

---

# **Data Warehouse Performance Management Techniques.**

---

**Author:** Andrew Holdsworth  
**Organization:** Oracle Services,  
Advanced Technologies,  
Data Warehousing Practice.  
**Date:** 2/9/96  
**Address:** 500 Oracle Parkway,  
Redwood Shores  
CA 94065  
**email:** aholdsw@us.oracle.com

---

**Document Status:**Draft

**Abstract:**

---

This document describes the management and technical issues involved in a Data Warehouse project. Eventually these techniques will be refined through time and experience to become part of Oracle's overall Data Warehousing strategy.

## Introduction to Data Warehouse

### Performance.

The process of Data Warehouse performance management is similar to that of the design of a Data Warehouse. It is similar in that like the design and analysis phases, the techniques utilized are very different from the processes adopted in a conventional OLTP type system life cycle.

In a conventional system life cycle there exists usually numerous levels of analysis and planning. In the Data Warehouse environment the system builders are seldom given this luxury and are required to assemble a Data Warehouse in a rapid manner with little time for performance analysis and capacity planning.

This makes the Data Warehouse performance management process extremely difficult as the work loads very often cannot be predicted until finally the system is built for the first time and the data is in a production status. As a system goes production for the first time only then may a system administrator discover there are performance problems.

**If there are to many performance problems in the running of the Data Warehouse the viability of the project becomes marginal or questionable.** It is important to remember the success of a Data Warehouse can only be measured once the data is loaded and users are able to make business level decisions by extracting data from the Data Warehouse.

This background information describes the atmosphere on any Data Warehouse project at any phase. There are so many unknowns through out the process be it loading, query work load and future work load the database administrators life is one of continual uncertainty.

This process of continual uncertainty is not necessarily a bad thing either. The reason for this can be determined from the need

for a Data Warehouse. If the users knew everything about the contents of their data they would not need a Data Warehouse. As the users do not know what they don't know how can the Data Warehouse team accurately assess their computing requirements. **In essence the process of performance management and planning in the Data Warehouse is a mixture of intuition and well-educated guess work.**

This document will not rehash existing capacity planning techniques as these are all largely based upon knowing the system requirements prior to deployment of the system.

The performance of a Data Warehouse is largely a function of the quantity and type of data stored within a database and the query/data loading work load placed upon the system. When designing and managing such a database there are numerous decisions that need to be made that can have a huge impact on the final performance of the system.

Unfortunately many of the decisions made in the construction of a Data Warehouse are made in ignorance of the performance characteristics of the Data Warehouse hardware and software environments. This has resulted in massive differences in expectations between system users and system designers and administrators. It is essential that system/database administrators work closely with the Data Warehouse designers to understand and best characterize the database performance requirements. This will enable the system/database administrators to make a better best guess for the system configuration that will yield good performance.

### **Business Requirements for Data Warehouse**

The business requirements for a Data Warehouse usually involve some of the following criteria:-

Ensure Consistency of Data from Disparate Data Sources.

Load and Maintain a List of Various Business Tables.

Provide Structured Query Access to the Data Warehouse.

Provide Ad Hoc Query Access to the Data Warehouse.

Provide Reporting Access to the Data Warehouse.

Provide Exports and Summary Data to other Analytical Query Systems.

Periodic Update and Purging of the Data Warehouse Data.

For each system each of these requirements will have a varying degree of importance based upon the Data Warehouse's target deployment strategy. These should determine the initial capacity planning requirements for the Data Warehouse and a when making these estimates all efforts should be made to satisfy the most critical business requirements.

It should be also noted that having successfully built and implemented your Data Warehouse the Warehouse, **you must guard against the warehouse becoming a victim of its own success.** A well implemented Data Warehouse will by definition be regularly used. If the project is a success the users of the system will increase beyond the point anticipated when the system was first built. For this reason all estimates should include some margins or factors of safety for system growth and estimation errors. It is important to remember that the sizing of Data Warehouse systems is not a precise science.

## **Performance Definition of Acceptable Performance.**

The simplest definition of acceptable performance is that the Data Warehouse is able to meet all of the business requirements in the required time scales.

With all these types of issues the definition of acceptable becomes blurred because some of the business requirements are simply unacceptable or impossible. The impossible or “would like to have” issues need to be identified early in the life of the Data Warehouse.

Many of these problems are often caused by ignorance of the data or naivety with regards to the chosen technology for the Data Warehouse.

Example: A Data Warehouse is required to run queries on a large table that involves full table scans. The response times for these queries are very critical. To reduce the query response times parallel query is utilized. Running one of these queries utilizes 75% of all system resources on the machine for a minute. The business requirements determine that over two hundred users need to perform this query each once about every 2 minutes. Clearly here we have problem in that this sort of problem probably cannot be solved by any machine. In this example it is possible to see where a well informed set of system designers can ensure they do not commit to deliver a business requirement that is clearly impossible to deliver.

In this example a proactive database administrator may notice that all these queries could be satisfied by a simple summary table. In this case the business requirements could be satisfied with ease. In essence it takes more than just raw machine horsepower to solve Data Warehouse problems and

in fact a proactive entrepreneurial approach is what is really required.

This example is given to show when embarking on a Data Warehouse project it is important to be very careful when developing service agreements and similar agreements all parties understand what they imply. This is particularly important bearing in mind the unknown quantity of some of the Data Warehouse workloads. The following business functions are particularly difficult to assess for capacity planning purposes:-

**Data Loading and Scrubbing:** Very often the amount of data processing is unknown until the process actually begins or after the data is in production and users find data errors and anomalies. The data processing and conversion processes can be very resource and time intensive. In many cases the full database meta model is not complete until the data has been cleaned up. The process will yield more exceptions than envisaged and meta models may be subject to considerable amounts of change after this process is complete.

**Ad Hoc Queries.** This work load will strike fear into a system or database administrator. Ad Hoc Queries by definition mean a highly variable workload that can be put on the system at any time for a unspecified length of time. Numerous Ad Hoc queries that involve parallel table scans may bring the system to a complete halt. With Ad Hoc Queries a mechanism for governing their execution is required to ensure the predictable workload users can function with minimal impact.

It may be determined having built the Data Warehouse some of the requirements cannot be satisfied on the machine or indeed on any machine. However by re-engineering parts of the application and the creation of summary tables it may mean that the business requirements can be fully or partially satisfied. This should be regarded as normal evolution of the Data Warehouse. The process of continual re-engineering as user workload increases should be an every day occurrence on an active Data Warehouse.

## **Performance within Budget Requirements.**

It is fully understood that on many Data Warehouse projects the hardware capital budgets will impact the size of the machine that may be used. The rest of the document describes how to build a balanced system. Should hardware budgets limit the size of the Data Warehouse machine all parties should be aware and modify their expectations accordingly. It should be noted however that well sized machine initially will allow more freedom and give the project a higher chance of success and subsequent growth.

It is important to have the Data Warehouse in production as soon as possible. **Economies in hardware specification that cause time delays are only false economies in the sense that you cannot generate revenue from a Data Warehouse that is not in production.**

# Capacity Planning for the Data Warehouse

## Producing a Balanced Design or Configuration

When configuring a Data Warehouse system it is important to insure that all components are balanced. What this means is that all active components ( CPU, Memory and I/O ) are all effectively used and when one component reaches its maximum operating range all the other components approach their maximum at the same time.

This practice has been perfected by engineers for centuries. A Civil engineer building a bridge will design each component to fail at the same time. After all, the strength of a structure is determined by the weakest link in the structure. In addition an engineer will optimize the structure to use the most cost effective materials and construction methods to build the structure.

This principal can be applied to configuration of the Data Warehouse. We know that the CPU is the most expensive non renewable resource on the system. With this fact appreciated the memory and I/O subsystem should be constructed to ensure that the CPU can be kept busy and the design held in balance. Failure to do this makes the memory or I/O subsystem the weak link in the structure.

## CPU Requirements and Scalability.

CPU sizing and requirements for a Data Warehouse are a function of the database size and the user workload. Both of these factors should be considered when determining the size and the number of CPUs.

The database size will impact the number of CPUs simply because a database needs to be administered. A database requires periodic reorganization which requires extracts of data, data reloads and data re-indexing. A database requires backing up and restoring in the case of a media component or operational failure. All of these processes requires CPU resources. As the

data volume increases it becomes more important to run these operations in parallel to make the database batch windows and availability requirements. For this reason the database size will have a large impact on the number and size of the CPUs required. As a rule of thumb if the database size drives the CPU requirements a recommended would be to assume about 5 Gig of Data per CPU assuming fairly conventional availability requirements. This rule is not a true linear rule because when the database size increases some economies of scale can be exploited. Also as systems increase in size their I/O subsystems tend to be upgraded and more efficient.

When adding additional CPUs to a system the scalability with regards to additional CPUs should be considered and fully understood. It is naive to assume in a multiple CPU environment ( SMP or MPP ) that you will achieve linear scalability on all operations for all the CPUs. It is important to understand which operations scale well and under what conditions which operations are inherently single threaded and do not scale at all.

In summary characteristics of CPUs are simple and predictable. Faster CPUs process single threaded jobs faster than slow ones. If your Data Warehouse workload has a number of single threaded jobs or processes look to get the machine with the most powerful CPUs. Multiple CPUs allow multiple jobs to run concurrently hence increasing system throughput. The increase in throughput is a function of the scalability of the system. Multiple CPUs also allow the Data Warehouse to take advantage of the parallel features of the Oracle RDBMS ( 7.1 Onwards ). Again the speedup that can be gained from these features is a function of the scalability of the system.

From these simple facts it is possible that a machine with few very powerful CPUs may provide a better solution than a machine with multiple low powered CPUs and vice versa. Historically machines with more CPUs have been able to support more users and provide more consistent response times than machines with fewer CPUs because they are less sensitive to a large process dominating the whole machine. Again making an

initial assessment of the Data Warehouse workload will help guide the selection of the correct CPU configuration.

The following Table describes the types of CPU configurations and comments on their Scalability and single threaded performance.

CPU Environment	Single Threaded Performance	Scalability	Comments
Single CPU	equivalent to CPU rating	None	Big CPU machines give the best performance e.g. HP and DEC Alpha
Low End SMP (Small Intel, Sun 1000 )	Poor	Good	Good scaling can be anticipated in 1 4 CPU range
High End SMP ( DEC Alpha Turbo Laser, Hp, Pyramid, Sun 2000, Large Intel)	Excellent	Good	Good scaling can be anticipated in 1 12 CPU range
MPP ( IBM SP/2 Pyramid RM1000 )	equivalent to CPU rating	Varied	Excellent scaling on processes that are designed to scale

Scalability is a function of many items of which a great number of them are out the control of the system administrator and are built into the hardware, operating system and RDBMS. There are however a number of items which are under the control of the capacity planner and system administrator which will effect the scalability of the system.

One of the core issues is to ensure that the system is balanced such that the CPUs can be fully utilized. The CPUs are the most critical resources on the machine so they should be the most fully

utilized when ever possible. A fully utilized CPU means that the most work is being done and the Data warehouse is effectively working at its optimum rate. If performance is not satisfactory the number or type of CPUs requires increasing or upgrading.

In modern SMP and parallel processing machines it is often difficult to get the CPUs fully utilized because there is bottleneck in the system and the system is out of balance. The balance can be re-established by the increasing other system resources such as memory or I/O bandwidth.

When conceptually modeling the Data warehouse machine it is useful to imagine the I/O subsystem as a pump to supply data to the CPUs. With an increase in Memory the pump may have to work less hard as the data is in memory and less pumps are required to keep the CPUs busy. With an increase in I/O bandwidth ( Disks and Controllers ) we are effectively creating a bigger pump. As we increase the number of CPUs on a system the requirement for a bigger I/O subsystem becomes more apparent.

Sizing of the I/O subsystem is crucial to the scalability of any parallel operations in Data Warehouse. Failure to recognize this and address it early in any Warehouses evolution will result in time consuming rebuilds and user frustration.

### **Memory Requirements.**

Memory on any RDBMS application is used for a number of distinct tasks. For each of these should insufficient memory resources be available a bottleneck will occur in the system causing the CPUs to work at a lower efficiency and the system performance to drop.

The main memory on the machine is used for the following distinct tasks:-

Process and Operating System Virtual Memory.

Buffer Caches.

Sort Space.

### **Process and Operating System Virtual memory.**

This memory is used to allow execution of the actual processes on the machine. This memory embraces all the elements of memory described in this section, however when considering just the process space and operating system overhead the requirements in the Data Warehouse environment are relatively modest. This is because relatively few active processes exist.

The other elements of memory may contribute to the situation where the sum of all the active virtual memory exceeds the actual amount of memory. In this scenario the operating system has to manage the virtual memory by prioritizing the least busy pages of memory and storing them on disk. This process is often called paging and in more pathological cases an entire process state may be stored on disk and this causes swapping. In any case should this happen performance will be heavily degraded and for this reason a system should not be allowed to page or swap. Should paging or swapping occur it may be necessary to scale back memory allocation to the database to ensure process can run in memory. If this is unacceptable for performance reasons further memory should be purchased.

There is however usually limits to the amount of memory that can be physically assembled or addressed on most modern machines. In many cases the amount of memory is constrained by the number slots in the computer backplane. If the backplane to the machine can accommodate a great deal of memory the operating system may not be able to address the memory effectively. This is due to the fact that most operating system are

32 Bit systems and will hit addressable limits at 2 Gig and 4 Gig respectively depending if signed or unsigned machine integers are utilized. With the development of 64 Bit operating systems many of these type of issues and problems will be eliminated.

### **Buffer Caches.**

The majority of main memory in RDBMS applications has usually been applied to the construction of very large buffer caches. In Oracle terms this cache is often called the SGA and is sized by the init.ora parameter db\_block\_buffers. Large caches are very advantageous to database applications if you consider the performance issues. To read a data page from memory takes about 20-30 micro-seconds and to read a page from disk takes about 20-30 milli-seconds. Knowing these numbers, if the most often used data pages are kept in memory considerable performance advantages can be gained. The Oracle RDBMS uses sophisticated Least Recently Used algorithms to determine which pages shall remain in memory or will get flushed to disk. Many tuning documents will refer to cache hit ratios. This ratio is very simply the fraction of times when a data page is accessed relative to the times it is located in memory. This number should be conventionally in the high 90s on normal systems.

There are additional caches within the database that also require correct sizing. These are usually associated with building caches of information to support data dictionary activity. These can be sized by using the init.ora parameter shared\_pool\_size.

The other data cache that usually gets neglected is the operating system file system cache. Very often this can grow to be very large especially if the database is built using file system files as opposed to raw disks. It is important to understand the implications of the size of this cache as very often incorrect decisions are made about performance without fully understanding the size and impact of the file system cache.

## **Sort Space.**

Any database will spend considerable time and resources sorting large sets of data. This process will run faster if the sort process can run entirely in memory without the need to use intermediate disk storage in the process of sorting. The amount of memory and Oracle process uses for sorting is controlled by the init.ora parameter `sort_area_size`. However when using the parallel query option the total amount of memory used for a sort may be the degree of parallelism multiplied by the `sort_area_size`. If multiple concurrent queries execute simultaneously this can add up to a considerable amount of memory. Care should be taken to assess and control the amount of sorting memory that gets used to avoid the possibility of paging and swapping occurring.

## **Memory in the Data Warehouse.**

In Data Warehouses the allocation of memory should be a function of the type of workload on the system. If the Data Warehouse is performing parallel table scans, sort and joins of large tables with very little indexed access there is very little point in building a large buffer cache as the chances of getting a good cache hit ratio are minimal. In this situation the memory should be allocated to the sorting process. If however the Data warehouse is working off keyed access to data like in a star query it will be important to get good cache hit ratios and the system memory should be configured into buffers. Naturally all systems are a cross between the two extremes, however, knowledge of type operation and how best to configure for each type will yield huge performance gains.

## **64 Bit Computing**

Earlier reference to machines with a 64 Bit operating system was made to supporting very large memory configurations. These are now a commercial reality with machines from DEC and future machines from SGI, HP and Sun. To take advantage of the large memory configurations requires no additional work other than to

increase the size of the init.ora parameters. It has been noticed that significant speed ups have been achieved using buffer caches in the order of 4-8 Gigabytes and total sort size in Gigabytes.

### **Disk and I/O Subsystem Requirements.**

It is described earlier in this document that the I/O subsystem can be compared to a pump that pumps bytes of information at the CPUs to enable them to perform a workload. This is largely true. In addition the I/O subsystem provides a permanent data store for the information. This second factor is unfortunately what usually drives capacity planners in their sizing of a Data Warehouse I/O subsystem. What usually happens is that a database I/O subsystem size is determined from storage requirements rather than performance requirements.

The following notes describe the problems involved when configuring an I/O subsystem and provided hints as to how best match hardware configurations to potential user workloads.

### **I/O Subsystem Components**

The main components in the construction of an I/O subsystem will be an array of disk drive devices which are controlled by a number of disk controllers. The controllers are usually then linked to the main system bus via various components with various names such as I/O adapters, Channels etc. The precise names and the architecture for each vendor will be different as they adopt different design philosophies. When working with the hardware vendors it is important to ensure you explain your design objectives in disk I/O terms so that they can build a balanced I/O subsystem.

As stated earlier the disk drive(s) will be the core component of any I/O subsystem. Disk drive technology whilst it has improved over the last few years has not improved at the speed of CPU development. On average CPU speeds double every 18 months however disk performance may at best only increase a few percent over the same period of time. For this reason as CPUs become more powerful there will be a need for more powerful

I/O subsystems and hence more disk drives. Disk drive manufacturers whilst not increasing disk performance greatly ( mainly due to laws of physics ) have greatly increased the storage capacity of a disk, reduced the cost of manufacture and have increased the reliability of their components.

The increased size of disk drives has provided cheaper mass storage ( it cost as much to make a 4 Gig disk as a 2 Gig disk ) but has confused the industry because the disk throughput has not increased proportionately. In a database application to increase the I/O bandwidth the number of disk drives needs increasing. However if the number of drives is halved due to having bigger disks a performance problem is only inevitable.

In summary the performance of the disk drive is very static at about 30 I/Os per second as good working average and will remain at this level for a considerable period of time. It is important to not be side tracked by disk storage capacity when dealing with performance issues. In large modern SMP machines there is usually significant over capacity in terms of disk storage space if the I/O subsystem is configured to keep the CPUs busy. This statement assumes the database size is less than about 100 Gigabytes in size. For larger databases the number of spindles may mean the I/O subsystem is indeed more powerful than the CPUs and a capacity planning model by volume is required rather than capacity planning by throughput.

### **Capacity Planning Estimation Techniques**

The previous section described that most I/O subsystem capacity planning should be done by anticipating system performance and ensuring that the I/O subsystem can keep the CPUs busy. A great deal of this will be alien to administrators who are used allocating disk space by volume.

It is important to remember that in a Data Warehouse most of the work loads are indeed unknown. With this fact understood it would seem sensible to design an I/O subsystem that will keep the CPUs busy. This means that the one query that utilizes all system resources will work as fast as possible and that is after all

the objectives of the Data Warehouse to get the query results to the users as fast as possible.

With this in mind and the fact that there are so many unknowns it is best to use a very simple method of estimating the disk requirements. On embarking the process of estimating the number of disks, controllers etc. the capacity planner usually has only a few inputs namely:-

The size of the database

The number of CPUs

An initial estimation of activity on the Data Warehouse e.g. Table Scans, Star Queries, Conventional Queries etc.

From these it is possible to work up a possible configuration bearing in mind additional operation factors such as the following:-

Is the database to be mirrored.

How space is required to stage the database prior to data load.

How much space to allow for Indexing and Temporary Sort Space.

The best way to show how to work up hardware estimate is by performing a worked example. These calculations follow a back of an envelope style, and the quality of the results will be a function of the quality of the assumptions made whilst working with the client in the analysis and early requirement phases.

## Worked Example

A customer wishes to build a Data Warehouse to support their customer information system. The database consists of about 10 core tables that comprise about 60 Gigabytes of data. They are going to perform a great deal of table scan activities as they mine the data for purchasing trends. They know the data is coming from a number of legacy systems and anticipate many problems consolidating the data. They anticipate regularly sweeping or scrubbing entire tables and creating updated copies. For this reason they know the processing requirements are going to be extremely high and for this reason have ordered a 20 CPU Sparc Server 2000E. They have configured 2 Gigabytes of memory into the system and now they need estimates as to the size of the I/O subsystem. They have requested that the database is fully mirrored in case of media failure.

### Assessment of Database Size and Volume based Estimate

(base data+indexes+temp) \* Factor for Admin.

$$(60 + 15 + 30) * 1.7 = 179 \text{ Giga Bytes}$$

The factors are arbitrary and are based upon experience they can be altered when applicable.

The disks available are 4 Gig Volumes so the number of disks prior to mirroring will be  $179 / 4 = 45$  Disk Drives

### Assessment of Disk Drives from CPU performance

#CPUS \* Disks/CPU estimate

$$20 * 8 = 160 \text{ Disk Drives}$$

In this case the number of disk drives to keep the CPUs busy exceeds the storage requirement. Using a volume based estimate could mean that there could be an I/O bottle neck on the database.

Mirroring the database would double the number of disk drives to 90. Mirroring will also assist query processing as reads can be satisfied by multiple spindles. However the scale up is not perfect so we can assume an I/O scale up of about 50%.

This means that the effective number of drives using a volume based estimate would approach about ( 1.5 \* 45 ) 68 drives again well short of the 160 drives calculated to keep the CPUs busy.

At this point these estimates need to be communicated to the project team and cost benefit assessment needs to be made. To get all CPUs busy will require doubling the number of disk drives. An assessment needs to be made to see if budget limitations allow the project buy almost twice the number of disks. At this point it usually becomes a judgment issue and compromises are required. Other factors that should be considered when making this estimate.

Is the number of CPUs going to increase in the future

Are the CPUs going to be upgraded to faster ones in the future

What are the chances of the database further increasing after phase one of the project.

All these issues encourage early investment in disks however budgets will usually override the technical arguments. It is stated that this process is a very imprecise science and I/O subsystem capacity planning is one of the hardest to get correct. However if the database administrators cannot negotiate enough I/O resources there will be I/O bottlenecks later. If obtaining hardware resources are not an issue 160 Disks should be obtained as part of creating a balanced system.

### **Configuration for Performance**

Having secured considerable amounts of disks on which to build the Data Warehouse thought has to be given as to how to best layout the database so the full potential of the I/O subsystem can be realized.

The process of spreading parts of the database out over multiple disk drives is commonly known as striping. When striping a database the database administrator has a number of options on how to perform the striping operations. It has been stated earlier that the work loads on the database will be fairly unpredictable and for this reason it is important to come up with a disk configuration that does not penalize any type of workload.

The types of I/O that is likely to take place on the database files will be either random single block reads and writes or sequential reads and writes. The majority of I/O activities will be read I/Os upon the database however an understanding of what writes take place is required.

The core database I/O activities are described in the table below.

I/O Type	Database Activity
Sequential Read	Table Scans, Read from Temp segment, Recovery of Database from Redo Log
Sequential Write	Direct path Load, Log Writer Write, Write using Sort Direct Write (7.2), Write using unrecoverable option (7.2).
Random Read	Any Indexed or key access operation
Random Write	Dirty buffer write

### **Avoiding Contention**

When organizing and building the database it is essential to avoid contention. Contention can be defined as disk I/O operations that negatively impact another process performing a disk I/O operation. An example of this is two processes simultaneously wishing to read two different parts of a disk. One process will have to wait until the first disk I/O is performed before the second I/O request is performed. This introduces the issue of

queuing for a device. Contention can be made worse if each process then wishes to read the next block after the block they just read. In this case to perform the I/O operation the disks would have to seek to the position to perform the I/O. This takes time and if there was no contention at all the disk I/O would take very little time at all. In order to minimize contention we are effectively attempting to reduce disk queue lengths and minimize seek operations on the disk.

In the previous table it shows the type of I/O operations performed within the database. There are certain process that continually perform sequential I/O and this process is critical to the performance of the database. In particular the log writer performs sequential writes when logging database transactions and sequential reads when recovering the database. The log writer I/O operations are designed to be and need to be very fast. For this reason the database redo log files should always be located on a dedicated disk without interference from other processes.

Similarly when locating the database files the database designer needs to follow the same logic. It is important to separate those database objects that are subject to sequential I/O such as tables when they are being scanned and those subject to single block random I/Os such as index blocks and tables being accesses by key access.

To resolve the contention it is recommended that a truth table of anticipated access paths to each data object be built as it gets defined within the database. Using the table it will be possible to see if an object can coexist on the same disks as other object or if it should be given a dedicated set of disks.

### **Techniques for Striping**

There are two methods of striping a data object. The first method is manual striping in which the database administrator physically locates an object over multiple devices by creating multiple data files. The other method is to use machine or hardware striping. Hardware striping allows the database

administrator to define a single logical file that spans multiple physical devices. The administrator is also able to define the size of the stripe taken from each device.

The advantages and disadvantages of the various striping techniques are described in this table:-

Striping Method	Advantages	Disadvantages
Manual Striping	Best performance for single query parallel scan. Good for benchmarks.  Conceptually Simple.	Difficult to setup and administer in future use.  No ability to load balance if multiple parallel scans on the same object.
Machine Striping	Easy to setup and administer.  Load balances if multiple parallel scans occur on the same object	Performance penalty of about 5-10% when performing single parallel query.

From this table it is easy to see when striping the database the administrator needs to decide whether to invest time on manual striping or to use machine striping with a small performance penalty. In Data Warehouse where the workload profile is unknown it is recommended that machine striping is used to stripe all objects. To eliminate contention for disks it is recommended that tables that are subject to multiple concurrent parallel scans be given a dedicated set of disks striped to give satisfactory I/O bandwidth and load balancing abilities.

The final issue with machine striping is that of stripe size. This issue is a hotly debated issue. The stripe size will drastically impact table scan performance as well as database operational issue such as backups and restores. When setting the strip size

the administrator should attempt to insure that each I/O can be satisfied within one stripe. This means if table scanning the stripe size should be at minimum blocks size times the value for the init.ora parameter db\_block\_read\_count. It is unlikely however that the table scan will align itself with the stripe boundaries so a larger number is actually required. **Do not allow the system administrator to set the stripe size to the Oracle blocks size.** This has been shown on many occasions to be very degrading to all database operations. **A recommended size for the stripe size is about 1 Meg and on very large systems 5 Meg.**

## **Tape and Media Management Requirements.**

When building the Data warehouse at some point in time the database will need to be backed up and restored. As many of these databases are very large sophisticated tape and media management system will be required.

When evaluating the media system the following issues should be addressed:-

Cost of Loss of Data.

Speed of Backup and Restore.

Amount of Operator Intervention Required.

Compactness and size of media storage.

Cost of overall Media management subsystem as a percentage of whole system cost.

## **Data Placement Techniques within a Data Warehouse**

### **Compromising Administration against Space Used**

Having determined the size and made best guesses on the access patterns on each object within the Data Warehouse, the database designer or administrator needs to formulate a strategy as to how best store the object within an Oracle database. This process usually involves determining which object should reside in each Oracle database tablespace. The database designers and administrators will then ensure that each tablespace is created with the correct I/O characteristics such as striping to ensure satisfactory I/O performance.

The object to tablespace mapping involves a number of variables but the most important thing is to remember the design objectives of this phase. When determining an object to tablespace mapping there two core objectives that must be addressed. These are as follows.

- Give sufficient I/O subsystem bandwidth to each database object ( Table, Index or Cluster )

- Layout the database objects in a manner that will minimize future database administration and system downtime.

Whilst these objectives should be obvious they very often conflict and this places great responsibilities on the database designer and administrator to resolve these issues and understand the impact of any compromises.

There are some very simple rules of thumb to creating an object to tablespace mapping. These are as follows.

Give Large objects their own tablespace.

Group objects of similar access patterns together into a single tablespace. For Example Tables, indexes.

Separate objects with high degrees of space management activity into their own tablespace. For example temporary or sort segments, rollback segments and derived or summary tables.

### **Working to reduce fragmentation and row chaining**

Fragmentation and row chaining will have serious performance and operational impact on your data warehouse. This degree this affects each Data Warehouse will largely be determined by the nature of each workload. There are however some things a database designer and administrator can do on creation of the database and its objects to attempt to minimize these problems.

Fragmentation within a tablespace will be caused by the dynamic allocation of extents for database objects and their subsequent de-allocation. This process over time will leave gaps and holes within the database that cannot be used and will eventually waste database space and make some database operations inoperable. Once this happened a database re-organization is usually inevitable.

To eliminate fragmentation is impossible however there are steps that can be taken to minimize or at least slow down the process. These are as follows:-

When creating database objects ensure that the storage parameter PCTINCREASE is set to zero. This parameter when left at the system default of 10% probably causes more fragmentation than any other single issue.

When creating objects within a single tablespace it is best to adopt a uniform extent size for all objects within the tablespace. This prevents a great deal of fragmentation because any reallocated extents will be immediately reused. This technique is essential for temporary and rollback segment tablespaces.

In tablespaces where large objects are being stored make the extent size approximately one tenth the size of the data files for that tablespace. In this way monitoring space utilization will be simple and require minimal effort.

### **Impact of Direct Loader and Parallel Features on Fragmentation**

Use of the Direct path SQL\*Loader and the parallel features is essential on any Data Warehouse database of any significant size. These facilities however work in a manner that can cause fragmentation and all database administrators should be aware of these features. These are as follows.

When loading data using the direct path loader in parallel mode the loader will truncate the last extent to be populated in the data load from the length specified in the storage clause to the length of the populated extent. This means when using multiple direct loads into the same file gaps can be created that cannot be filled.

When building Indexes in parallel or utilizing the unrecoverable create table as select each parallel slave process will allocate extents according to the storage clauses and again will truncate the length of the last extent allocated. Again this can create gaps in the database of empty space.

## Data Cleaning

The data cleaning or scrubbing process can itself make or break the entire Data Warehouse project. The data may require so much processing that in the life of the Data Warehouse it is never perfect.

It is recommended that if possible most data cleaning is done prior to insertion into the database. Much of this work can be done with tools designed specifically for the process. These tools also have the advantage in many cases that they provide historical mapping data to map the Data Warehouse data representation back through the cleaning process to the operational data. This issue may be crucial when building very complex corporate Data Warehouses. In smaller more bespoke systems the cleaning process simply filtering the data may be sufficient.

The data cleaning process prior to data loading should however allow for the following features or characteristics.

Product data output suitable for loading by the direct path SQL\*Loader. This allows rapid loading into the database with minimal overhead. A full explanation of the benefits of the direct path approach is covered in the application section of this document.

The cleaning process should function in parallel. With larger and larger databases this process cannot afford to run as single threaded process. The process must be able to run in parallel to facilitate data cleaning within a reasonable operational time window.

The cleaning process should be self documenting and should generate summary statistics and exceptions to the process.

The project should determine “How Clean ?” the data needs to be before putting the data into production. A great deal

information of useful work can be done on imperfect data. A cost/benefit approach needs to be applied on when to stop cleaning and the process of Data Warehousing begins. This is very analogous to cleaning the bottom of a boat. Once most of the nastiest weeds and barnacles have been removed there is nothing to stop the boat being used. There is however a law of diminishing returns once the boat bottom has reasonable level of finish.

Small numbers of rows ( 100s ) can be manually corrected or cleaned up later whilst in the database using SQL. However the bulk of data modification and transformation needs to be done prior to loading into the database.

### **Types of Data Cleaning and Transformation**

There will be various types of operation that is required to clean a set of data. Some of this should be done external to the database. Some of the cleaning processes it is possible to use the database utilities to clean and validate the data. If the direct path loader is used to load the data the data loading process is very rapid. A table may get loaded into the database for analysis only to determine more data cleaning is required. The data may be cleaned in the external files and the table may get reloaded multiple times. Another solution is use the Unrecoverable option of the Create Table as Select ( CTAS ) in parallel and recreate the cleaned table within the database. The purpose of this discussion is to demonstrate that for each data cleaning problem there is multiple solutions and it will be the skill of the Data Warehousing project to determine the best process. The problems of data cleaning can be classified under the following classes of problems:-

### **Application Cleaning**

This can be summarized as manipulation of the data to make the warehouse function. An example of this would include unification of customer identifiers from diverse data sources.

### **Data Inconsistency Cleaning**

This can be summarized as the process of cleaning up the small inconsistencies that introduce themselves into the data.

Examples include duplicate keys and unreferenced foreign keys.

### **Column Level Cleaning**

This involved checking the contents of each column field and ensuring it conforms to a set of valid values. These may be enumerated data types or actual values. Free format and text fields should be stored in the correct collating sequence. For example convert EBCDIC to ASCII.

This Table is not complete but it does however attempt to describe some of the most common data cleaning problems.

Problem	Class of Problem	Potential Solution	Simplicity of Solution
Unification of data into common Entities from multiple systems.	Application.	Use of tools or well documented transformation mechanisms.	Largely depends on diversity of data. Should initially be regarded as complex.
Elimination of Duplicate values.	Data Inconsistency ..	Use SQL joining table to its self or use RDBMS Constraint utilities.	Simple to perform. Can be very time consuming.
Elimination or resolution of orphaned records.	Data Inconsistency .	Use SQL with Anti-Join Query or use RDBMS Constraint utilities.	Simple to perform however time consuming. Will require business experience to resolve inconsistencies.
Invalid Dates	Column level Inconsistency .	Can be eliminated on data load. However application may need to further reduce applicable date ranges.	Many of these problems derive from storage mechanisms in legacy data. Use tools or filters to pre-process data into format suitable for database.

<p>Invalid Column Values</p>	<p>Column level Inconsistency / Application.</p>	<p>For large sets of rows pre-process prior to data load or if loaded recreate table using CTAS. For small data quantities use SQL UPDATES.</p>	<p>This will be function of business rules complexity. Remember other fields may be suitable for other query work. Use Cost/benefit to determine if this processing is required immediately.</p>
<p>Number Precision</p>	<p>Column level Inconsistency / Application.</p>	<p>As for invalid column values.</p>	<p>Complex to determine optimum levels of number precision. However precision should not exceed accuracy of data.</p>

## **Assessing Data Cleaning requirements**

When estimating or bidding for any Data Warehouse project the data cleaning component represents the least quantifiable and exposes any project to the most risk of cost and time overruns. In this section an attempt is made to identify the parameters that make the data cleaning process simple and those that increase complexity and hence risk. Readers of this document should be aware for data sourced from old legacy systems very often a customers staff will not be able to give you accurate schema information about their own data and the process of data cleaning could be better renamed as data discovery. Do not expect perfect data and supporting documentation from customers as much of the knowledge on these systems existed over 5-10 years ago when these systems were developed. The staff who held this knowledge base will have left the organizations a great deal of time ago.

### **Factors that Contribute to Simpler Data Cleaning**

Single source of data

Source data stored in Oracle database

Source data stored in relational database

Source data stored in same byte order as DW

Source data uses same collating sequence

Source data uses relational constraints

Source data uses column level constraints

### **Factors that Contribute to Complex Data Cleaning**

Multiple diverse sources of data

Data not initially stored in relational format

Data not stored in same byte order ( Byte swapping required )

Data stored in different collating sequence

Data coming from application that enforces minimal validation and constraint definitions.

Non Oracle Sources ( Gateways can assist here )

## **Application Performance Techniques.**

### **SQL Statement Design and Tuning Techniques**

Tuning SQL statements is an area where there is literally no substitute for experience and practice. A great deal of SQL statement tuning is subjective and usually exploits some knowledge about the data that the current optimizer is unable to fully model or represent. The optimizer improves with every release of Oracle as better statistics and plan generation algorithms are incorporated into the kernel. As users however everybody should be fully aware that the SQL optimizer has limitations and these will be recognized by experience. Fortunately for developers it is possible to “hint” SQL statements and manually override the optimizer to manually achieve optimum execution plans.

For more formal methods of tuning SQL statements please refer to Oracle documentation or the book “Oracle Performance Tuning” by Corrigan and Gurry. These documents describe the mechanics and well as the background to this process. In Data Warehouse applications the query execution plans in many cases if they are not correct may mean the results cannot be derived in reasonable time. For this reason identifying the good and bad plans becomes very important and should be part of any Data Warehouse process.

The usual mistakes and areas for investigation in SQL statements are as follows.

#### **Table Scan or Indexed Access**

When accessing each table in a query there are two fundamental methods ( hash clusters excluded ). These consist of scanning the entire table or using an index on the keyed fields within the “where” clause of the query. There is a very simple rule of thumb when an index is better than a full table scan. If you

going to retrieve less than 20-25% of a table then index access is better solution otherwise use a full table scan. In many cases where consultants have forced queries to use indexes they slowed down queries because a table scan is faster. It is important to always consider the selectivity of indexes prior to their use. Use of an index does not imply an optimum execution plan.

### **Incorrect Join order and Type**

This is particularly prevalent on SQL statements written to execute 7.2 or lower of the Oracle RDBMS. This is because the optimizer will only optimize the SQL statement fully if there are 5 or less tables in the SQL statement. If a 7 table join is required it will probably generate a sub-optimum execution plan.

Additionally with 7.3 of the database there is 3 methods to join each table namely, nested loops, sort merge, and hash joining. In a complex query the number of possible join mechanisms becomes a very large set of combinations. In order to determine the best execution plans the SQL statement tuner needs to use the guidance provided by the optimizer with his or her knowledge of the data to determine the best join order and join mechanism.

### **Lack of Index Definition or Poor Index Definition**

Poor index design is responsible for great number of performance problems. The selectivity of an index can be greatly increased by the use of concatenated index to include many of the “where” clause predicates and hence reduce the size of an index range scan and the subsequent number of table lookups. Further refinements can be made to the design of a concatenated index by appending the remainder columns from the select list of the query such that not table lookup are performed in the query at all. In this case the query is entirely satisfied from within the index.

Another issue index designs is that very often little thought is given to key ordering when specifying a concatenated index. If possible is best to place the columns in order of selectivity with the most selective column first. This is particularly important to verify index definitions and designs generated by case tools as

these tools tend to generate the index definition according to the dictionary ordering of columns and not by selectivity of the columns.

### **Unnecessary Sorting**

A great deal of time in any query can be wasted by unnecessary sorting and filtering. Some of the most common of these may involve the use of a “union” statement where a “union all” would have been suitable. In this case the “union all” eliminates the need for a sort to eliminate duplicate rows in the set operation.

A more common example is when using an “order by” clause. If an index range scan is used and the index is built according to the ordering specified by the “order by” clause the query is able to retrieve the data presorted and then eliminate the sort operation to satisfy the “order by” clause.

### **Implicit Datatype Conversion**

This problem is a very common problem in all types of system and its discovery and subsequent correction has increased the performance of thousands of systems worldwide. It is particularly common on tables that have a key that consists entirely of digits however it is stored as character or varchar field. Examples of these would be check numbers, serial numbers, sku numbers or any key that whist containing only digits no arithmetic operations are performed upon it. The usual fault made by the SQL programmer is when assigning equality statements in the “where” clause predicate list. On specifying a the equality the equals sign is used but the variable is not surrounded by quotes. This forces the statement to be executed as numerical comparison rather than a lexical comparison. This will implicitly force a full table scan rather than using the desired index.

Example:

Table Customer has a primary key cid. This is ten byte character field which comprises entirely of digits.

This SQL will perform an implicit data type conversion by performing an implicit to\_number on the data. This will execute as full table scan.

```
select * from customer  
where cid = 999999999;
```

However this SQL will perform a lexical comparison and will use the primary key index.

```
select * from customer  
where cid = '999999999';
```

## **The Transactional Options within Oracle**

There are two basic mechanisms to manipulate data within an Oracle database. The conventional route is via SQL by use of Update, Delete and Insert statements. This mechanism uses the full transaction layer that has been optimized for high performance OLTP applications. This mechanism uses rollback segments for the storage of read consistent images and redo logs to ensure the changes are made permanent. Whilst the use of SQL is excellent for OLTP when manipulating the size of objects found in a Data Warehouse this approach reaches various practical limitations. The alternative approach is to make use of the direct interface to the database that bypasses the entire SQL and subsequent transactional layer.

Since the late versions of V6 or Oracle SQL\*Loader has been shipped with a direct path option. This option provides a rapid mechanism for loading of data into the database. It does this by bypassing the SQL interface and writing database blocks directly to file. This process bypasses the buffer cache, the transaction layer avoiding the overhead of logging and for this reason is very

fast. The only disadvantage of this mechanism is that it does not provide full database recovery and the database should be backed up after the data load. Whilst this is not ideal for mission critical OLTP systems it is ideal for Data Warehouses where rapid loading and subsequent backup is what is required without the addition of numerous transaction logs. In the various 7.X releases of Oracle this basic philosophy has been extended to other operations to allow parallel direct path loading into a single table and most what will be the most useful for Data Warehouse operations the “unrecoverable” clause on “create index” and “create table X as select ..” operations. This functionality allows rapid creation and loading of tables that do not require any of the overheads of the transaction layer and they function in parallel. This allows rapid movement of data into and within the database and does not stress the read consistency models and also reduces the time for database reorganization operations.

### **Use of Parallelism in a Data Warehouse**

Data Warehouse databases are by their very definition are in the class of a Very large database or VLDB. For this reason to practically load and administer these databases a single CPU will not be adequate to perform all operational functions. For this reason multi-CPU machines and software to enable jobs to execute in parallel are essential.

To exploit fully the parallel nature of these machines and software great care has to be taken to ensure that the process does not serialize upon common resource or as it more commonly referred to as single threading upon a resource. An example of this would be parallel table scan were multiple CPUs are attempting scan a table that was stored on one disk drive. Very rapidly it would become obvious that it was impossible to keep the CPUs busy because they single threading upon this disk drive.

From this very simple example it is very simple to understand that when performing operation in parallel everything has to be done to eliminate single threading. Unfortunately detecting

single threading is by no means a trivial task and again requires a great deal of experience and internals knowledge. If single threading is suspected it is best to collect the statistics defined in later sections on this document and seek help from staff familiar with the hardware and software components.

### **Use of Partitioning in a Data Warehouse**

A great deal has been written about partitioning within database in numerous books and publications. These notes will not rehash the same principals other than to give the some the key benefits of partitioning in a Data Warehouse.

The first issue is that database administration may make some sort of partitioning inevitable. If a table grows to large such that basic operations such as re-orgs and index builds take to long the table will require partitioning. The manner in which this partition occurs will have large impacts on the final applications.

The second issue is that partitioning can be used to speed query performance. If a table is partitioned horizontally by specified key boundaries such as date ranges queries can be made more selective by only querying those partitions which satisfy the query criteria.

### **Purging and Deleting of Data.**

Systematic purging of data from the Data Warehouse is one of the more difficult and resource intensive operations. This operation needs to be executed periodically in a minimum of elapsed time. Usually at this time it is also best to de-fragment and reload the data for optimum query performance.

To purge data from the database there are number of options that may be adopted. Each one has its benefits and disadvantages. Data Warehouse designers and administrators need to build a purging strategy into the long term Warehouse strategy. The options for purging of data are as follows.

Purging Mechanism	Advantages	Disadvantages	Example
SQL Delete Statement	Simple to program. Works well for small data quantities.	Very slow for large data volumes. Can cause excessive rollback segment and redo log I/O. Will leave holes in the database.	delete from sales where salesdate between '1-Mar-95' and '31-Mar-95'; commit;
Use of Horizontal Partitioning	Requires a ranged horizontal partitioned solution to allow purging process to simply drop partition of data to be purged.	Purge process very fast. However will impact query and data load programming.	drop table salesmar95;

<p>Selection of Delete Compliment into new table.</p>	<p>Can select compliment of delete statement into a new table. e.g. query the rows you wish to keep and then remove the old table. This process can run in parallel and will defragment the database. For large deletes it will not be constrained by single threading and the overhead of the transaction layer. This is Particularly useful when partitioning is not an option.</p>	<p>Requires significant D.B.A. attention during the process. however this process is very fast.</p>	<pre>create table newsalesas select * from sales where salesdate&gt;'31-Mar-95' or salesdate&lt;'1-Mar-95';  drop table sales;  rename newsales to sales;</pre>
---	---	---	---

## **Specialist Data Warehouse Query Techniques.**

One of the most important query types that will be exploited within a Data Warehouse practice is the star query. As great deal of Data Warehouses are constructed using a star schema. To fully exploit this model the Oracle kernel from 7.2 onwards will attempt to optimize star queries.

A star schema consists of a central intersection table or more commonly called “fact” table with a large number of satellites tables related to the central table. The satellite tables define the dimensions of the star query. An example of a star schema would be a central sales table that stores the value of each sale. On the fact table there would be a large number of foreign key references to the satellite tables. These could be product, product type, business period, geographical region, and sales organization. The star query is composed by referencing the outer satellite queries within the query ‘where’ clause predicates. The specification of all the join conditions insures all the tables are correctly linked. An example of the type of business question that a star query would consist of would may read as follows:

Give me the sum of sales of soft drinks sold in the states of California and Texas in the month of June from Vending machines.

This query may be written in SQL some thing like this:

```
select sum( sales ), state  
from months m, sales_line s, products p, location l,  
sales_fact f  
where f.mid = m.mid  
and f.sid = s.sid  
and f.pid = p.pid  
and f.lid = l.lid  
and m.month = 'June'  
and s.route = 'Vending Machine'  
and p.product_type = 'Soft Drinks'  
and l.state in ( 'CA', 'TX' )  
group by state;
```

This query will be executed as Cartesian join of the outer satellite tables to produce a combination product. The join to the central fact table is then completed by performing nested loop join. This is done by building an concatenated index on all the foreign keys in the fact table. The order this index is specified will require tuning according to the selectivity of the various keys.

To further speed this query if the sales column is appended to the index the whole query will be satisfied in the index without even having to look at the fact table. This query is very fast. In queries that use more than 5 tables insure that the tables are specified in join order with the fact table last and use an “ordered” hint within the query.

## **Monitoring the Data Warehouse.**

### **Bottleneck Detection.**

Bottleneck Detection is the process where by the database administrator will detect for what reason the database performance level has reached a plateau. To increase the performance from this plateau may require the addition of more hardware resources or reconfiguration of the system software( O/S, RDBMS or Application ). The following notes provide an insight as to determine the various types of bottlenecks. For more complex bottlenecks seek specialist tuning and performance assistance.

## Hardware bottlenecks

The following table describes what utilities use to determine hardware bottlenecks and the statistical information that should be interpreted from the statistics.

Hardware Resource	Monitoring Tools	Results Interpretation	Addition Comments
CPU	"sar -u", vmstat, iostat	When the CPU is fully utilized it should split  60-80% User Code  40-20% System Code	If there is no idle time and the CPU is utilized as described. The system is CPU constrained. Either buy more CPUs or rework application to use less CPU.
Memory	"sar -?", vmstat	There should be no paging or swapping activities	If paging or swapping occurs consider either reducing memory buffer sizes or if this is unacceptable buy more memory.
Disks	"sar -d", iostat	The key statistics are:- I/Os per second per device. (Should be less than 30) Device queues (should average between 1.0 and 1.5) Device response times (Should average at 0 milli sec wait time and 20-30 milli sec service time)	In addition high wait I/O value from "sar -u" will indicate I/O bottlenecks. If disk statistics look good and still high wait I/O investigate disk controller and channel bottlenecks.

## **RDBMS Bottlenecks**

Unlike hardware bottlenecks where detection is very simple RDBMS bottlenecks are more subjective and require considerable knowledge and experience of the Oracle Kernel internals. With this fact in mind this document does not attempt to address every internal Oracle bottleneck. This document describes a method that a system administrator can take effective statistics that when given to database performance specialist they can be interpreted easily.

The core database statistics to collect are those provided by the Oracle in memory performance tables. These are often referred to the V\$ and X\$ tables. There are numerous of these tables which are all documented in various Oracle documents. The most important thing is to collect these results in a format that is useful for performance analysis as well as long term performance tracking.

These statistics are best captured using a snapshot tool like the utlstat & utlestat scripts shipped with the database or some of the more sophisticated scripts shipped from the performance studies group with Oracle UK. These snapshot tools in essence measure the deltas in the database statistics over finite period of time. They then generate a report that can be used to determine bottlenecks.

The second set of statistics are the real time statistics and are useful to watch in real time as the database is showing its performance problems. The key statistical table to watch in real time is v\$session\_wait. This allows to get real time information about the state of each database process. Use of this table allows the database administrator very simply determine internal bottlenecks as any process in a wait state will be displayed with the wait event. This tool is particularly useful resolving scalability problems.