

Introduction

This is a book that relates my personal experiences and knowledge about building scalable and performant applications that make use of an Oracle database. It has been my longstanding experience that applications dependent on the database for processing – and today that is pretty much every application of note – succeed or fail based on how they use the database.

There are many schools of thought out there today that describe how to build applications that will scale and be performant but the one thing they all have in common is they use a database to store and process data and, if you strip away the GUI at the end of the day – it is really about data and processes accessing and using that data. Therefore, it is my premise that regardless of the tools you use, the approaches you take to implementing your application – sound and proper use of the database from day one will increase your odds of success many times over.

I am not going to promote the use of one development technique over another but rather will stress that database design and implementation and the choices you make as to what features to use in the database versus do yourself outside of the database in the application will affect you.

Who should use this book?

The target audience of this book is the development TEAM – the group of people that have 100% of the control over the overall performance of the system. This is contrary to the popular myth that the DBA is solely responsible and has total control over application performance inside of the database. The best way to understand that this is a myth would be to use a car racing analogy. The DBA is the pit stop guy who changes the tires, makes sure the engine is gassed up, that the car functions. If you give the pit stop guy (DBA) a Lincoln Navigator (a truly huge truck) and tell them to race the Indy 500 with it – what'll happen? The DBA can make sure the truck runs as fast as it can but he cannot affect the performance of the truck on a tight corner at 100+ miles per hour. There is really very little to be done after the car has been designed and built (short of throwing out the car and starting over – the car is the application here). That analogy is frightening in its applicability to so many systems. A Lincoln Navigator was built where a high performance Indy car was needed **or vice versa**. Remember – we need 18 wheelers, minivans, race cars – everything. There is no one size fits all here.

This book is for the Development team that is not quite sure how to approach this seemingly daunting task we call “performance tuning”. This is **not** a beginners book, this book is for the developer and DBA who already knows how to enter SQL statements, use SQL*Plus and so on. It will not teach you SQL, it will teach you how to write “good” SQL by making you aware of the options you have – the tools you can use.

I will use yet another analogy to describe how this book will present information. Pretend for a moment that the developer is instead a medical doctor and the application is the patient. There are many types of MD's:

- **The emergency room doctor.** They do “triage” – separating the hopeless from the ones that can be helped. Performing quick fixes to keep patients alive for as long as possible. They strive for short term band-aids to fix up the patient. They will take a patient with a heart attack induced by smoking, bad diet and no exercise and get them stabilized.
- **The operating room doctor.** They get the patient after the ER doctor has triaged them and patched them up. They strive for long term fixes to keep the patient not only alive but as fully functioning as possible. They perform the by-pass operation on that heart attack attempting to clear the arteries.
- **The physical therapist.** They get the patient after the operating room doctor is finished and begin a long and painful (not to mention expensive) process of rehabilitation.
- **The preventative medicine doctor.** They strive to avoid the above three doctors at all costs. They counsel the patient to quit smoking, eat a healthy diet, and exercise – developing a phased plan to get

them in shape. If they do their job right – with the exception of unfortunate accidents (like a car accident), the patient will never see the ER, OR or PT doctors.

Now, the world needs all types of doctors – accidents do happen after all. But one of the most important types of doctors is that last one, the preventative medicine doctor. The one that tries hard to avoid having their patient need the other three.

It is my belief (experience) that most people and books approach tuning using the mindset of the first three doctor types above. They are in support of the **hero Developer/DBA**; e.g. the ER or OR doctor. These Developers/DBAs seem to get all of the fame as they snatch the patient from the grasp of death (save the system by doing something miraculous). They get called in at the last moment, work horribly hard for an extended period of time trying to keep the patient alive (and get paid handsomely as well). The physical therapist are the unlucky soles that get the system after the ER/OR doctor has patched it up. They are the ones responsible for keeping this system going.

I feel I am well equipped to speak from that perspective. I am in fact one of those “heros”. I am called in to “lay hands on” systems and make them better. I could write that book – I won’t.

What I propose is a book that acts as a mentor to the reader, providing an overall structure and approach to performance tuning:

- Start tuning before you start designing
- Design with specific performance goals in mind and continuously test towards them
- Try things, see how they work, don’t assume that how you think they “should” work is in fact the way they work (many performance issues are directly related to not understanding how the database software works in the first place)
- Since accidents do happen, as they do in real life, we’ll also take a look at being that ER or OR doctor as well although this will definitely not be the end goal, more of a “just in case”. The good news is that 100% of the knowledge you learn being that preventative medicine doctor will apply here.

How this book is structured

This book will present all of its information in the same fashion. I will make a claim, I will develop and present quantitative proof that it is true and I will describe why it is relevant (the claim). I will not use terms such as “In my opinion”, “I feel”, “I think”, and so on. Rather, I will present facts, facts that can be proven quantitatively via testing or benchmarking. That will leave no doubt as to the veracity of any claims made in this book.

Outline

Introduction

Setting Up

1. The Right Approach to Building Applications
 - 1.1. It’s a Team Effort
 - 1.1.1. DBA and Developer Roles
 - 1.1.1.1. DBA Do’s and Don’ts
 - 1.1.1.2. Developer Do’s and Don’ts
 - 1.2. Read the Documentation
 - 1.2.1. A Guide to the Guides
 - 1.2.1.1. The Concepts Guide

ISBN: 0-07-223605-7

2-9

This material is for review only and is subject to change without prior notice.
Copyright © 2003 by the McGraw-Hill Companies, Inc. (Publisher). All rights reserved.

- 1.2.1.2. New Features Guide
- 1.2.1.3. Application Developers Guide
- 1.2.1.4. PL/SQL Users Guide and Reference
- 1.2.1.5. Performance Tuning Guide and Reference
- 1.2.1.6. Backup and Recovery Concepts
- 1.2.1.7. Recovery Manager Reference
- 1.2.1.8. Administrators Guide
- 1.2.2. Road Maps to Reading
 - 1.2.2.1. Required Reading for Both Developers and DBAs
 - 1.2.2.2. Required Reading for Developers
 - 1.2.2.3. Required Reading for DBAs
 - 1.2.2.4. Recommended Reading
- 1.3. Avoid the Black Box Syndrome
 - 1.3.1. Database Independence versus Database Dependence
 - 1.3.2. Dangers of Block Box Syndrome
 - 1.3.2.1. Inability to Perform
 - 1.3.2.2. Inability to Get the Correct Answer
 - 1.3.2.3. Inability to Quickly Deliver Software
 - 1.3.2.4. Inability to Maximize Your Investment
- 1.4. It's a Database, Not a Data Dump
 - 1.4.1. Use Primary and Foreign Keys
 - 1.4.2. Test the Overhead of Referential Integrity
 - 1.4.3. Middle Tier Checking Is Not a Panacea
 - 1.4.3.1. Is It Faster?
 - 1.4.3.2. Is It More Flexible?
 - 1.4.3.3. Is It Database-Independent?
 - 1.4.3.4. Is It More Secure?
- 1.5. Build a Test Environment
 - 1.5.1. Test Against Representative Data
 - 1.5.1.1. Consider Using Partition Elimination
 - 1.5.1.2. Know the Optimizer Will Change Query Plans over Time
 - 1.5.2. Don't Test with a Single User
 - 1.5.3. Don't Test in a Dust-Free Lab
- 1.6. Design to Perform; Don't Tune to Perform
 - 1.6.1. Don't Use Generic Data Models
 - 1.6.2. Design Your Data Model for Efficiency
 - 1.6.2.1. Design for Writes or Reads?
 - 1.6.2.2. Which Data Model Worked?
- 1.7. Define Your Performance Goals from the Start
 - 1.7.1. Work to Clear, Specific Metrics
 - 1.7.2. Collect and Log Metrics over Time
 - 1.7.3. Don't Do It Because "Everyone Knows You Should"
- 1.8. Benchmark, Benchmark, Benchmark
 - 1.8.1. Small-Time Benchmarking
 - 1.8.1.1. Use a Simple Test Harness
 - 1.8.1.2. Other Benchmarking Tools
 - 1.8.2. Big-Time Benchmarking
 - 1.8.2.1. Test with Representative Amounts of Data
 - 1.8.2.2. Test with Realistic Inputs
 - 1.8.2.3. Make Sure You Verify the Results
 - 1.8.2.4. Don't View Benchmarking As a Chore
- 1.9. Instrument the System
 - 1.9.1. Trace from asktom.oracle.com
 - 1.9.2. Instrument for Remote Debugging

- 1.9.3. Use DBMS_APPLICATION_INFO Everywhere
- 1.9.4. Use DEBUG.F in PL/SQL
- 1.9.5. Turn on SQL_TRACE in Your Application
- 1.9.6. Use Industry-Standard APIs
- 1.9.7. Build Your Own Routines
- 1.9.8. Audit Is Not a Four-Letter Word
- 1.10. Question Authority
 - 1.10.1. Beware of Universal “Bests”
 - 1.10.2. Suspect Ratios and Other Myths
- 1.11. Don’t Look for Shortcuts
- 1.12. Keep It Simple
 - 1.12.1. Consider Alternate Approaches
 - 1.12.2. Let the Database Do What It Does Best
- 1.13. Use Supplied Functionality
 - 1.13.1. We Heard Feature X Is Slow
 - 1.13.2. We Heard Feature X Is Complicated
 - 1.13.3. We Don’t Want to
 - 1.13.4. We Didn’t Know
 - 1.13.5. We Want Database Independence
- 1.14. Summary
- 2. The Tools You’ll Want
 - 2.1. SQLPlus
 - 2.2. Setup
 - 2.3. SQLPlus tips
 - 2.3.1. Use a login.sql
 - 2.3.2. Use @connect
 - 2.3.3. Use SQLPATH
 - 2.3.4. Read the documentation!
 - 2.4. Explain Plan
 - 2.4.1. Setup
 - 2.4.2. Using Explain Plan
 - 2.4.3. Reading an Explain Plan
 - 2.4.4. The Explain Plan ‘trap’
 - 2.4.5. DBMS_XPLAN and V\$SQL_PLAN
 - 2.5. Autotrace
 - 2.5.1. Setting up for AUTOTRACE
 - 2.5.2. Using AUTOTRACE
 - 2.5.3. Formatting the output
 - 2.5.4. Understanding the output
 - 2.5.5. So, what are you looking for?
 - 2.5.5.1. Recursive SQL
 - 2.5.5.2. DB block gets/Consistent Gets
 - 2.5.5.3. Physical Reads
 - 2.5.5.4. Redo Size
 - 2.5.5.5. SQL*Net statistics
 - 2.5.5.6. Sorts and Rows processed
 - 2.5.6. Autotrace Summary
 - 2.6. TKPROF
 - 2.6.1. Enabling TKPROF
 - 2.6.2. Running TKPROF
 - 2.6.3. TKPROF for the masses
 - 2.7. Runstats
 - 2.7.1. Setting up
 - 2.7.2. Using Runstats

- 2.8. Statspack
 - 2.8.1. Setting up Statspack
 - 2.8.2. Using Statspack
 - 2.8.3. What people do wrong
 - 2.8.4. Statspack at a glance
- 2.9. DBMS_PROFILER
 - 2.9.1. Why you want to use the profiler
 - 2.9.2. Getting Practical Information on the Profiler
- 2.10. Jdeveloper (and debugging)
- 2.11. Summary
- 3. Scalable Architecture
 - 3.1. Understand Shared Server Versus Dedicated Server Connections
 - 3.1.1. How Do Dedicated Server Connections Work?
 - 3.1.1.1. Dedicated Server Connection Steps
 - 3.1.1.2. Pros and Cons of Dedicated Server Connections
 - 3.1.2. How Do Shared Server Connections Work?
 - 3.1.2.1. Shared Server Connection Steps
 - 3.1.2.2. Shared Server Command Processing
 - 3.1.2.3. Pros and Cons of Shared Server Connections
 - 3.1.3. Common Misconceptions about Shared Server Connections
 - 3.1.3.1. Shared Server Uses Significantly Less Memory
 - 3.1.3.2. You Should Always Use Shared Server Configuration with Application Servers
 - 3.1.3.3. Shared Servers Are Only for Client/Server Applications
 - 3.1.3.4. An Instance Runs in Shared Server Mode (or Not)
 - 3.1.4. Dedicated Server Versus Shared Server Wrap-up
 - 3.2. Take Advantage of Clustering
 - 3.2.1. How Does RAC Work?
 - 3.2.1.1. An Instance versus Database Refresher
 - 3.2.1.2. Instances in RAC
 - 3.2.1.3. A Service Refresher
 - 3.2.2. What Are the Benefits of RAC?
 - 3.2.3. Clustering Wrap-up
 - 3.3. Use Locally Managed Tablespaces
 - 3.3.1. Why Are DMTs Obsolete?
 - 3.3.2. Use System-Managed LMTs When You Do Not Know How Big Your Objects Will Become
 - 3.3.3. Use Uniform Extent Sizes When You Know the Ultimate Size of an Object
 - 3.3.4. Some LMT Caveats
 - 3.3.4.1. The Magic Number for Uniformly Sized Extents Is 64KB
 - 3.3.4.2. System-Managed LMT Allocates from Files Differently
 - 3.3.4.3. How to Autoextend Datafiles
 - 3.3.4.4. Beware of Legacy Storage Clauses!
 - 3.3.5. LMT and DMT Wrap-up
 - 3.4. Know When to Use Partitioning
 - 3.4.1. Partitioning Concepts
 - 3.4.1.1. Partitioning Schemes
 - 3.4.1.2. Partition Pruning
 - 3.4.2. The Partitioning Myth
 - 3.4.3. Why Use Partitioning?
 - 3.4.3.1. Increased Availability
 - 3.4.3.2. Easier Administration
 - 3.4.3.3. Increased Performance
 - 3.4.4. Partitioning Wrap-up
 - 3.5. Know When to Use Parallel Operations

- 3.5.1. The Parallel Myth
 - 3.5.1.1. Parallel Processing Overhead
 - 3.5.1.2. Parallel Processing Scalability
- 3.5.2. Parallel Administration
- 3.5.3. Parallel Query
 - 3.5.3.1. Parallel Query Settings
 - 3.5.3.2. When Are Parallel Queries Useful?
- 3.5.4. Parallel DML
- 3.5.5. DIY Parallelism
- 3.5.6. Parallel Processing Wrap-up
- 3.6. Summary
- 4. Effective Administration
 - 4.1. Use SPFILES
 - 4.1.1. Are PFILES Obsolete?
 - 4.1.2. Help, My SPFILE Is Broken and I Cannot Start
 - 4.1.3. How Can I Tell Whether I'm Using an SPFILE
 - 4.1.4. Pros and Cons
 - 4.2. Use Oracle Managed Files
 - 4.2.1. Pros and Cons
 - 4.3. Bullet Proof Your Backups
 - 4.3.1. Virtually all Systems Must Be in Archivelog Mode
 - 4.3.2. Archive Logs Must Be Moved/Copied off of the Source System
 - 4.3.3. You Must Keep at Least Two, If Not More, Copies of Your Backups
 - 4.3.4. The Physical Backups Must Be Tested
 - 4.3.5. Restoring Must Be Practiced
 - 4.3.6. Use Tools to Reduce the Chances of Error
 - 4.3.7. Data Guard
 - 4.3.8. Backup and Recovery Summary
 - 4.4. Give Automatic Segment Space Management a Try
 - 4.4.1. Freelists and Freelist Groups
 - 4.4.2. Pctfree and Pctused
 - 4.4.3. The Case for ASSM
 - 4.5. Use Automatic UNDO Management for Operational Simplicity
 - 4.5.1. Setting the UNDO_RETENTION
 - 4.5.2. UNDO tablespace caveat
 - 4.5.3. UNDO Tablespace Summary
 - 4.6. Administrative Efficiency Summary
- 5. Statement Processing
 - 5.1. Understand the Types of SQL Statements
 - 5.2. How Are Statements Executed?
 - 5.2.1. Parsing
 - 5.2.1.1. Syntax and Semantic Checks
 - 5.2.1.2. Shared Pool Check
 - 5.2.1.2.1. Checking for Symantec Match
 - 5.2.1.2.2. Checking for Environment Match
 - 5.2.1.2.3. Parsing Wrap-up
 - 5.2.2. Optimizing and Row-Source Generation
 - 5.2.3. Execution
 - 5.2.4. Statement Execution Wrap-up
 - 5.3. Queries from Start to Finish
 - 5.3.1. A Quick-Return Query
 - 5.3.2. A Slow-Return Query
 - 5.3.3. Consistent Reads

- 5.4. Modification DML from Start to Finish
- 5.5. DDL Processing
- 5.6. Using Bind Variables
 - 5.6.1. What Are Bind Variables?
 - 5.6.2. Advantages of Using Bind Variables
 - 5.6.2.1. Without Bind Variables, Performance will Suffer
 - 5.6.2.2. Without Bind Variables, Your System will not Scale
 - 5.6.2.3. Without Bind Variables Your Code is Harder to Write
 - 5.6.2.4. Without Bind Variables, Your code is less Secure
 - 5.6.3. Bind Variables and Java
 - 5.6.4. Summary: As a General rule, Use bind variables
 - 5.6.5. There Are Exceptions to Every Rule
 - 5.6.5.1. Queries per Second Systems
 - 5.6.5.2. Seconds per Query Systems
- 5.7. Parse as Little as Possible
 - 5.7.1. How to Reduce Your Parses
 - 5.7.1.1. Use PLSQL
 - 5.7.1.2. Watch out for Dynamic SQL in PLSQL
 - 5.7.1.3. Move SQL out of Triggers
 - 5.7.1.4. Prepare ONCE, Execute Many
- 5.8. Summary
- 6. Getting the Most out of the Cost Based Optimizer
 - 6.1. Why the RBO Is Dead
 - 6.2. Making the CBO Work Right
 - 6.2.1. OPTIMIZER_INDEX_CACHING
 - 6.2.2. OPTIMIZER_INDEX_COST_ADJUST
 - 6.2.3. Using SYSTEM Statistics
 - 6.3. Optimizing the CBO
 - 6.3.1. Cursor_sharing
 - 6.3.2. Compatible
 - 6.3.3. Db_file_multiblock_read_count
 - 6.3.4. Hash_join_enabled
 - 6.3.5. Optimizer_dynamic_sampling
 - 6.3.6. optimizer_features_enable
 - 6.3.7. optimizer_max_permutations
 - 6.3.8. optimizer_mode
 - 6.3.8.1. CHOOSE
 - 6.3.8.2. RULE
 - 6.3.8.3. ALL_ROWS and FIRST_ROWS
 - 6.3.9. query_rewrite_enabled, query_rewrite_integrity
 - 6.3.10. Bitmap_merge_area_size, sort_area_size, Hash_area_size
 - 6.3.11. star_transformation_enabled
 - 6.3.12. Others Parameters that We Have Covered
 - 6.4. The 10053 Event
 - 6.5. Summary
- 7. Effective Schema Design
 - 7.1. Assumed Knowledge
 - 7.2. Fundamental Schema Design Principles
 - 7.2.1. Let the database do your work (RI)
 - 7.2.2. Use the correct datatype
 - 7.2.3. Optimize to your most frequently asked questions
 - 7.3. Choosing Tables and Indexes
 - 7.4. Overview of Table Types

- 7.5. Clusters
 - 7.5.1. Creating Clusters
 - 7.5.1.1. B*Tree Clusters
 - 7.5.1.2. Hash Clusters
 - 7.5.2. Using Clusters
 - 7.5.2.1. Clusters are Effective when you can Control how the data is Loaded
 - 7.5.2.2. B*Tree Clusters can Reduce IO and Increase Buffer Cache Efficiency
 - 7.5.2.3. Hash Clusters Eliminate Index Blocks Altogether
 - 7.5.2.4. Single Table Hash Clusters are Useful for Read only/Lookup Tables
 - 7.5.3. Clusters Summary – Pros and Cons
 - 7.6. Index Organized Tables (IOT)
 - 7.6.1. Using IOTs
 - 7.6.1.1. IOTs are a space-saving Alternative for Association Tables
 - 7.6.1.2. IOTs excel at collocating randomly inserted data
 - 7.6.2. IOT's pros and cons
 - 7.7. External Tables
 - 7.7.1. Use External Tables to load and Merge Data
 - 7.7.1.1. Setting up external tables
 - 7.7.1.2. Direct Path Loading
 - 7.7.1.3. Parallel Direct Path Loading
 - 7.7.1.4. Merging
 - 7.7.1.5. Handling/Detecting Errors
 - 7.8. Overview of Index Types
 - 7.9. Function Based Indexes
 - 7.9.1. Using FBIs – Thinking outside the box
 - 7.9.1.1. Indexing selectively
 - 7.9.1.2. For complex constraints
 - 7.10. Domain Indexes
 - 7.10.1. Using Domain Indexes
 - 7.10.1.1. Domain Indexes can implement third-party Indexing Techniques
 - 7.10.1.2. Creating "Locator" Indexes
 - 7.11. Compression
 - 7.11.1. Index Key Compression
 - 7.11.2. Table Compression
 - 7.11.3. Compression Summary
 - 7.12. Summary
- 8. Efficient SQL
 - 8.1. Understanding Access Paths
 - 8.1.1. Full Scans
 - 8.1.2. Rowid Access
 - 8.1.3. Index Scans
 - 8.1.4. Cluster Scans
 - 8.2. Understanding Joins
 - 8.2.1. Nested Loop
 - 8.2.2. Hash Joins
 - 8.2.3. Sort Merge
 - 8.2.4. Cartesian
 - 8.2.5. Anti-Joins
 - 8.2.6. Full Outer Joins
 - 8.3. The Schema Matters (Physical)
 - 8.4. Really Know SQL
 - 8.4.1. Rownum
 - 8.4.2. Scalar Subqueries
 - 8.4.3. Analytics

- 8.4.4. Materialized Views
- 8.5. Don't Tune a Query
- 8.6. Summary
- 9. Effective PL/SQL Programming
 - 9.1. Why PL/SQL?
 - 9.1.1. PL/SQL Is the Most Efficient Language for Data Manipulation
 - 9.1.2. PL/SQL is Portable and Reusable
 - 9.1.3. Write As Little As You Can
 - 9.1.4. Use Packages
 - 9.1.5. Use Static SQL
 - 9.1.6. Bulk Processing
 - 9.1.7. Returning Data
 - 9.1.8. Use %TYPE, %ROWTYPE
 - 9.1.9. Authid current_user
 - 9.1.10. Looking It Up
 - 9.1.11. Autonomous Transactions
 - 9.1.12. Implicit/Explicit Cursors
 - 9.1.12.1. Single Row Selects
 - 9.1.12.2. Result Sets with a Limited Number of Rows
 - 9.1.12.3. Result Sets with Large Numbers of Rows
 - 9.1.12.4. Dynamic SQL
 - 9.1.12.5. Summary on Implicit/Explicit Cursors
 - 9.2. Summary
- 10. So you had an accident
 - 10.1. What is different from your testbed
 - 10.2. Don't panic
 - 10.3. Change one thing at a time
 - 10.4. Have a sound reason for changing that one thing
 - 10.5. Never never never change something "because", don't guess
 - 10.6. Un-change things
 - 10.7. Start with the application – TKPROF
 - 10.8. Then the system – statspack
 - 10.9. Then the OS – os dependent
 - 10.10. Compare to your history – something changed
 - 10.11. Building a test case (for support, for yourself)

Appendix

A. Case Study