduce the number of alternate indexes. More indexes on tables will speed up data selection but will slow down data updates, sometimes drastically. The fact is that if SQL code is not utilizing referential integrity indexing, there may be a mismatch between the data model and application requirements, or the data model is simply inappropriate.

- How are individual SQL code statements tuned in Oracle Database?
 - Often the most effective method is to examine the SQL code by hand and make sure that the SQL statements conform to indexing in the database, namely in selecting, filtering, sorting, and grouping clauses. The more complex SQL code is, the more difficult it will be to tune in this manner. In fact, complex SQL code is sometimes beyond the capabilities of the optimizer.
 - Use the Oracle Database EXPLAIN PLAN command to examine the optimizer's best execution path for SQL code. EXPLAIN PLAN will show where potential improvement can be made. A simple query plan is shown in Figure I.3.

Figure I.3

*A simple query
plan using
EXPLAIN PLAN*

Query Plan from EXPLAIN PLAN

```
Cost = 102 SELECT STATEMENT on
  HASH JOIN on
    TABLE ACCESS FULL on CUSTOMER
  HASH JOIN on
    TABLE ACCESS FULL on CURRENCY
  HASH JOIN on
  HASH JOIN on
  HASH JOIN on
    TABLE ACCESS FULL on CURRENCY
  INLIST ITERATOR on
    TABLE ACCESS BY INDEX ROWID on INVOICE
  INDEX RANGE SCAN on ADX_INVOICE_1
  TABLE ACCESS FULL on ORDER
  TABLE ACCESS FULL on ORDERLINE
```

- Trace files and TKPROF can be used to tune SQL code, but tracing produces excessive amounts of information. Tracing should be a last-resort method of tuning SQL code.

- Make sure coders use bind variables in both PL/SQL and applications-embedded SQL code. Bind variables are not as significant after Oracle Database 8i Release 1 (8.1.6) because of the cursor-sharing configuration parameter, but forcing cursor sharing lowers statistical accuracy and deteriorates optimizer performance.
- Beware of generic or generated SQL code, which is common at the applications level. This type of embedded SQL code can be very difficult to tune.

Configuration and Physical Tuning

Configuration Tuning

Possibly one of the most effective tuning practices, especially as far as configuration tuning is concerned, is proper initial Oracle installation and configuration of your databases. Do not leave your Oracle installation as Oracle installation software and database creation tools create it. It might be expedient to use the Oracle Database creation tool for a first experimental database creation only.

Note: (10g) The database creation tool (Database Configuration Assistant) has become more sophisticated over time. However, most database administrators still prefer to use scripting because it can be modified at will and executed in parts. This approach is no longer necessary except for highly complex installations.

The database creation tool creates a fairly well-organized physical and configured structure, but its default settings are geared toward very small databases. Take the time to properly organize the database and its configuration initially, and you will be less likely to have large amounts of down-time later on.

Physical Tuning

Physical tuning involves the removal of competition for resources. Physical tuning covers the following areas:

- Physical space usage and proper storage structure usage in terms of how blocks are used and reused. Tuning at the block level depends on the application type.
- Growth management of the database and capacity planning must be monitored.

- Setting files such as log files and rollback segments to sizes and frequencies as appropriate to database usage.

Note: **10g** Manual rollback is deprecated. Use automated undo.

- There can be contention between processes, memory usage, and data. Some of this type of tuning falls within the scope of both configuration and physical tuning.
- I/O contention can be dealt with in the file system by spreading the I/O load using multiple disks or RAID arrays. RAID arrays are most efficiently used with random access on data spaces and sequential access on index spaces. In non-RAID systems, it is sensible to separate data and index spaces because both are read more or less concurrently. Oracle partitioning is a classic example of this type of tuning.

How Is This Book Organized?

This book is broken into four parts: relational data model tuning in Part I, SQL code tuning in Part II, physical and configuration tuning in Part III, and tuning everything at once in Part IV. The object in this book is to cover all aspects of tuning and Oracle installation. Each chapter focuses on a particular aspect of tuning Oracle Database. As you read through each part, each chapter will build on the preceding chapter, digging deeper step by step.

Part I. Data Model Tuning

Chapters 1 and 2 introduce and offer tuning solutions for relational data modeling. Chapter 3 looks at alternative data modeling methodologies, and Chapter 4 provides some historical background.

Chapter 1. The Relational Database Model

Introduces normalization and referential integrity.

Chapter 2. Tuning the Relational Database Model

Tuning using efficient normalization, referential integrity indexing, alternate indexing, denormalization, and some useful tricks.

Chapter 3. Different Forms of the Relational Database Model

This chapter provides a contrast of relational, object, and object-relational concepts.

Chapter 4. A Brief History of Data Modeling

This is a brief history of relational data modeling and the roots of SQL.

Part II. SQL Code Tuning

The format in Part II is to first describe problems and then offer solutions and tools to help find those solutions. Chapters 5 to 7 present SQL and describe how to tune SQL code without having to dig into the depths of the Oracle optimizer. Chapter 8 looks at indexing. Chapters 9 to 11 dig into the Oracle optimizer, statistics, and hints. Chapters 12 and 13 present tools used to help find SQL code performance problems. Solving as opposed to finding SQL code problems is in preceding chapters of Part II.

Chapter 5. What Is SQL?

A brief description of SQL and the basics that Oracle SQL can accomplish. Everything is explained in detail and using practical examples.

Chapter 6. The Basics of Efficient SQL

This chapter is the most important chapter in Part II: SQL Code Tuning. This chapter will teach you step by step how to write properly tuned SQL code. Robust realistic data sets and Oracle Database schemas are used to prove all aspects of tuning. Some tables in schemas are in excess of 1 million rows. There is plenty of data to tune with.

Note: The number of rows in tables varies as the book progresses.

Chapter 7. Advanced Concepts of Efficient SQL

This chapter expands on the details learned in Chapter 6, by introducing more complex concepts, such as joins and subqueries.

Chapter 8. Common-Sense Indexing

Teaches how to build indexes properly, building on tuning of SQL code taught in preceding chapters. Indexing and index types

are analyzed, explained, and compared for speed in relation to various database activities.

Chapter 9. Oracle SQL Optimization and Statistics

Covers very deep-level tuning of SQL code in Oracle with an emphasis on using the optimizer, without wasting time on too many nitty-gritty details. This chapter describes how the Oracle SQL code optimizer functions and what can be done to influence the optimizer. If you have read, understood, and implemented the advice and pointers made in previous chapters, you may never need to understand the optimizer on this level.

Chapter 10. How Oracle SQL Optimization Works

This chapter examines and explains the ways in which Oracle Database internally accesses data. An understanding of the various access methods will help when deciphering query plans.

Chapter 11. Overriding Optimizer Behavior Using Hints

This chapter examines and explains the ways in which Oracle Database internally accesses data. An understanding of the various access methods will help when deciphering query plans.

Chapter 12. How to Find Problem Queries

Describes the ins and outs of optimizer query plans using EXPLAIN PLAN, Tracing, and TKPROF. Additional details covering Oracle V\$ performance views for SQL code tuning are also provided.

Chapter 13. Automated SQL Tuning

A brief picture of automated SQL code tuning tools, contained within the Database Control.

Part III. Physical and Configuration Tuning

As in Part II, the format in Part III is to first describe problems and then offer solutions and tools to help find those solutions. Part III describes how to build cleanly structured Oracle installations.

Chapter 14. Tuning Oracle Database File Structures

Tuning Oracle Database file structures encompasses both the physical and logical layers of tuning from the Oracle Instance and

Oracle Database file system layers through to logical layers and tablespaces.

Chapter 15. Object Tuning

Object tuning covers database objects such as tables, indexes, sequences, and synonyms. Numerous tweaks can be made to various database objects to tune for performance.

Chapter 16. Low-Level Physical Tuning

This chapter covers physical block and extent structures and how to build a physical structure from the ground up.

Chapter 17. Hardware Resource Usage Tuning

Hardware resource usage tuning is introduced, covering tuning CPU usage, Oracle Database memory cache buffers, and finally tuning of I/O activity.

Chapter 18. Tuning Network Usage

This chapter covers the Listener, network naming methods, and shared server configuration. There are numerous ways in which Oracle network connectivity can be tuned for better performance, depending on requirements.

Chapter 19. Oracle Partitioning and Parallelism

Oracle Partitioning allows for breaking up of large tables into separate objects, which can have SQL code executed on them in parallel for excellent performance improvements.

Part IV. Tuning Everything at Once

This is where you get to see all aspects of tuning put together and into practice.

Chapter 20. Ratios: Possible Symptoms of Problems

Traditionally, Oracle Database practitioners use large numbers of ratios to assess and provide pointers as to what and where to tune. The secret is to not tune the ratios themselves. Use the ratios as indicators of and pointers to potential problems.

Chapter 21. Wait Events

A wait event occurs when a process or session is forced to wait for another to finish using something both processes require.