The Low Administration Oracle Specification (LAOS), Part I: Avoiding the Need for Administration

Version 1.6

Revised: 15-Mar-1999

Author: Thomas B. Cox

Purpose: This document describes the steps to configure an Oracle database for a near-zero maintenance environment on Windows NT. (Many of these guidelines can be used on other operating system platforms.)

Acknowledgements: This paper was inspired by the work of Craig Shallahammer and Cary Millsap, among others (see References), and has been influenced by many of my co-workers over the last several years. I am especially pleased that Automatic Data Processing, Inc. implemented the first (to my knowledge) Low Admin commercial product, ADP PC/Payroll for Windows, perfecting many of the same ideas found in this white paper, and I am indebted to Inna Brovman of ADP for her faith in the LAOS ideal.

Part II is not currently available. Please check back at http://home.att.net/~tbcox/ for updates.

Please send your corrections, suggestions, and feedback to me at the address below, with your return address so I may credit your contribution. Thank you.

-Thomas B. Cox
tbcox@worldnet.att.net

LAOS describes the steps to configure an Oracle database for a near-zero maintenance environment on Windows NT. (Many of these guidelines can be used on other operating system platforms.)

Scope

This specification is for reducing the overall administrative overhead required to keep a database running in production. The goal is to allow some databases to run for years without active administration (other than normal backups and occasional recovery).

Several sorts of databases are candidates for LAOS implementations. First, any database that will be geographically remote, isolated, or located away from professional database administrators (databases run by service bureau clients; databases on aircraft or ships; or departmental servers managed by central staff). Second, small databases used for control or monitoring of other larger databases (these can include Enterprise Manager repositories, RMAN Server Managed Backup and Recovery repositories, home-grown capacity planning databases, etc.). Third, embedded databases that have extremely controlled and delimited functions and that may have no direct human interaction.

First we review the tasks required to keep a database alive in a positive-maintenance environment. Then we examine which tasks can be eliminated, how to do so, and what assumptions this makes about the target database and the application running on it. For those tasks that may be eliminated in more than one way, we examine the design trade-offs between the available options. (A separate document, LAOS Part II, describes how to automate those tasks that Part I couldn't eliminate.)

Tasks Throughout the Database Life Cycle

Below is an overview of the live cycle of a database, from the initial business analysis, through creation of an application, on to maturity, and changes to the database after it goes production.

Database Life Cycle Stage:	**Architect**	**Data Administrator**	**DBA**
1. Strategy, Analysis	Table Design, Security Design	Data Ownership, Security Policy	
2. Design, Build	Impact Analysis	Change Management	Installation, Datafile Creation, Table-to-Tablespace Mapping, Table Creation, Backup/Recovery Planning, Security Procedures
3. Youth			Physical Reorganization, Performance Tuning
4. Maturity			Exception Monitoring, Performance Monitoring, Performance Tuning, Backup/Recovery Execution, Datafile Management
A. Minor Change	Impact Analysis	Change Management	Table Modification
B. Major Change	Impact Analysis, Table Re-design	Change Management	Table Creation

The Low Maintenance issue can be seen as eliminating, simplifying, or automating each task in the 'Maturity' section above.

Tasks to Support a Mature Oracle7 Database

Below is a list of all the tasks that a DBA would normally perform on a mature, production database, when each task is performed, and steps to eliminate the task or minimize it.

Task	When	How to Avoid or Minimize
1. Backing up	Depends on application profile; anywhere from weekly to daily	Should only be eliminated for a read-only database; even then, the ability to restore the data should still exist; *automate the backup function*
2. Recovering a Database	Disk failure; user error requiring recovery of earlier data	mirror disks; limit user power; train users
3. Datafile Management	Data outgrows existing tablespaces; disk hot spots require datafiles be moved to different disks to balance I/O	1. use auto-extending datafiles, or, pre-allocate space so data won't outgrow space available 2. keep user load low or allocate enough hardware resources so disk I/O is not a performance issue 3. put all user tables in a single tablespace with a single datafile 4. use OFA
4. Exception Monitoring	daily; examine logs for errors	May be eliminated for a well behaved packaged application, provided the database is not used for any other work (especially not development) [See Part II on how to automate this.]
5. Performance Monitoring	To proactively tell if key tests of database performance are falling; allows corrective action before users notice slowdown	keep user load low or allocate enough hardware resources so performance is not an issue or manage user expectations about performance under untested conditions (i.e. workloads beyond what the system is configured to support); *automate sampling of performance times of key queries* [See Part II on how to automate this.]
6. Performance Tuning	If performance drops below user acceptable levels, or when user expectations exceed observed performance	research how the growth of data affects the given database; create canned process to rebuild unbalanced indexes (rare); keep work load below maximum hardware capability; change database configuration as workload changes
7. Physical Reorganization	Fragmentation is high, or chaining is high, or data objects in a given tablespace need to be separated	don't place an index in the same tablespace as its table; follow OFA guidelines on object placement; size tables so chaining is minimized; *pre-allocate each table with enough space so it should never extend; use a standard NEXT extent size for a given tablespace, and PCTINCREASE of zero, and unlimited extents* [See Part II on how to automate this.]

Eliminating, Simplifying, and Automating Tasks

Assumptions made in this paper:

1. The database in question is Oracle release 7.3 or later, including Oracle8.
2. The database is running on the Windows NT operating system.
3. There is *NOT* a professional DBA on hand to manage the database.
4. The database is *NOT* running an application that is both mission critical and in need of high availability – i.e. we assume the database can be down as much as 3 days a year.
5. The Export utility for logical backups will not be used. Only physical backups (or RMAN backups) will take place.

Backing Up

All backups come in two types: physical and logical. A logical backup of a database consists of one or more files that contain data from the database, usually in a file format useful for moving data between databases; the Export and Import programs create and use logical backup files. A physical backup of a database consists of making a copy of the database files; it must be possible to use this copy to re-create the database.

We would only want to re-create the database if it were somehow damaged; this re-creation process is described in the next section, Recovering. The task of Backing Up is not a candidate for elimination. It is, however, a candidate for simplification and automation.

We assume the database is running in NOARCHIVELOG mode, and that only Cold physical backups will be performed. A version of LAOS for databases running in ARCHIVELOG mode is still being worked on for Part II of this paper. If you need this, or if you think you need logical backups of your LAOS-enabled database, please contact the author.

Simplifying Backup

Physical Backup

An Oracle7 database requires several files: at least one control file; at least two log files; and at least one data file. It will also have an INIT.ORA file. A successful backup requires, at the very least, the copying of all the data files and one control file, and should also contain a copy of the INIT.ORA file. For a cold backup, it also requires that the database be shut down using either 'Shutdown Normal' or 'Shutdown Immediate'. The 'Shutdown Abort' command does not synchronize the data files with the control files, so a backup of the data files right after a 'Shutdown Abort' is useless – it cannot be restored from. (If you ever have to perform a Shutdown Abort, then you should right away do a Startup Restrict and then a Shutdown Immediate.)

Logical Backup: Export

Oracle7 also allows 'logical' backups of data. This is done using the Export utility. This is typically useful when one wishes to:

1. reorganize a table that takes up multiple extents into a single extent;
2. allow for restoring data in a single user table (rather than restoring the entire database), i.e. recovering from user error;
3. migrate database objects from one user's schema into another user's schema
4. move data between databases without using database links

Since we assume that we do not want to perform the above tasks for a mature database, then *the use of Export for logical backups can be eliminated.*

Oracle8 Hot Backup: RMAN

Oracle8 allows the use of the RMAN utility to back up data. This is typically useful when one:

1. is managing multiple databases;
2. is willing to master the complexities of RMAN;
3. must run in ARCHIVELOG mode;
4. is willing to support an RMAN repository database in addition to all other databases.

Use of RMAN may be discussed in a future version of Part II of this paper.

Recovering

Several things can cause a need to recover the database from backup files. These include disk failure and user error.

Avoiding Recovery

To avoid recovery, avoid the errors that lead to it: disk failure and serious user mistakes. A simple (if somewhat expensive) way to prevent disk failure from harming the database is to mirror all disks containing database files. If mirroring (i.e. RAID 1, or better yet RAID 1+0) is too expensive, then RAID 5 can be used in a pinch. Any important database that needs to stay up a lot and that cannot tolerate much data loss should be run on at least RAID 5.

To avoid the sort of serious user mistakes – e.g. deleting the master price list – that would force you to have to recover from a backup, you should: use database security; train users carefully; and write applications so as to prevent a user from erroneously trashing data. (Some DBAs use Export as a means to recover from certain user errors.)

Automating Recovery

To automate recovery, make sure you have locked down the backup process, and make sure that backups are done (a) regularly, (b) in the prescribed fashion, and (c) preferably using an automated backup process.

A simple automated recovery process would perform these steps:

1. Shut down the database
2. Copy the backed up data files back to their original locations, overwriting the ‘current’ data files; also copy the backup control file so that it overwrites the ‘current’ control file
3. Start up the database

Note that, if the most recent backup took place after the database was damaged, then this ‘restore’ would not fix the problem.

More on automating Recovery can be found in Part II.

Avoiding Datafile Management

Use auto-extending datafiles, or pre-allocate space (for both objects and datafiles) so that normal growth of data won’t exceed available storage. Pre-configure datafiles on disks by following OFA guidelines [Millsap, 1994] to avoid disk contention and the subsequent need to shuffle datafiles around. Part of OFA says “make your datafiles a consistent size”. Doing this makes it vastly easier to re-locate datafiles if this should become needful.

Developers will have to know exactly which database objects will be used simultaneously by frequently executed transactions, i.e. table A will be updated along with index B, master table M will be read along with detail table D, etc., in order to place these objects on different disks. Developers should then test at various system loads to verify

that the configuration does in fact minimize disk contention. This is a standard part of I/O tuning and will not be dealt with in depth here.

Suffice it to say that if you don't know in advance what your application's I/O profile is, i.e. if you don't know where to put datafiles on disks so as to minimize disk contention, then your application is not mature and you aren't ready to create a low-admin database.

Monitoring Exceptions

Write a batch file to scan the alert.log file for ORA- errors. Write your application to handle exceptions well, and report serious exceptions to a central monitoring file or table. See LAOS Part II for guidelines on automating Exception Monitoring.

To minimize Exception Monitoring, first define levels of seriousness for different exceptions, and allow monitoring to be done at different levels, similar to the Trace Levels of SQL*Net and Net8. If too much information is gathered about a host of trivial exceptions, important ones could get lost in the pile.

Monitoring Performance

Write sample queries that are representative of real user work. Run these queries weekly and time them. (If performance begins to fall off seriously, you may have failed to configure the system well enough to avoid having to tune, or perhaps users are entering a lot more data than was anticipated.) See LAOS Part II for guidelines on automating Performance Monitoring.

Tuning Performance

If you test sufficiently and use a configuration management tool such as the Oracle System Sizer, the initial configuration should be sufficient to meet user expectations, and additional tuning (after the database has been placed in production) will not be needed.

Avoiding Physical Reorganization

The most likely reason to physically reorganize objects in a database is to eliminate fragmentation [Shallahamer, 1994].

Avoiding Fragmentation

Fragmentation is easier to avoid than to fix, so this section focuses on prevention. There are five sorts of fragmentation we consider:

1. Tablespace free space bubble fragmentation
2. Tablespace free space honeycomb fragmentation
3. Segment fragmentation
4. Table row fragmentation (chaining)
5. Table block fragmentation

We consider each of these in turn.

Avoiding Tablespace Free Space Bubble Fragmentation

Size every object in the tablespace to have the exact same NEXT extent size, and MAXEXTENTS set to UNLIMITED. (Follow OFA guidelines so that objects with similar growth behavior will be stored in the same tablespace with each other.)

Avoiding Tablespace Free Space Honeycomb Fragmentation

Size every object in the tablespace to have the exact same NEXT extent size, and MAXEXTENTS set to UNLIMITED. (This technically does not prevent free space honeycomb fragmentation, but it does prevent it from being a problem.)

Avoiding Segment Fragmentation

When a database Segment consists of two or more Extents, it is technically fragmented. This may be due to natural growth of data, or it may be performed manually to 'stripe' the object across multiple disks. We assume here that the latter case is not going to occur in a near-zero-administration database. Therefore, only segment fragmentation caused by growth is at issue.

Segment fragmentation caused by growth can be avoided by pre-allocating each object with an INITIAL extent that is large enough to contain all of the growth that the object is likely to encounter.

Note that each Rollback Segment should be created with a minimum of 20 extents [Millsap, 1995]. Such rollback segments are therefore fragmented, but this fragmentation is expected, and therefore should be ignored.

Finally, "[r]esearch clearly shows segment fragmentation does not impact real-life production system performance" – [Shallahamer, 1994, page 13]. Therefore, the only reason to worry about segment fragmentation is that in versions of Oracle prior to 7.3 it was possible to run out of extents, preventing the object from growing and causing database work on that object to fail with an error. If objects (tables and indexes) are created with MAXEXTENTS set to UNLIMITED then the only remaining concern is that the database might run out of free space. If the application's data growth is properly understood, this should not occur.

Avoiding Table Row Fragmentation (Chaining)

Chaining occurs whenever a given row cannot fit into its Oracle database block. Normally a row is only placed into a block where it will fit. But if the row is later updated and made larger, then the extra data may be stored in one or more additional data blocks. When the row is read, the Oracle server has to jump around to all the row pieces and reconstruct the row – a process that wastes time.

(Additionally, if a table is defined such that every row will be larger than one Oracle database block, then every row will become chained. Avoiding this is easy: define a larger Oracle database block size when creating the database. This source of chaining is not of further concern to us in this context.)

It is very hard to avoid all chaining that is caused by growth of a row. Since such chaining only happens when a row is Updated, the best way to avoid this is to carefully examine Update operations in the application, test whether any of them cause much chaining in the lab, and carefully size the PCTFREE and PCTUSED parameters of that table to minimize chaining.

Setting PCTFREE high and PCTUSED low will reduce chaining in a given table, at the cost of wasting some space.

An exception to this discussion is the case where every row is always bigger than an Oracle data block. In such a case, every row will be chained. The only fix in such a case is to create the database with a bigger Oracle block size in the first place, or change the table to have smaller rows, or just live with it. (Oracle8 allows you the option to store Large OBjects, LOBs, in storage that is separate from the rest of the row; this minimizes the chaining issue. See the Oracle8 documentation for details.)

Finally, a special kind of chaining is "Row Migration", which is so rare that, for our purposes, it can safely be ignored.

Avoiding Table Block Fragmentation

When rows in a table are deleted or are updated in such a way as to shrink, the table's database blocks end up with empty space in them. If enough blocks are affected, this can make the Oracle7 server retrieve a lot of empty blocks in order to access a given number of rows.

Setting PCTUSED high will reclaim this space sooner. See Avoiding Chaining above. Table block fragmentation is unlikely to be a source of serious performance problems for most applications. Only if problems are detected during testing and a particular table found to be experiencing a lot of block fragmentation should any corrective action be taken.

Common Mistakes

There are several errors that are commonly made by developers in configuring a database for embedded (and hence low-admin) use. We consider several of these. Most fall under the general heading of "not testing enough".

1. **Bottlenecks during peak load times**
   If the users in the field subject the system to a volume of work (or perhaps just atypically high volume in one particular area of the application), they may encounter built-in bottlenecks to system performance that were not detected and corrected during shakedown testing. Consider a failure to create enough *rollback segments*. If users submit a large enough number of updates rapidly enough and (say) only one rollback segment is available for them, they may experience delays as each user gets exclusive access to his or her portion of the rollback segment. Or, delays during updates may come from contention over *redo log latches* if too few such latches are made available. In such cases, it's common for users to see a dramatic drop in system responsiveness, as if the system had "hit a wall" or "fallen off a cliff". Such sudden changes are good indicators of a bottleneck of this type.
2. **Atypical growth of data**
   If users in the field enter more data in a particular table than the designers had anticipated, then several problems can result.
   a) index imbalance and a drop in index lookup performance
   b) decreased full table scan performance
   c) error inserting or updating: MAXEXTENTS reached
   d) error inserting or updating: out of free space
3. **Lack of Protection**
   Allowing other users onto the database, or using the database as a host for other applications, or allowing users to log in with tools or applications other than the defined one, or allowing users to log in with DBA privileges.
4. **Missing a critical item**
   The theme of LAOS is "prevention not cure" – and anything you fail to anticipate, you will then fail to prevent, and will then have to cure. The answer is to test, test, test, and always, when you find an error in your LAOS implementation, fix both the implementation and the process.

Conclusions

A near-zero administration Oracle database can realistically be achieved if:

1. the user load is *below the maximum* user load that the hardware and software is capable of supporting (i.e. one does not need to perform post-installation tuning tasks to coax additional performance from the hardware; the configuration is fast enough without such intervention)
2. user expectations are met by the *default installation* provided by the application developers (i.e. the default installation does not need to be modified in any custom, site-by-site fashion in order to meet user expectations)

3. the developers take on the task of ensuring that the *application database design is capable of being run in a near-zero-administration fashion* – this requires extra testing of the application on typical user hardware through non-trivial data growth, particularly to determine proper sizing for INITIAL extents of tables
4. the developers take on the task of *simplifying* backup, recovery, and any other remaining administration tasks, to the point that the application's users can perform these tasks; any database administration task not automated and not performed by a user or super-user has been eliminated

Bibliography

1. Millsap, Cary V. (1994). *The OFA Standard, Oracle7 for Open Systems*.
2. Shallahamer, Craig A. (1994). *Avoiding A Database Reorganization*.
3. Millsap, Cary V. (1995) *Oracle7 Server Space Management* Revision 1.4b (95/10/31) An Oracle Services Advanced Technologies Research Paper
4. Millsap, Cary V. (1996) *Designing your System to Meet Your Requirements*

One note on this bibliography: each of the above papers is excellent, and will reward any DBA or developer who cares to read them. Each is freely available from a variety of sources including the Oracle corporate web site, typically in Adobe Acrobat's Portable Document Format (PDF). If anyone reading the LAOS specification cannot find the above papers, please write to me at <tcox@us.oracle.com> and I will e-mail you a ZIP file containing all four.

Glossary

Back up	To make a copy of (a database or file) as a potential replacement if the original becomes lost or damaged
Backup	A copy so made
OFA	Optimal Flexible Architecture, a method for setting up a database to allow maximum DBA flexibility with maximum ease and minimal effort and difficulty
Offline	An object (i.e. database or file) not currently being looked at or changed
Online	An object (i.e. database or file) that currently (or potentially) is being looked at or changed
Recover	To make a restored file current, as by applying incremental backups, archived logs, or online logs; also, to roll forward logged transactions to make a database current
Restore	To replace a lost or damaged original file using a backup copy